Oracle® GoldenGate

Windows and UNIX Reference Guide

11*g* Release 2 Patch Set 1 (11.2.1.0.1)

E29399-01

April 2012

Oracle GoldenGate Windows and UNIX Reference Guide 11g Release 2 Patchset 1 (11.2.1.0.1)

E29399-01

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

Typographic conventions used in this manual

This manual uses the following style conventions.

- Parameter and command arguments are shown in upper case, for example:

  ```
  CHECKPARAMS
  ```

- File names, table names, and other names are shown in lower case unless they are case-sensitive to the operating system or software application they are associated with, for example:

  ```
  account_tab
  GLOBALS
  ```

- Variables are shown within `< >` characters, for example:

  ```
  <group name>
  ```

- When one of multiple mutually-exclusive arguments must be selected, the selection is enclosed within braces and separated with pipe characters, for example:

  ```
  VIEW PARAMS {MGR | <group> | <file name>}
  ```

- Optional arguments are enclosed within brackets, for example:

  ```
  CLEANUP EXTRACT <group name> [, SAVE <count>]
  ```

- When there are numerous multiple optional arguments, a placeholder such as `[<option>]` may be used, and the options are listed and described separately, for example:

  ```
  TRANLOGOPTIONS [<option>]
  ```

- When an argument is accepted more than once, an ellipsis character (...) is used, for example:

  ```
  PARAMS ([<requirement rule>] <param spec> [, <param spec>] [, ...])
  ```

- The ampersand (&) is used as a continuation character in Oracle GoldenGate parameter files. It is required to be placed at the end of each line of a parameter statement that spans multiple lines. Most examples in this documentation show the ampersand in its proper place; however, some examples of multi-line statements may omit it to allow for space constraints of the publication format.

Getting more help with Oracle GoldenGate

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

Getting help with the Oracle GoldenGate interface

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

GGSCI commands

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

```
HELP <command name>
```

Example:

```
HELP ADD EXTRACT
```

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

Oracle GoldenGate Director and Oracle GoldenGate Monitor

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

Getting help with questions and problems

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

CHAPTER 1 Oracle GoldenGate GGSCI Commands

The Oracle GoldenGate Software Command Interface (GGSCI) is the command interface between users and Oracle GoldenGate functional components.

Command Summary

The following is a summary of the GGSCI commands.

Command Group	Purpose
Manager commands	Start and manage the Manager process.
Extract commands	Create and manage Extract groups.
Replicat commands	Create and manage Replicat groups.
ER commands	Control multiple Extract and Replicat groups as a unit.
Trail commands	Link trails to an Extract group and provide file-management parameters.
Parameter commands	Run an editor to define or alter parameters.
Database commands	Issue database-related commands.
`Trandata commands`	Configure the database to log additional information that is needed for Oracle GoldenGate to replicate UPDATE operations.
Checkpoint table commands	Create and manage the Oracle GoldenGate checkpoint table.
Oracle trace table commands	Create and manage a trace table to prevent data looping in a bidirectional configuration.
`DDL commands`	Relate to DDL synchronization.
Miscellaneous commands	Control miscellaneous functions.

Manager commands

Use Manager commands to control the Manager process. Manager is the parent process of Oracle GoldenGate and is responsible for the management of its processes and files, resources, user interface, and the reporting of thresholds and errors.

Command summary

The following is a summary of the Manager commands.

INFO MANAGER

SEND MANAGER

START MANAGER

STATUS MANAGER

STOP MANAGER

INFO MANAGER

Use INFO MANAGER to determine whether or not the Manager process is running. If Manager is running, the port number is displayed. This command is an alias for STATUS MANAGER.

Syntax

```
INFO MANAGER
```

SEND MANAGER

Use SEND MANAGER to retrieve the status of the active Manager process or to retrieve dynamic port information as configured in the Manager parameter file.

Syntax

```
SEND MANAGER
[CHILDSTATUS [DEBUG]]
[GETPORTINFO [DETAIL]
[GETPURGEOLDEXTRACTS]
```

Argument	Description
`CHILDSTATUS [DEBUG]`	Retrieves status information about processes started by Manager. DEBUG returns the port numbers that are allocated to processes.
`GETPORTINFO [DETAIL]`	By default, retrieves the current list of ports that have been allocated to processes and their corresponding process IDs. DETAIL provides additional details.
`GETPURGEOLDEXTRACTS`	Displays information about trail maintenance rules that are set with the PURGEOLDEXTRACTS parameter in the Manager parameter file. For more information about PURGEOLDEXTRACTS, see page 292.

Example 1 SEND MANAGER CHILDSTATUS DEBUG returns a child process status similar to the following. The basic CHILDSTATUS option returns the same display, without the Port column.

```
ID  Group   Process  Retry  Retry Time  Start Time           Port
1   ORAEXT  2400     0      None        2011/01/21 21:08:32  7840
2   ORAEXT  2245     0      None        2011/01/23 21:08:33  7842
```

Example 2 SEND MANAGER GETPORTINFO DETAIL returns a dynamic port list similar to the following.

```
Entry  Port   Error   Process   Assigned             Program
0      8000   0       2387      2011-01-01 10:30:23
1      8001   0
2      8002   0
```

Example 3 SEND MANAGER GETPURGEOLDEXTRACTS outputs something similar to the following.

```
PurgeOldExtracts Rules
Fileset                      MinHours MaxHours MinFiles MaxFiles UseCP
S:\GGS\DIRDAT\EXTTRAIL\P4\*         0        0        1        0     Y
S:\GGS\DIRDAT\EXTTRAIL\P2\*         0        0        1        0     Y
S:\GGS\DIRDAT\EXTTRAIL\P1\*         0        0        1        0     Y
S:\GGS\DIRDAT\REPTRAIL\P4\*         0        0        1        0     Y
S:\GGS\DIRDAT\REPTRAIL\P2\*         0        0        1        0     Y
S:\GGS\DIRDAT\REPTRAIL\P1\*         0        0        1        0     Y
OK
Extract Trails
Filename                     Oldest_Chkpt_Seqno  IsTable  IsVamTwoPhaseCommit
S:\GGS\8020\DIRDAT\RT                         3        0        0
S:\GGS\8020\DIRDAT\REPTRAIL\P1\RT            13        0        0
S:\GGS\8020\DIRDAT\REPTRAIL\P2\RT            13        0        0
S:\GGS\8020\DIRDAT\REPTRAIL\P4\RT            13        0        0
S:\GGS\8020\DIRDAT\EXTTRAIL\P1\ET            14        0        0
S:\GGS\8020\DIRDAT\EXTTRAIL\P2\ET            14        0        0
S:\GGS\8020\DIRDAT\EXTTRAIL\P4\ET            14        0        0
```

START MANAGER

Use START MANAGER to start the Manager process. This applies to a non-clustered environment. In a Windows cluster, you should stop Manager from the Cluster Administrator.

Syntax

```
START MANAGER
```

STATUS MANAGER

Use STATUS MANAGER to determine whether or not the Manager process is running. If Manager is running, the port number is displayed.

Syntax

```
STATUS MANAGER
```

STOP MANAGER

Use STOP MANAGER to stop the Manager process. This applies to non-clustered environments. In a Windows cluster, Manager must be stopped through the Cluster Administrator.

Syntax

```
STOP MANAGER [!]
```

Argument	Description
!	(Exclamation point) Bypasses the prompt that confirms the intent to shut down Manager.

Extract commands

Use Extract commands to create and manage Extract groups. The Extract process captures either full data records or transactional data changes, depending on configuration parameters, and then sends the data to a target system to be applied to target tables or processed further by another process, such as a load utility.

Command Summary

ADD EXTRACT

ALTER EXTRACT

CLEANUP EXTRACT

DELETE EXTRACT

INFO EXTRACT

KILL EXTRACT

LAG EXTRACT

REGISTER EXTRACT

SEND EXTRACT

START EXTRACT

STATS EXTRACT

STATUS EXTRACT

STOP EXTRACT

UNREGISTER EXTRACT

ADD EXTRACT

Use ADD EXTRACT to create an Extract group. Unless a SOURCEISTABLE task or an alias Extract is specified, ADD EXTRACT creates checkpoints so that processing continuity is maintained from run to run. Review the Oracle GoldenGate *Windows and UNIX Administrator's Guide* before creating an Extract group.

Command limitations

Oracle GoldenGate supports up to 5,000 concurrent Extract and Replicat groups per instance of Oracle GoldenGate Manager. At the supported level, all groups can be controlled and viewed in full with GGSCI commands such as the INFO and STATUS commands. Oracle GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at the default level of 300 or below in order to manage your environment effectively.

This command cannot exceed 500 bytes in size for all keywords and input, including any text that you enter for the DESC option.

Syntax For a regular, passive, or data pump Extract

```
ADD EXTRACT <group name>
{, SOURCEISTABLE |
   , TRANLOG [<bsds name>] |
   , INTEGRATED TRANLOG |
   , VAM |
   , EXTFILESOURCE <file name> |
   , EXTTRAILSOURCE <trail name> |
   , VAMTRAILSOURCE <VAM trail name>}
{, BEGIN {NOW | yyyy-mm-dd [:hh:mi:[ss[.cccccc]]]} |
   , EXTSEQNO <seqno>, EXTRBA <relative byte address> |
   , LOGNUM <log number>, LOGPOS <byte offset> |
   , EOF |
   , LSN <value> |
   , EXTRBA <relative byte address> |
   , EOF | LSN <value> |
   , PAGE <data page>, ROW <row> |
   }
[, THREADS <n>]
[, PASSIVE]
[, PARAMS <parameter file>]
[, REPORT <report file>]
[, DESC “<description>”]
```

Syntax For an alias Extract

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

Argument	Description
`<group name>`	The name of the Extract group. For group naming conventions, see the Oracle GoldenGate *Windows and UNIX Reference Guide.*

Argument	Description
`SOURCEISTABLE`	Creates an Extract task that extracts entire records from the database for an initial load using the Oracle GoldenGate direct load method or the direct bulk load to SQL*Loader method. If SOURCEISTABLE is not specified, ADD EXTRACT creates an online change-synchronization process, and one of the other data source options must be specified. When using SOURCEISTABLE, do not specify any service options. Task parameters must be specified in the parameter file. For more information about initial load methods, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

`TRANLOG`
`[<bsds name>]`
	Specifies the transaction log as the data source. Use this option for all databases except Teradata. TRANLOG requires the BEGIN option.
	(DB2 on z/OS) Use the <bsds name> option for DB2 on a z/OS system to specify the Bootstrap Data Set (BSDS) file name of the transaction log. Make certain that the BSDS name you provide is the one *for the DB2 instance to which the Extract process is connected*. Oracle GoldenGate does not perform any validations of the BSDS specification.
	(Oracle) As of Oracle Standard or Enterprise Edition 11.2.0.2, this mode is known as *classic capture* mode. Extract reads the Oracle redo logs directly. See also INTEGRATED TRANLOG for an alternate configuration.

`INTEGRATED TRANLOG`	Adds this Extract in integrated capture mode. In this mode, Extract integrates with the database logmining server, which passes logical change records (LCR) directly to Extract. Extract does not read the redo log. Before using INTEGRATED TRANLOG, use the DBLOGIN or MININGDBLOGIN command to log into the source database or downstream mining database, and then the REGISTER EXTRACT command to register the Extract with that database. For information about integrated capture, see the Oracle GoldenGate *Oracle Installation and Setup Guide.*
`VAM`	(Teradata) Specifies that the Extract API known as the *Vendor Access Module* (VAM) will interface with the Teradata Access Module (TAM). Use for Teradata databases.

`EXTFILESOURCE`
`<file name>`
	Specifies an extract file as the data source. Use this option with a secondary Extract group (data pump) that acts as an intermediary between a primary Extract group and the target system.
	For <file name>, specify the relative or fully qualified path name of the file, for example dirdat\extfile or c:\ggs\dirdat\extfile.

Argument	Description
`EXTTRAILSOURCE <trail name>`	Specifies a trail as the data source. Use this option with a secondary Extract group (data pump) that acts as an intermediary between a primary Extract group and the target system. For <trail name>, specify the relative or fully qualified path name of the trail, for example dirdat\aa or c:\ggs\dirdat\aa.
`VAMTRAILSOURCE <VAM trail name>`	(Teradata) Specifies a VAM trail. Use this option when using Teradata maximum protection mode. For <VAM trail name>, specify the relative or fully qualified path name of the VAM trail to which the primary Extract group is writing. Use a VAM-sort Extract group to read the VAM trail and send the data to the target system.
`BEGIN {NOW | yyyy-mm-dd [:hh:mi:[ss[.cccccc]]]}`	Specifies a timestamp in the data source at which to begin processing. Valid values: ◆ NOW ◆ A date and time in the format of: `yyyy-mm-dd[:hh:mi:[ss[.cccccc]]]` **What NOW means:** For all databases except DB2 LUW, NOW specifies the time at which the ADD EXTRACT command is issued. For DB2 LUW, NOW specifies the time at which START EXTRACT takes effect. It positions to the first record that *approximately* matches the date and time. This is because the only log records that contain timestamps are the commit and abort transaction records, so the starting position can only be calculated relative to those timestamps. This is a limitation of the API that is used by Oracle GoldenGate. Do not use NOW for a data pump Extract except to bypass data that was captured to the trail prior to the ADD EXTRACT statement. **Positioning by timestamp in a SQL Server transaction log** Positioning by time is affected by the following limitations of SQL Server: ◆ The timestamps recorded in the SQL Server transaction log use a 3.3333 microsecond (ms) granularity. This level of granularity may not allow positioning by time between two transactions, if the transactions began in the same 3.3333 ms time interval. ◆ Timestamps are not recorded in every SQL Server log record, but only in the records that begin and commit the transaction, as well as some others that do not contain data.

Argument	Description

	◆ SQL Server timestamps are not from the system clock, but instead are from an internal clock that is specific to the individual processors in use. This clock updates several times a second, but between updates it could get out of sync with the system clock. This further reduces the precision of positioning by time.
	◆ Timestamps recorded for log backup files may not precisely correspond to times recorded inside the backup (however this imprecision is less than a second).
	Positioning to an LSN is precise. See <LSN <value>.
	Positioning by timestamp in a Sybase transaction log
	Sybase only records timestamps in BEGIN and COMMIT records. Regardless of the actual timestamp that is specified, the start position will be the first record of the transaction that starts closest to, or at, the specified timestamp. The Extract report will display the following positions:
	Positioning To: This is the specified begin time, for example:
	`Positioning to begin time Jan 1, 2011 12:13:33 PM.`
	Positioned To: If the specified timestamp is less than, or equal to, the timestamp of the transaction log that contains the BEGIN or COMMIT record, "Positioned To Page" is displayed as in this example:
	`2011-01-01 12:13:39  INFO    OGG-01516  Positioned to`
	`Page #: 0004460243`
	`Row #: 00111, Jan 1, 2011 12:13:38 PM.`
	First Record Position: This is the position of the first valid record at, or after, the **Positioned To** position, as in this example:
	`2011-01-01 12:13:39  INFO    OGG-01517  Position of first`
	`record processed`
	`Page #: 0004460243`
	`Row #: 00111, Jan 1, 2011 12:13:38 PM.`

`EXTSEQNO <seqno>, EXTRBA <relative byte address>`	Valid for a primary Extract for Oracle and NonStop SQL/MX, and for a data pump Extract. Specifies either of the following: ◆ sequence number of an Oracle redo log and RBA within that log at which to begin capturing data. ◆ the NonStop SQL/MX TMF audit trail sequence number and relative byte address within that file at which to begin capturing data. Together these specify the location in the TMF Master Audit Trail (MAT). ◆ the file in a trail in which to begin capturing data (for a data pump). Specify the sequence number, but not any zeroes used for padding. For example, if the trail file is c:\ggs\dirdat\aa000026, you would specify EXTSEQNO 26. By default, processing begins at the beginning of a trail unless this option is used. Contact Oracle Support before using this option. For more information, go to http://support.oracle.com.

Argument	Description
`EXTRBA <relative byte address>`	Valid for DB2 on z/OS. Specifies the relative byte address within a transaction log at which to begin capturing data.
`EOF`	Valid for SQL Server. Configures processing to start at the end of the log files that the next record will be written to. Any active transactions will not be captured.

`LSN <value>`	Valid for SQL Server. Specifies the LSN in a SQL Servertransaction log at which to start capturing data. The specified LSN should exist in a log backup or the online log. An alias for this option is EXTLSN.
	For SQL Server, an LSN is composed of one of these, depending on how the database returns it:
	1. Colon separated Hex string (8:8:4) padded with leading zeroes and 0X prefix, as in 0X00000d7e:0000036b:01bd
	2. Colon separated Decimal string (10:10:5) padded with leading zeroes, as in 0000003454:0000000875:00445
	3. Colon separated Hex string with 0X prefix and without leading zeroes, as in 0Xd7e:36b:1bd
	4. Colon separated Decimal string without leading zeroes, as in 3454:875:445
	5. Decimal string, as in 3454000000087500445
	In the preceding, 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.
	You can find the LSN for named transactions by using a query like:
	`select [Current LSN], [Transaction Name], [Begin Time]`
	`  from fn_dblog(null, null)`
	` where Operation = 'LOP_BEGIN_XACT'`
	`   and [Begin Time] >= <time>`
	You can determine the time that a particular transaction started, then find the relevant LSN, and then position between two transactions with the same begin time.

`EOF | LSN <value>`	Valid for DB2 LUW. Specifies a start position in the transaction logs when Extract starts. ◆ EOF configures processing to start at the active LSN in the log files. The active LSN is the position at the end of the log files that the next record will be written to. Any active transactions will not be captured.

Argument	Description
	◆ LSN <value> configures processing to start at an exact LSN if a valid log record exists there. If one does not exist, Extract will abend. Note that, although Extract might position to a given LSN, that LSN might not necessarily be the first one that Extract will process. There are numerous record types in the log files that Extract ignores, such as DB2 internal log records. Extract will report the actual starting LSN to the Extract report file.

`PAGE <data page>,`
`ROW <row>`
	Valid for Sybase. Specifies a data page and row that together define a start position in a Sybase transaction log. Because the start position must be the first record of the transaction that starts closest to, or at, the specified PAGE and ROW, the Extract report will display the following positions:
	◆ **Positioning To** is the position of the record that is specified with PAGE and ROW.
	◆ **Positioned To** is the position where the first BEGIN record is found at, or after, the Positioning To position.
	◆ **First Record Position** is the position of the first valid record at, or after, the Positioned To position.

`PARAMS`
`<parameter file>`
	Specifies the full path name of an Extract parameter file in a location other than the default of dirprm within the Oracle GoldenGate directory.

`REPORT`
`<report file>`
	Specifies the full path name of an Extract report file in a location other than the default of dirrpt within the Oracle GoldenGate directory.

`THREADS <n>`	Specifies the number of producer threads that Extract maintains to read redo logs. ◆ For integrated capture, this value must be 1. This adds a producer thread to the Extract structures which parses and filters the logical change records that are sent by the Oracle logmining server, and then sends the data to the Extract memory cache. The consumer thread reads a memory queue to obtain the committed transactions, and then it handles formatting, fetching, checkpointing, metadata, and writing to the trail. By decoupling these responsibilities into two asynchronous threads, with memory storage and queueing, Extract performance is increased. You can set the size of the queue with the OUTQUEUESIZE option of the THREADOPTIONS parameter. ◆ For classic capture mode, use in an Oracle RAC configuration to specify the number of producer threads. These are the Extract threads that read the different redo logs on the various RAC nodes. The value must be the same as the number of nodes from which you want to capture redo data.

Argument	Description
PASSIVE	Specifies that this Extract group runs in passive mode and can only be started and stopped by starting or stopping an alias Extract group on the target system. Source-target connections will be established not by this group, but by the alias Extract from the target. This option can be used for a regular Extract group or a data-pump Extract group. It should only be used by whichever Extract on the source system is the one that will be sending the data across the network to a remote trail on the target. For instructions on how to configure passive and alias Extract groups, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.
DESC "<description>"	Specifies a description of the group, such as "Extracts account_tab on Serv1". The description must be within quotes. You may use the abbreviated keyword DESC or the full word DESCRIPTION.
RMTHOST {<host name> | <IP address>}	Identifies this group as an alias Extract and specifies either the DNS name of the remote host or its IP address.
MGRPORT <port>	Use for an alias Extract to specify the port on the remote system where Manager is running.
RMTNAME <name>	Use for an alias Extract. Specifies the passive Extract name, if different from that of the alias Extract.

ADD EXTRACT examples

Example 1 The following creates an Extract group named "finance" that extracts database changes from the transaction logs. Extraction starts with records generated at the time when the group was created with ADD EXTRACT.

```
ADD EXTRACT finance, TRANLOG, BEGIN NOW
```

Example 2 The following creates an Extract group named "finance" that extracts database changes from Oracle RAC logs. Extraction starts with records generated at the time when the group was created. There are four RAC instances, meaning there will be four Extract threads.

```
ADD EXTRACT finance, TRANLOG, BEGIN NOW, THREADS 4
```

Example 3 The following creates an Extract group named "finance" that extracts database changes from the transaction logs. Extraction starts with records generated at 8:00 on January 21, 2011.

```
ADD EXTRACT finance, TRANLOG, BEGIN 2011-01-21:08:00
```

Example 4 The following creates an integrated capture Extract group.

```
ADD EXTRACT finance, INTEGRATED TRANLOG, BEGIN NOW
```

Example 5 The following creates an Extract group named “finance” that interfaces with a Teradata TAM in either maximum performance or maximum protection mode. No BEGIN point is used for Teradata sources.

```
ADD EXTRACT finance, VAM
```

Example 6 The following creates a VAM-sort Extract group named “finance.” The process reads from the VAM trail /ggs/dirdat/vt.

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

Example 7 The following creates a data-pump Extract group named “finance.” It reads from the Oracle GoldenGate trail c:\ggs\dirdat\lt.

```
ADD EXTRACT finance, EXTTRAILSOURCE dirdat\lt
```

Example 8 The following creates an initial-load Extract named “load.”

```
ADD EXTRACT load, SOURCEISTABLE
```

Example 9 The following creates a passive Extract group named “finance” that extracts database changes from the transaction logs.

```
ADD EXTRACT finance, TRANLOG, BEGIN NOW, PASSIVE
```

Example 10 The following creates an alias Extract group named “financeA.” The alias Extract is associated with a passive extract named “finance” on source system sysA. The Manager on that system is using port 7800.

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

ALTER EXTRACT

Use ALTER EXTRACT for the following purposes:

- To change the attributes of an Extract group created with the ADD EXTRACT command.
- To increment a trail to the next file in the sequence.
- To upgrade to an integrated capture configuration.
- To downgrade from an integrated capture configuration.

Before using this command, stop Extract with the STOP EXTRACT <group name> command.

Syntax

```
ALTER EXTRACT <group name>
[, <ADD EXTRACT attribute>]
[, UPGRADE INTEGRATED TRANLOG]
[, DOWNGRADE INTEGRATED TRANLOG]
[, THREAD <number>]
[, ETROLLOVER]
```

Argument	Description
`<group name>`	The name of the Extract group that is to be altered.

Argument	Description

`<ADD EXTRACT attribute>`	You can change any of the attributes specified with the ADD EXTRACT command, except for the following:
	◆ Altering an Extract specified with the EXTTRAILSOURCE option.
	◆ Altering the number of RAC threads specified with the THREADS option.
	For these exceptions, delete the Extract group and then add it again.
	If using the BEGIN option, do not combine other options in the statement. Issue separate statements, for example:
	`ALTER EXTRACT finance, BEGIN 2011-01-01`
	`ALTER EXTRACT finance, ETROLLOVER`

`UPGRADE INTEGRATED TRANLOG`	Upgrades the Extract group from classic capture to integrated capture. Before using this command, use the DBLOGIN to connect to the source Oracle database or the MININGDBLOGIN command to connect to the downstream mining database. After issuing this command, issue REGISTER EXTRACT with the DATABASE argument to register Extract with that database. To support the upgrade, the transaction log that contains the start of the oldest open transaction must be available on the source or downstream mining system. For information about integrated capture, see the documentation for Oracle database.
`DOWNGRADE INTEGRATED TRANLOG`	Downgrades the Extract group from integrated capture to classic capture. Before using this command, use the DBLOGIN to connect to the source Oracle database or the MININGDBLOGIN command to connect to the downstream mining database. After issuing this command, issue UNREGISTER EXTRACT with the DATABASE argument to unregister Extract from that database. To support the downgrade, the transaction log that contains the start of the oldest open transaction must be available on the source or downstream mining system. For information about integrated capture, see the Oracle GoldenGate documentation for Oracle database.
`THREAD <number>`	In an Oracle RAC configuration, alters Extract only for the specified redo thread. Only one thread number can be specified.
`ETROLLOVER`	Causes Extract to increment to the next file in the trail sequence when restarting. For example, if the current file is ET000002, the current file will be ET000003 when Extract restarts. A trail can be incremented from 000001 through 999999, and then the sequence numbering starts over at 000000.

Example 1 The following alters Extract to start processing data from January 1, 2011.

```
ALTER EXTRACT finance, BEGIN 2011-01-01
```

Example 2 The following alters Extract to start processing at a specific location in the trail.

```
ALTER EXTRACT finance, EXTSEQNO 26, EXTRBA 338
```

Example 3 The following alters Extract in an Oracle RAC environment, and applies the new begin point only for redo thread 4.

```
ALTER EXTRACT accounts, THREAD 4, BEGIN 2011-01-01
```

Example 4 The following alters Extract in a SQL Server environment to start at a specific LSN.

```
ALTER EXTRACT sales, LSN 3454:875:445
```

Example 5 The following alters Extract to increment to the next file in the trail sequence.

```
ALTER EXTRACT finance, ETROLLOVER
```

Example 6 The following alters Extract to upgrade to integrated capture.

```
ALTER EXTRACT finance, UPGRADE INTEGRATED TRANLOG
```

Example 7 The following alters Extract to downgrade to classic capture.

```
ALTER EXTRACT finance, DOWNGRADE INTEGRATED TRANLOG
```

CLEANUP EXTRACT

Use CLEANUP EXTRACT to delete run history for the specified Extract group. The cleanup keeps the last run record intact so that Extract can resume processing from where it left off. Before using this command, stop Extract by issuing the STOP EXTRACT command.

Syntax

```
CLEANUP EXTRACT <group name> [, SAVE <count>]
```

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* cleans up all Extract groups whose names start with T.
`SAVE <count>`	Excludes the specified number of the most recent records from the cleanup.

Example 1 The following deletes all but the last record.

```
CLEANUP EXTRACT finance
```

Example 2 The following deletes all but the most recent five records.

```
CLEANUP EXTRACT *, SAVE 5
```

DELETE EXTRACT

Use DELETE EXTRACT to delete an Extract group. This command deletes the checkpoint file that belongs to the group, but leaves the parameter file intact. You can then re-create the group or delete the parameter file as needed.

Before using DELETE EXTRACT, stop Extract with the STOP EXTRACT command.

To delete the trail files that are associated with the Extract group, delete them manually through the operating system.

Syntax

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

Argument	Description
`<group name>`	The name of an Extract group or a wildcard specification (*) to specify multiple groups. For example, T* deletes all Extract groups whose names start with T.
`!`	(Exclamation point) Deletes all Extract groups associated with a wildcard without prompting.

INFO EXTRACT

Use INFO EXTRACT to view the following information.

- Status of Extract (STARTING, RUNNING, STOPPED or ABENDED).
- Approximate Extract lag.
- Checkpoint information.
- Process run history.
- The trail(s) to which Extract is writing.
- Status of upgrade to, or downgrade from, integrated capture

Extract can be running or stopped when INFO EXTRACT is issued. In the case of a running process, the status of RUNNING can mean one of the following:

- Active: Running and processing (or able to process) data. This is the normal state of a process after it is started.
- Suspended: The process is running, but suspended due to an EVENTACTIONS SUSPEND action. In a suspended state, the process is not active, and no data can be processed, but the state of the current run is preserved and can be continued by issuing the RESUME command in GGSCI. The RBA in the INFO command reflects the last checkpointed position before the suspend action. To determine whether the state is active or suspended, issue the SEND EXTRACT command with the STATUS option.

The basic command, without either the TASKS or ALLPROCESSES argument, displays information only for online (continuous) Extract processes. Tasks are excluded.

The following is an example:

Figure 1 INFO EXTRACT

```
EXTRACT                    EXTCUST Last Started 2011-01-05 16:09 Status RUNNING
Checkpoint Lag             00:01:30 (updated 97:16:45 ago)
Log Read Checkpoint File   /rdbms/data/oradata/redo03a.log
                           2011-01-05 16:05:17  Seqno 2952, RBA 7598080
```

About lag

Checkpoint Lag reflects the lag, in seconds, at the time that the last checkpoint was written to the trail. For example, if the following is true...

- Current time = 15:00:00
- Last checkpoint = 14:59:00
- Timestamp of the last record processed = 14:58:00

...then the lag is reported as 00:01:00 (one minute, the difference between 14:58 and 14:59).

A lag value of UNKNOWN indicates that the process could be running but has not yet processed records, or that the source system's clock is ahead of the target system's clock (due to clock imperfections, not time zone differences).

For more precise lag information, use LAG EXTRACT (see page 34).

Showing detail

The following is an example of output for the DETAIL option.

Figure 2 INFO EXTRACT with DETAIL

```
EXTRACT    ORAEXT    Last Started 2011-01-15 16:16   Status STOPPED
Checkpoint Lag       00:00:00 (updated 114:24:48 ago)
Log Read Checkpoint  File C:\ORACLE\ORADATA\ORA920\REDO03.LOG
                     2011-01-15 16:17:53 Seqno 46, RBA 3757568

Target Extract Trails:

Remote Trail Name                      Seqno    RBA   Max MB

  c:\goldengate802\dirdat\xx               0  57465       10
  c:\goldengate802\dirdat\jm               0  19155       10

Extract Source                         Begin             End

C:\ORACLE\ORADATA\ORA920\REDO03.LOG     2011-01-15 16:07 2011-01-15 16:17
C:\ORACLE\ORADATA\ORA920\REDO03.LOG    2011-01-15 15:55  2011-01-15 16:07
C:\ORACLE\ORADATA\ORA920\REDO03.LOG    2011-01-15 15:42  2011-01-15 15:55
C:\ORACLE\ORADATA\ORA920\REDO03.LOG   2011-01-15 15:42  2011-01-15 15:42
  Not Available                        * Initialized *   2011-01-15 15:42

Current directory    C:\GoldenGate802

Report file          C:\GoldenGate802\dirrpt\ORAEXT.rpt
Parameter file       C:\GoldenGate802\dirprm\ORAEXT.prm
Checkpoint file      C:\GoldenGate802\dirchk\ORAEXT.cpe
Process file         C:\GoldenGate802\dirpcs\ORAEXT.pce
Error log            C:\GoldenGate802\ggserr.log
```

Showing checkpoints

Extract checkpoint positions are composed of read checkpoints in the data source and write checkpoints in the trail. The following is a sampling of checkpoint information displayed with the SHOWCH option. In this case, the data source is an Oracle RAC database cluster, so there is thread information included in the output. You can view past checkpoints by specifying the number of them that you want to view after the SHOWCH entry.

Figure 3 INFO EXTRACT with SHOWCH

```
EXTRACT    JC108XT Last Started 2011-01-01 14:15   Status ABENDED
Checkpoint Lag       00:00:00 (updated 00:00:01 ago)
Log Read Checkpoint  File /orarac/oradata/racq/redo01.log
          2011-01-01 14:16:45  Thread 1, Seqno 47, RBA 68748800
Log Read Checkpoint  File /orarac/oradata/racq/redo04.log
          2011-01-01 14:16:19  Thread 2, Seqno 24, RBA 65657408

Current Checkpoint Detail:

Read Checkpoint #1

   Oracle RAC Redo Log

   Startup Checkpoint (starting position in data source):
       Thread #: 1
       Sequence #: 47
       RBA: 68548112
       Timestamp: 2011-01-01 13:37:51.000000
       SCN: 0.8439720
       Redo File: /orarac/oradata/racq/redo01.log

   Recovery Checkpoint (position of oldest unprocessed transaction in
   data source):
       Thread #: 1
       Sequence #: 47
       RBA: 68748304
       Timestamp: 2011-01-01 14:16:45.000000
       SCN: 0.8440969
       Redo File: /orarac/oradata/racq/redo01.log

   Current Checkpoint (position of last record read in the data source):
       Thread #: 1
       Sequence #: 47
       RBA: 68748800
       Timestamp: 2011-01-01 14:16:45.000000
       SCN: 0.8440969
       Redo File: /orarac/oradata/racq/redo01.log

Read Checkpoint #2

   Oracle RAC Redo Log
   Startup Checkpoint(starting position in data source):
       Sequence #: 24
       RBA: 60607504
       Timestamp: 2011-01-01 13:37:50.000000
       SCN: 0.8439719
       Redo File: /orarac/oradata/racq/redo04.log

   Recovery Checkpoint (position of oldest unprocessed transaction in
   data source):
       Thread #: 2
       Sequence #: 24
       RBA: 65657408
       Timestamp: 2011-01-01 14:16:19.000000
       SCN: 0.8440613
       Redo File: /orarac/oradata/racq/redo04.log

   Current Checkpoint (position of last record read in the data source):
       Thread #: 2
       Sequence #: 24
       RBA: 65657408
       Timestamp: 2011-01-01 14:16:19.000000
       SCN: 0.8440613
       Redo File: /orarac/oradata/racq/redo04.log

Write Checkpoint #1

   GGS Log Trail

   Current Checkpoint (current write position):

       Sequence #: 2
       RBA: 2142224
       Timestamp: 2011-01-01 14:16:50.567638
       Extract Trail: ./dirdat/eh

   Header:
       Version = 2
       Record Source = A
       Type = 6
       # Input Checkpoints = 2
       # Output Checkpoints = 1

   File Information:
       Block Size = 2048
       Max Blocks = 100
       Record Length = 2048
       Current Offset = 0

   Configuration:
       Data Source = 3
       Transaction Integrity = 1
       Task Type = 0
```

```
Status:
   Start Time = 2011-01-01 14:15:14
   Last Update Time = 2011-01-01 14:16:50
   Stop Status = A
   Last Result = 400
```

About Extract read checkpoints

Extract places a read checkpoint in the data source.

Startup checkpoint

The startup checkpoint is the first checkpoint that is made in the data source when the process starts. This statistic is composed of the following:

- Thread #: The number of the Extract thread that made the checkpoint, if Oracle GoldenGate is running in an Oracle RAC environment. Otherwise, this statistic is not displayed.
- Sequence #: The sequence number of the transaction log where the checkpoint was made.
- RBA: The relative byte address of the record at which the checkpoint was made.
- Timestamp: The timestamp of the record at which the checkpoint was made.
- SCN: The system change number of the record at which the checkpoint was made.
- Redo File: The path name of the transaction log containing the record where the checkpoint was made.

Recovery checkpoint

The recovery checkpoint is the position in the data source of the record containing the oldest transaction not yet processed by Extract. The fields for this statistic are the same as those of the other read checkpoint types.

Current checkpoint

The current checkpoint is the position of the last record read by Extract in the data source. This should match the Log Read Checkpoint statistic shown in the summary and in the basic INFO EXTRACT command without options. The fields for this statistic are the same as those of the other read checkpoint types.

About Extract write checkpoints

Extract places a write checkpoint in the trail.

Current checkpoint

The current checkpoint is the position in the trail where Extract is currently writing. This statistic is composed of the following:

- Sequence #: The sequence number of the trail file where the checkpoint was written.
- RBA: The relative byte address of the record in the trail file at which the checkpoint was made.
- Timestamp: The timestamp of the record at which the checkpoint was made.
- Extract trail: The relative path name of the trail.

Other SHOWCH information

The Header, File Information, Configuration, and Status statistics at the end of the SHOWCH display are for use by Oracle Support analysts. They contain internal information that is useful when resolving a support case.

Syntax

```
INFO EXTRACT <group name>
[, SHOWCH [<n>]]
[, DETAIL]
[, TASKS | ALLPROCESSES]
[, UPGRADE | DOWNGRADE]
```

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* shows information for all Extract groups whose names start with T.
`SHOWCH [<n>]`	The basic command shows information about the current Extract checkpoints. Specify a value for <n> to include the specified number of previous checkpoints as well as the current one. Note: You might see irregular indents and spacing in the output. This is normal and does not affect the accuracy of the information.
`DETAIL`	Displays the following: ◆ Extract run history, including start and stop points in the data source, expressed as a time. ◆ Trails to which Extract is writing.
`TASKS`	Displays only Extract tasks. Tasks that were specified by a wildcard argument are not displayed by INFO EXTRACT.
`ALLPROCESSES`	Displays all Extract groups, including tasks.
`UPGRADE | DOWNGRADE`	◆ UPGRADE displays whether the Extract can be upgraded from classic capture mode to integrated capture mode. ◆ DOWNGRADE displays whether the Extract can be downgraded from integrated capture mode to classic capture mode. If Extract cannot be upgraded or downgraded, the reason why is displayed. A wildcarded Extract name is not allowed with this option. Before using this command, issue the DBLOGIN command.

Example 1 `INFO EXTRACT fin*, SHOWCH`

Example 2 `INFO EXTRACT *, TASKS`

Example 3 `INFO EXTRACT finance UPGRADE`

KILL EXTRACT

Use KILL EXTRACT to kill an Extract process running in regular or PASSIVE mode. Use this command only if a process cannot be stopped gracefully with the STOP EXTRACT command. The Manager process will not attempt to restart a killed Extract process.

Syntax

```
KILL EXTRACT <group name>
```

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* kills all Extract processes whose group names start with T.

Example

```
KILL EXTRACT finance
```

LAG EXTRACT

Use LAG EXTRACT to determine a true lag time between Extract and the data source. LAG EXTRACT calculates the lag time more precisely than INFO EXTRACT because it communicates with Extract directly, rather than reading a checkpoint position in the trail.

About Extract 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.

The following is sample output for LAG EXTRACT.

Figure 4 LAG EXTRACT output

```
Sending GETLAG request to EXTRACT CAPTPCC...
Last record lag: 2 seconds.
At EOF, no more records to process.
```

Syntax

```
LAG EXTRACT <group name>
```

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* determines lag time for all Extract groups whose names start with T.

Example 1

```
LAG EXTRACT *
```

Example 2

```
LAG EXTRACT *fin*
```

REGISTER EXTRACT

Use REGISTER EXTRACT to register a primary Extract group with an Oracle database to:

- Enable integrated capture mode.
- Enable Extract in classic capture mode to work with Oracle Recovery Manager to retain the archive logs needed for recovery.

REGISTER EXTRACT is not valid for a data pump Extract.

To unregister en Extract group from the database, use the UNREGISTER EXTRACT command (see page 52).

Syntax

```
REGISTER EXTRACT <group name>
{DATABASE | LOGRETENTION}
```

Argument	Description
`<group name>`	The name of the Extract group that is to be registered. Do not use a wildcard.
`DATABASE`	Enables integrated capture for the Extract group. In this mode, Extract integrates with the database logmining server to receive change data in the form of logical change records (LCR). Extract does not read the redo logs. Extract performs capture processing, filtering, transformation, and other requirements. For support information and configuration steps, see the Oracle GoldenGate documentation for the Oracle database. Before using REGISTER EXTRACT with DATABASE, use the DBLOGIN or MININGDBLOGIN command with the privileges granted through the dbms_goldengate_auth.grant_admin_privilege procedure. See those comands for additional requirements. After using REGISTER EXTRACT, use ADD EXTRACT with the INTEGRATED TRANLOG option to create an Extract group of the same name. If REGISTER EXTRACT with LOGRETENTION was previously used for this Extract, it is automatically unregistered for log retention because the log files are managed as part of integrated capture.
`LOGRETENTION`	Enables an Extract group in classic capture mode to work with Oracle Recovery Manager (RMAN) to retain the logs that Extract needs for recovery. LOGRETENTION creates an underlying Oracle Streams capture process that: ◆ is dedicated to the Extract group and only used for the purpose of log retention ◆ has a similar name LOGRETENTION is ignored if the Extract group is configured for integrated capture.

Argument	Description
	The logs are retained from the time that REGISTER EXTRACT is issued, based on the current database SCN. The log-retention feature is controlled with the LOGRETENTION option of the TRANLOGOPTIONS parameter. Before using REGISTER EXTRACT with LOGRETENTION, issue the DBLOGIN command with the privileges shown in Table 1 on page 78.

Example 1 REGISTER EXTRACT sales LOGRETENTION

Example 2 REGISTER EXTRACT sales DATABASE

SEND EXTRACT

Use SEND EXTRACT to communicate with a running Extract process. The request is processed as soon as Extract is ready to accept commands from users.

Syntax

```
SEND EXTRACT <group name>, {
BR {BRINTERVAL <interval> |
   BRSTART |
   BRSTOP |
   BRCHECKPOINT {IMMEDIATE | IN <N>{H|M} | AT YYYY-MM-DD HH:MM[:SS]}} |
CACHEMGR {CACHESTATS | CACHEQUEUES | CACHEPOOL} |
FORCESTOP |
FORCETRANS <Transaction ID> [THREAD <n>] [FORCE] |
GETLAG |
GETTCPSTATS |
LOGEND |
LOGSTATS |
REPORT |
RESUME |
ROLLOVER |
SHOWTRANS [<Transaction ID>] [THREAD <n>] [COUNT <n>]
   [DURATION <duration><unit>] [TABULAR]
   [FILE <name> [DETAIL]] |
SKIPTRANS <Transaction ID> [THREAD <n>] [FORCE] |
STATUS |
STOP |
TRACE[2] <tracefile> |
TRACE[2] OFF |
TRACE OFF <file name> |
TRACEINIT |
TRANLOGOPTIONS {PURGEORPHANEDTRANSACTIONS | NOPURGEORPHANEDTRANSACTIONS} |
TRANLOGOPTIONS TRANSCLEANUPFREQUENCY <minutes> |
VAMMESSAGE "<Teradata command>" |
VAMMESSAGE {"ARSTATS" | "INCLUDELIST [filter]" | "EXCLUDELIST [filter]"} |
VAMMESSAGE "OPENTRANS"
}
```

Argument	**Description**
`<group name>`	The name of the Extract group or a wildcard (*) to specify multiple groups. For example, T* sends the command to all Extract processes whose group names start with T. If an Extract is not running, an error is returned.

`BR {BRINTERVAL <interval> |`
`BRSTART |`
`BRSTOP |`
`BRSTATS |`
`BRCHECKPOINT {IMMEDIATE |`
`IN <N>{H|M} |`
`AT YYYY-MM-DD HH:MM[:SS]}}`
	Sends commands that affect the Bounded Recovery mode of Extract.
	BRINTERVAL <interval>
	Sets the time between Bounded Recovery checkpoints. Valid values are from 20 minutes to 96 hours specified as M for minutes or H for hours. The default interval is 4 hours.
	BRSTART
	Starts Bounded Recovery. This command should only be used under direction of Oracle Support.
	BRSTOP
	Stops Bounded Recovery for the run and for recovery. Consult Oracle Support before using this option. In most circumstances, when there is a problem with Bounded Recovery, it turns itself off.
	BRCHECKPOINT {IMMEDIATE | IN <N>{H|M} | AT YYYY-MM-DD HH:MM[:SS]}}
	Sets the point at which a bounded recovery checkpoint is made.
	◆ IMMEDIATE causes Extract to issue the checkpoint immediately when SEND EXTRACT is issued.
	◆ IN <N>{H | M} causes Extract to issue the checkpoint in the specified number of hours or minutes from when SEND EXTRACT is issued.
	◆ AT YYYY-MM-DD HH:MM[:SS]}} causes Extract to issue the checkpoint at exactly the specified time.

`CACHEMGR {CACHESTATS |`
`CACHEQUEUES | CACHEPOOL <n>}`
	Returns statistics about the Oracle GoldenGate memory cache manager.
	◆ CACHESTATS returns CACHEMGR statistics for virtual memory usage and file caching.
	◆ CACHEQUEUES returns statistics for the CACHEMGR free queues only.
	◆ CACHEPOOL <n> return statistics for the specified object pool only.
	CACHESTATS should only be used as explicitly directed by Oracle Support.

Argument	Description
`FORCESTOP`	Forces Extract to stop, bypassing any notifications. This command will stop the process immediately.

`FORCETRANS <Transaction ID>`
`[THREAD <n>]`
`[FORCE]`
	(MySQL, Oracle, SQL Server, Sybase) Forces Extract to write a transaction specified by its Transaction ID number to the trail as a committed transaction. Get the Transaction ID number with SHOWTRANS or from an Extract runtime message. Extract will ignore any data added to the transaction after this command is issued. A confirmation prompt must be answered unless FORCE is used. In order to use FORCETRANS, the transaction specified must be the oldest one in the list of transactions shown with SHOWTRANS.
	Options:
	◆ Use THREAD <n> to specify which thread generated the transaction in an Oracle RAC environment if there are duplicate transaction IDs across threads.
	◆ Use FORCE to bypass the confirmation prompt.
	FORCETRANS does not commit the transaction to the source database. It only forces the existing data to the trail so that it is processed (with an implicit commit) by Replicat.
	You can repeat the command for other transactions in order of their age.
	After using FORCETRANS, wait at least five minutes if you intend to issue SEND EXTRACT with FORCESTOP. Otherwise, the transaction will still be present.
	If FORCETRANS is used immediately after Extract starts, you might receive an error message that asks you to wait and then try the command again. This means that no other transactions have been processed yet by Extract. Once another transaction is processed, you will be able to force the transaction to trail.

`GETLAG`	Determines a true lag time between Extract and the data source. Returns the same results as LAG EXTRACT (see page 34).
`GETTCPSTATS`	Displays statistics about network activity between Extract and the target system. The statistics include: ◆ Local and remote IP addresses. ◆ Inbound and outbound messages, in bytes and bytes per second. ◆ Number of receives (inbound) and sends (outbound). There will be at least two receives per inbound message: one for the length and one or more for the data.

Argument	Description
	◆ Average bytes per send and receive. ◆ Send and receive wait time: Send wait time is how long it takes for the write to TCP to complete. The lower the send wait time, the better the performance over the network. Receive wait time is how long it takes for a read to complete. Together, the send and receive wait times provide a rough estimate of network round trip time. These are expressed in microseconds. ◆ Status of data compression (enabled or not). ◆ Uncompressed bytes and compressed bytes: When compared (uncompressed to compressed), these comprise the compression ratio, meaning how many bytes there were before and after compression. You can compare the compression ratio with the bytes that are being compressed per second to determine if the compression rate is worth the cost in terms of resource and network consumption. The TCPBUFSIZE option of RMTHOST and RMTHOSTOPTIONS controls the size of the TCP buffer for uncompressed data. What actually enters the network will be less than this size if compression is enabled. GETTCPSTATS shows post-compression throughput.
`LOGEND`	Confirms whether or not Extract has processed all of the records in the data source.
`LOGSTATS`	(Oracle) Instructs Extract to issue a report about the statistics that are related to the processing of data from the Oracle redo log files. Extract uses an asynchronous log reader that reads ahead of the current record that Extract is processing, so that the data is available without additional I/O on the log files. The statistics are: ◆ AsyncReader.Buffers*n*: There is a field like this for each buffer queue that contains captured redo data. It shows the size, the number of records in it, and how long the wait time is before the data is processed. These statistics are given for write operations and read operations on the queue. ◆ REDO read ahead buffers: The number of buffers that are being used to read ahead asynchronously. ◆ REDO read ahead buffer size: The size of each buffer. ◆ REDO bytes read ahead for current redo: Whether read-ahead mode is on or off for the current redo log file (value of ON or OFF). ◆ REDO bytes read: The number of bytes read from all redo log files that are associated with this instance of Extract.

Argument	Description
	◆ REDO bytes read ahead: The number of bytes that were processed by the read-ahead mechanism. ◆ REDO bytes unused: The number of read-ahead bytes that were subsequently dropped as the result of Extract position changes or stale reads. ◆ REDO bytes parsed: The nunber of bytes that were processed as valid log data. ◆ REDO bytes output: The number of bytes that were written to the trail file (not including internal Oracle GoldenGate overhead).
`REPORT`	Generates an interim statistical report to the Extract report file. The statistics that are displayed depend upon the configuration of the STATOPTIONS parameter when used with the RESETREPORTSTATS | NORESETREPORTSTATS option. See page 335.
`RESUME`	Resumes (makes active) a process that was suspended by an EVENTACTIONS SUSPEND event. The process resumes normal processing from the point at which it was suspended.
`ROLLOVER`	Causes Extract to increment to the next file in the trail when restarting. For example, if the current file is ET000002, the current file will be ET000003 after the command executes. A trail can be incremented from 000001 through 999999, and then the sequence numbering starts over at 000000.

`SHOWTRANS [<Transaction ID>]`
`[THREAD <n>]`
`[COUNT <n>]`
`[DURATION <duration><unit>]`
`[TABULAR] |`
`[FILE <name> [DETAIL]]`
	(MySQL, Oracle, SQL Server, Sybase) Displays information about open transactions. SHOWTRANS shows any of the following, depending on the database type:
	◆ Process checkpoint (indicating the oldest log needed to continue processing the transaction in case of an Extract restart)
	◆ Transaction ID
	◆ Extract group name
	◆ Redo thread number
	◆ Timestamp of the first operation that Oracle GoldenGate extracts from a transaction (not the actual start time of the transaction)
	◆ System change number (SCN)
	◆ Redo log number and RBA
	◆ Status (Pending COMMIT or Running). Pending COMMIT is displayed while a transaction is being written after a FORCETRANS was issued.

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td></td><td>Without options, SHOWTRANS displays all open transactions that will fit into the available buffer. To further control output, see the following options.

SHOWTRANS options:
◆ <Transaction ID> limits the command output to a specific transaction.
◆ THREAD <n> constrains the output to open transactions against a specific Oracle RAC thread. For <n>, use a RAC thread number that is recognized by Extract.
◆ COUNT <n> constrains the output to the specified number of open transactions, starting with the oldest one. Valid values are 1 to 100,000.
◆ DURATION <duration><unit> restricts the output to transactions that have been open longer than the specified time, where:
<duration> is the length of time expressed as a whole number.
<unit> is seconds, minutes, hours, or days in fully spelled out or abbreviated form:
S|SEC|SECS|SECOND|SECONDS
M|MIN|MINS|MINUTE|MINUTES
H|HOUR|HOURS
D|DAY|DAYS
For Sybase, which does not put a timestamp on each record, the duration is not always precise and depends on the time information that is stored in the transaction log for the BEGIN and COMMIT records.
◆ (Oracle) TABULAR generates output in tabular format similar to the default table printout from SQL*Plus. The default is field-per-row.
◆ FILE <name> forces Extract to write the transaction information to the specified file. There is no output to the console.
◆ (Oracle) FILE <name> DETAIL writes a hex and plain character dump of the data. This dumps the entire transaction from memory to the file. Viewing the data may help you decide whether to skip the transaction or force it to the trail.
Note: Basic detail information is automatically written to the report file at intervals specified by the WARNLONGTRANS CHECKINTERVAL parameter.</td></tr>
</table>

Argument	Description

`SKIPTRANS <Transaction ID>`
`[THREAD <n>]`
`[FORCE]`
	(MySQL, Oracle, SQL Server, Sybase) Forces Extract to skip the specified transaction, thereby removing any current data from memory and ignoring any subsequent data. A confirmation prompt must be answered unless FORCE is used.
	◆ <Transaction ID> is the transaction ID number. Get the ID number with SHOWTRANS or from an Extract runtime message. To use SKIPTRANS, the specified transaction must be the oldest one in the list of transactions shown with SHOWTRANS. You can repeat the command for other transactions in order of their age.
	◆ (Oracle) Use THREAD <n> to specify which thread generated the transaction in an Oracle RAC environment if there are duplicate transaction IDs. SKIPTRANS specifies the checkpoint index number, not the actual thread number. To specify the correct thread, issue the INFO EXTRACT <group> SHOWCH command, and then specify the READ checkpoint index number that corresponds to the thread number that you want to skip. See the examples for details.
	◆ Use FORCE to bypass the confirmation prompt.
	After using SKIPTRANS, wait at least five minutes if you intend to issue SEND EXTRACT with FORCESTOP. Otherwise, the transaction will still be present.

`STATUS`	Returns a detailed status of the processing state, including current position and activity. Possible processing status messages on the Current status line are: ◆ Delaying – waiting for more data ◆ Suspended – waiting to be resumed ◆ Processing data – processing data ◆ Starting initial load – starting an initial load task ◆ Processing source tables – processing data for initial load task ◆ Reading from data source – reading from the data source, such as a source table or transaction log ◆ Adding record to transaction list – adding a record to the file memory transaction list ◆ At EOF (end of file) – no more records to process

Argument	Description
	In addition to the preceding statuses, the following status notations appear during an Extract recovery after an abend event. You can follow the progress as Extract continually 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. **DB2 LUW** ◆ Group name and process ID ◆ Processing status ◆ LSN ◆ Timestamp **DB2 on z/OS** ◆ Group name and process ID ◆ Processing status ◆ Log RBA ◆ Timestamp ◆ BSDS **Oracle and Oracle RAC** ◆ Group name and process ID ◆ Processing status ◆ Redo thread number (RAC only) ◆ Redo log sequence number ◆ RBA in redo log ◆ Timestamp ◆ SCN (RAC only) ◆ Redo log name **SQL Server** ◆ Group name and process ID ◆ Processing status ◆ Timestamp **Teradata, primary Extract** ◆ Group name and process ID ◆ Processing status ◆ Timestamp

Argument	Description
	Teradata VAM-sort Extract ◆ Processing status ◆ VAM trail sequence number ◆ RBA in the VAM trail ◆ Timestamp ◆ VAM trail name **All databases, SOURCEISTABLE Extract task** ◆ Extract name and process ID ◆ RMTTASK ◆ Record number ◆ Timestamp ◆ Table name

`STOP`	Stops Extract.
	If there are any long-running transactions (based on the WARNLONGTRANS parameter), the following message will be displayed:
	`Sending STOP request to EXTRACT JC108XT...`
	`There are open, long-running transactions. Before`
	`you stop Extract, make the archives containing data`
	`for those transactions available for when Extract`
	`restarts. To force Extract to stop, use the SEND`
	`EXTRACT <group>, FORCESTOP command.`
	`Oldest redo log file necessary to restart Extract is:`
	`Redo Thread 1, Redo Log Sequence Number 150, SCN`
	`31248005, RBA 2912272.`

`TRACE[2] {<tracefile> | OFF}`	Turns tracing on and off. Tracing captures information to the specified file to reveal processing bottlenecks. ◆ TRACE captures step-by-step processing information. ◆ TRACE2 identifies code segments rather than specific steps. ◆ OFF turns off tracing. If a trace is running already, the existing trace file is closed and the trace is resumed to the new file specified with <tracefile>. Contact Oracle Support for assistance if the trace reveals significant processing bottlenecks. For more information, go to http://support.oracle.com.
`TRACE OFF <file name>`	Turns tracing off only for the specified trace file.

Argument	Description
`TRACEINIT`	Resets tracing statistics back to 0 and then starts accumulating statistics again. Use this option to track the current behavior of processing, as opposed to historical.

`TRANLOGOPTIONS`
`{PURGEORPHANEDTRANSACTIONS |`
`NOPURGEORPHANEDTRANSACTIONS}`
	Valid for Oracle RAC. Enables or disables purging of orphaned transactions that occur when a node fails and Extract cannot capture the rollback. See also “TRANLOGOPTIONS” on page 385.

`TRANLOGOPTIONS`
`TRANSCLEANUPFREQUENCY <minutes>`
	Valid for Oracle RAC. Specifies the interval, in minutes, after which Oracle GoldenGate scans for orphaned transactions and then re-scans to confirm and delete them. Valid values are from 1 to 43200 minutes. Default is 10 minutes. See also “TRANLOGOPTIONS” on page 385.

`VAMMESSAGE`
`“<Teradata command>”`
 	
`VAMMESSAGE {`
`“ARSTATS” |`
`“INCLUDELIST [filter]” |`
`“EXCLUDELIST [filter]”`
`}`
 	
`VAMMESSAGE “OPENTRANS”`
	Sends a command to the capture API that is used by Extract.
	<Teradata command> can be:
	◆ “control:terminate”
	Stops a replication group. Required before dropping or altering a replication group in Teradata.
	◆ “control:suspend”
	Suspends a replication group. Can be used when upgrading Oracle GoldenGate.
	◆ “control:resume”
	Resumes a replication group after it has been suspended.
	◆ “control:copy <database>.<table>”
	Copies a table from the source database to the target database.
	SQL/MX commands can be:
	◆ “ARSTATS”
	Displays TMF audit reading statistics
	◆ “INCLUDELIST [filter]”
	Displays the list of tables for which Extract has encountered data records in the audit trail that match the selection criteria in the TABLE parameters. The [filter] option allows use of a wildcard pattern to filter the list of tables returned.

Argument	Description
	◆ “EXCLUDELIST [filter]” Displays the list of tables for which Extract has encountered data records in the audit trail that do not match the selection criteria in the TABLE parameters. The [filter] option allows use of a wildcard pattern to filter the list of tables returned. Certain system tables that are implicitly excluded will always be present in the list of excluded tables. The module returns a response to GGSCI. The response can be either ERROR or OK along with a response message. **SQL Server command can be:** “OPENTRANS” Prints a list of open transactions with their transaction ID, start time, first LSN, and the number of operations they contain.

Example 1 `SEND EXTRACT finance, ROLLOVER`

Example 2 `SEND EXTRACT finance, STOP`

Example 3 `SEND EXTRACT finance, VAMMESSAGE “control:suspend”`

Example 4 `SEND EXTRACT finance, TRANLOGOPTIONS TRANSCLEANUPFREQUENCY 20`

Example 5 This example explains SKIPTRANS. Start with the following SHOWCH output, which shows that thread 2 is at Read Checkpoint #3.

```
GGSCI> INFO <extract> SHOWCH
Read Checkpoint #3
Oracle RAC Redo Log
Startup Checkpoint (starting position in the data source):
Thread #: 2
Sequence #: 17560
RBA: 65070096
Timestamp: 2011-07-30 20:04:47.000000
SCN: 1461.3499051750 (6278446271206)
Redo File: RAC4REDO/sss11g/onlinelog/group_4.292.716481937
```

Therefore, SKIPTRANS should be: SKIPTRANS <xid> THREAD 3.

Example 6 `SEND EXTRACT finance, SHOWTRANS COUNT 2`

Example 7 The following shows the output of SHOWTRANS.

SHOWTRANS default output

```
Oldest redo log file necessary to restart Extract is:
Redo Thread 1, Redo Log Sequence Number 148, SCN 30816254, RBA 17319664
------------------------------------------------------------
XID                 : 5.15.52582
Items               : 30000
Extract             : JC108XT
Redo Thread         : 1
Start Time          : 2011-01-18:12:51:27
SCN                 : 20634955
Redo Seq            : 103
Redo RBA            : 18616848
Status              : Running
------------------------------------------------------------
XID                 : 7.14.48657
Items               : 30000
Extract             : JC108XT
Redo Thread         : 1
Start Time          : 2011-01-18:12:52:14
SCN                 : 20635145
Redo Seq            : 103
Redo RBA            : 26499088
Status              : Running
```

SHOWTRANS output with TABULAR in effect (view is truncated on right)

```
XID         Items  Extract   Redo Thread  Start Time
5.15.52582  30000  JC108XT   1            2011-01-18:12:52:14

SHOWTRANS FILE <name> DETAIL
Dumping transaction memory at 2011-01-21 13:36:54.
Record #1:
Header (140 bytes):
       0: 0000 0A4A 0000 FFFF 0000 0000 0057 6C10        ...J........Wl.
      16: 02FF 3F50 FF38 7C40 0303 4141 414E 5A77        ..?P.8|@..AAANZw
      32: 4141 4641 4141 4B6F 4941 4144 0041 4141        AAFAAAKoIAAD.AAA
      48: 4E5A 7741 4146 4141 414B 6F49 4141 4400        NZwAAFAAAKoIAAD.
      64: 4141 414E 5A77 414A 2F41 4142 7A31 7741        AAANZwAJ/AABz1wA
      80: 4141 0041 4141 4141 4141 4141 4141 4141        AA.AAAAAAAAAAAAA
      96: 4141 4141 4100 0000 0140 FF08 0003 0000        AAAAA....@......
     112: 0000 0000 0000 70FF 0108 FFFF 0001 4A53        ......p.......JS
     128: 554E 2E54 4355 5354 4D45 5200                  UN.TCUSTMER.

Data (93 bytes):
       0: 2C00 0400 0400 0000 0100 0200 0300 0000        ,...............
      16: 0000 0000 0800 0000 1800 0000 2000 0400        ............ ...
      32: 1000 0600 0200 0000 284A 414E 456C 6C6F        ........(JANEllo
      48: 6352 4F43 4B59 2046 4C59 4552 2049 4E43        cROCKY FLYER INC
      64: 2E44 454E 5645 5220 6E43 4F20 7365 7400        .DENVER nCO set.
      80: 0000 0000 0000 0C00 0000 0000 00               .............
```

When analyzing the summary output of SHOWTRANS, understand that it shows all currently running transactions on the database (as many as will fit into a predefined buffer). Extract must track every open transaction, not just those that contain operations on tables configured for Oracle GoldenGate, because it is not known whether operations on configured tables will be added to a transaction at some point in the future.

The Items field of the SHOWTRANS output shows the number of operations in the transaction that have been captured by Oracle GoldenGate so far, not the total number of operations in the transaction. If none of the operations are for configured tables, or if only some of them are, then Items could be 0 or any value less than the total number of operations.

The Start Time field shows the timestamp of the first operation that Oracle GoldenGate extracts from a transaction, not the actual start time of the transaction itself.

NOTE Command output may vary somewhat from the examples shown due ongoing enhancements of the Oracle GoldenGate software.

START EXTRACT

Use START EXTRACT to start the Extract process. To confirm that Extract has started, use the INFO EXTRACT or STATUS EXTRACT command.

Extract also can be started from the operating system's command line for certain synchronization configurations. For more information on the proper configuration and startup method to use for your purposes, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

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

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* starts all Extract groups whose names begin with T.

Example

```
START EXTRACT finance
```

STATS EXTRACT

Use STATS EXTRACT to display statistics for one or more Extract groups. The output includes DML and DDL operations that are included in the Oracle GoldenGate configuration.

To get the most accurate number of operations per second that are being processed, do the following.

1. Issue the STATS EXTRACT command with the RESET option.
2. Issue the STATS EXTRACT REPORTRATE command. The LATEST STATISTICS field shows the operations per second.

Figure 5 Sample output using the LATEST and REPORTFETCH options

```
Sending STATS request to EXTRACT GGSEXT...

Start of Statistics at 2011-01-08 11:45:05.

DDL replication statistics (for all trails):
*** Total statistics since extract started     ***
   Operations                                    3.00
   Mapped operations                             3.00
   Unmapped operations                           0.00
   Default operations                            0.00
   Excluded operations                           0.00

Output to ./dirdat/aa:

Extracting from JDADD.EMPLOYEES to JDADD.EMPLOYEES:
*** Latest statistics since 2011-01-08 11:36:55 ***
   Total inserts                               176.00
   Total updates                                 0.00
   Total deletes                                40.00
   Total discards                                0.00
   Total operations                            216.00

Extracting from JDADD.DEPARTMENTS to JDADD.DEPARTMENTS:
*** Latest statistics since 2011-01-08 11:36:55 ***
No database operations have been performed.
End of Statistics.
```

NOTE The actual number of DML operations executed on a DB2 database might not match the number of extracted DML operations reported by Oracle GoldenGate. DB2 does not log update statements if they do not physically change a row, so Oracle GoldenGate cannot detect them or include them in statistics.

To get accurate statistics on a Teradata source system where Oracle GoldenGate is configured in maximum protection mode, issue STATS EXTRACT to the VAM-sort Extract, not the primary Extract. The primary Extract may contain statistics for uncommitted transactions that could be rolled back; whereas the VAM-sort Extract reports statistics only for committed transactions.

Syntax

```
STATS EXTRACT <group name>
[, <statistic>]
[, TABLE <table>]
[, TOTALSONLY <table spec>]
[, REPORTFETCH | NOREPORTFETCH]
[, REPORTRATE <time units>]
[, ... ]
```

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* returns statistics for all Extract groups whose names start with T.
`<statistic>`	The statistic to be displayed. More than one statistic can be specified by separating each with a comma, for example STATS EXTRACT finance, TOTAL, DAILY. Valid values: TOTAL Displays totals since process startup. DAILY Displays totals since the start of the current day. HOURLY Displays totals since the start of the current hour. LATEST Displays totals since the last RESET command. RESET Resets the counters in the LATEST statistical field.
`TABLE <table>`	Displays statistics only for the specified table or a group of tables specified with a wildcard (*).
`TOTALSONLY <table spec>`	Summarizes the statistics for the specified table or a group of tables specified with a wildcard (*).
`REPORTFETCH | NOREPORTFETCH`	Controls whether or not statistics about fetch operations are included in the output. The default is NOREPORTFETCH. See also “STATOPTIONS” on page 335.
`REPORTRATE <time units>`	Displays statistics in terms of processing rate rather than absolute values. Valid values: ◆ HR ◆ MIN ◆ SEC

Example The following example displays total and hourly statistics per minute for a specific table, and it also resets the latest statistics and outputs fetch statistics.

```
STATS EXTRACT finance, TOTAL, HOURLY, TABLE acct,
REPORTRATE MIN, RESET, REPORTFETCH
```

STATUS EXTRACT

Use STATUS EXTRACT to determine whether or not Extract is running. A status of RUNNING can mean one of the following:

- Active: Running and processing (or able to process) data. This is the normal state of a process after it is started.
- Suspended: The process is running, but suspended due to an EVENTACTIONS SUSPEND action. In a suspended state, the process is not active, and no data can be processed, but the state of the current run is preserved and can be continued by issuing the RESUME command in GGSCI. The RBA in the INFO command reflects the last checkpointed position before the suspend action. To determine whether the state is active or suspended, issue the SEND EXTRACT command with the STATUS option.

Syntax

```
STATUS EXTRACT <group name> [, TASKS | ALLPROCESSES]
```

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* returns status for all Extract groups whose names begin with T.
`TASKS`	Displays status only for Extract tasks. By default, tasks are not displayed unless you specify a single Extract group (without wildcards).
`ALLPROCESSES`	Displays status for all Extract groups, including tasks.

Example 1

```
STATUS EXTRACT finance
```

Example 2

```
STATUS EXTRACT fin*
```

STOP EXTRACT

Use STOP EXTRACT to stop Extract gracefully. The command preserves the state of synchronization for the next time Extract starts, and it ensures that Manager does not automatically start Extract.

If there are open, long-running transactions when you issue STOP EXTRACT, you might be advised of the oldest transaction log file that will be needed for that transaction when Extract is restarted. You can use the SEND EXTRACT option of SHOWTRANS to view details and data of those transactions and then, if desired, use the SKIPTRANS or FORCETRANS options to skip the transaction or force it to be written as a committed transaction to the trail. See page 36.

Syntax

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

Argument	Description
`<group name>`	The name of an Extract group or a wildcard (*) to specify multiple groups. For example, T* stops all Extract processes for groups whose names begin with T.

Example

```
STOP EXTRACT finance
```

UNREGISTER EXTRACT

Use UNREGISTER EXTRACT to remove the registration of an Extract group from an Oracle database. UNREGISTER EXTRACT is valid only for a primary Extract group. Do not use it for a data pump Extract.

To register an Extract group with the database, use the REGISTER EXTRACT command.

To upgrade an Extract from classic capture mode to integrated capture mode, use the ALTER EXTRACT command.

Syntax

```
UNREGISTER EXTRACT <group name>
{DATABASE | LOGRETENTION}
```

Argument	Description
`<group name>`	The name of the Extract group that is to be unregistered from the database. Do not use a wildcard. This group must currently be registered with the database.
`DATABASE`	Disables integrated capture mode for the Extract group. This command removes the database capture (mining) server that has the same name as the Extract group. For additional information about support for, and configuration of, the Extract capture modes, see the Oracle GoldenGate documentation for the Oracle database. Before using UNREGISTER EXTRACT with DATABASE, do the following: 1. Stop Extract with the STOP EXTRACT <group> command. 2. Log in to the mining database with the DBLOGIN or MININGDBLOGIN command with the privileges granted in the dbms_goldengate_auth.grant_admin_privilege procedure. Use MININGDBLOGIN if the mining database is a downstream database; otherwise, use DBLOGIN. 3. Delete the Extract group with DELETE EXTRACT.

Argument	Description
`LOGRETENTION`	Disables log retention for the specified Extract group and removes the underlying Oracle Streams capture process. Use UNREGISTER EXTRACT with LOGRETENTION only if you no longer want to capture changes with this Extract group. The log-retention feature is controlled with the LOGRETENTION option of the TRANLOGOPTIONS parameter. Before using UNREGISTER EXTRACT with LOGRETENTION, stop Extract with the STOP EXTRACT <group> command. Next, issue the DBLOGIN command with the privileges shown in Table 1 on page 78.

Example 1 `UNREGISTER EXTRACT sales LOGRETENTION`

Example 2 `UNREGISTER EXTRACT sales DATABASE`

Replicat commands

Use Replicat commands to create and manage Replicat groups. The Replicat process reads data extracted by the Extract process and applies it to target tables or prepares it for use by another application, such as a load application.

Command summary

ADD REPLICAT

Use ADD REPLICAT to create a Replicat group. Unless SPECIALRUN is specified, ADD REPLICAT creates checkpoints so that processing continuity is maintained from run to run. Before creating a Replicat group, review the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Command limitations

This command cannot exceed 500 bytes in size for all keywords and input, including any text that you enter for the DESC option.

Oracle GoldenGate supports up to 5,000 concurrent Extract and Replicat groups per instance of Oracle GoldenGate Manager. At the supported level, all groups can be controlled and viewed in full with GGSCI commands such as the INFO and STATUS commands. Oracle GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at the default level of 300 or below in order to manage your environment effectively.

Syntax

```
ADD REPLICAT <group name>
{, SPECIALRUN |
   , EXTFILE <file name> |
   , EXTTRAIL <trail name>}
[, BEGIN {NOW | yyyy-mm-dd:hh:mm[:ss[.cccccc]]} |
   , EXTSEQNO <seqno>, EXTRBA <rba>]
[, CHECKPOINTTABLE <owner.table> | NODBCHECKPOINT]
[, PARAMS <parameter file>]
[, REPORT <report file>]
[, DESC "<description>"]
```

Argument	Description
`<group name>`	The name of the Replicat group. For group naming conventions, see the Oracle GoldenGate *Windows and UNIX Reference Guide.*
`SPECIALRUN`	Creates a Replicat special run as a task. Either SPECIALRUN, EXTFILE, or EXTTRAIL is required. When Extract is in SPECIALRUN mode, do not start Replicat with the START REPLICAT command in GGSCI.

`EXTFILE`
`<full path name>`
	Specifies the relative or fully qualified name of an extract file that is specified with RMTFILE in the Extract parameter file.

`EXTTRAIL`
`<full path name>`
	Specifies the relative or fully qualified name of a trail that was created with the ADD RMTTRAIL or ADD EXTTRAIL command.

`BEGIN <start point>`	Defines an initial checkpoint in the trail. ◆ To begin replicating changes from when the group is created with ADD REPLICAT, use the NOW argument. ◆ To begin extracting changes from a specific time, use the date-time format of yyyy-mm-dd:hh:mm[:ss[.cccccc]].

Argument	Description
`EXTSEQNO <seqno>`	Specifies the sequence number of the file in a trail in which to begin processing data. Specify the sequence number, but not any zeroes used for padding. For example, if the trail file is c:\ggs\dirdat\aa000026, you would specify EXTSEQNO 26. By default, processing begins at the beginning of a trail unless this option is used. To use EXTSEQNO, you must also use EXTRBA. Contact Oracle Support before using this option. For more information, go to http://support.oracle.com.
`EXTRBA <rba>`	Specifies the relative byte address within the trail file that is specified by EXTSEQNO. Contact Oracle Support before using this option. For more information, go to http://support.oracle.com.
`CHECKPOINTTABLE <owner.table>`	Specifies that this Replicat group will write checkpoints to the specified table in the database. Include the owner and table name, as in hr.hr_checkpoint. This argument overrides any default CHECKPOINTTABLE specification in the GLOBALS file. The table must be added with the ADD CHECKPOINTTABLE command.
`NODBCHECKPOINT`	Specifies that this Replicat group will not write checkpoints to a checkpoint table. This argument overrides any default CHECKPOINTTABLE specification in the GLOBALS file. This argument is required if you do not want to use a checkpoint table with the Replicat group that is being created.
`PARAMS <parameter file>`	Specifies a parameter file in a location other than the default of dirprm within the Oracle GoldenGate directory. Specify the fully qualified path name.
`REPORT <report file>`	Specifies the full path name of a process report file in a location other than the default of dirrpt within the Oracle GoldenGate directory.
`DESC "<description>"`	Specifies a description of the group, such as "Loads account_tab on Serv2". The description must be within quotes. You can use either the abbreviated keyword DESC or the full word DESCRIPTION.

Example

```
ADD REPLICAT sales, EXTTRAIL dirdat\rt
```

ALTER REPLICAT

Use ALTER REPLICAT to change the attributes of a Replicat group that was created with the ADD REPLICAT command. Before using this command, stop Replicat by issuing the STOP REPLICAT <group name> command.

Syntax

```
ALTER REPLICAT <group name> , <option> [, ...]
```

Argument	Description
`<group name>`	The name of the Replicat group that is to be altered.
`<option>`	An ADD REPLICAT option. You can change the description or any service option that was configured with the ADD REPLICAT command, except for the CHECKPOINT and NODBCHECKPOINT options.

Example 1 `ALTER REPLICAT finance, EXTSEQNO 53`

Example 2 `ALTER REPLICAT finance, EXTRBA 0`

Example 3 `ALTER REPLICAT finance, BEGIN 2011-01-07:08:00:00`

CLEANUP REPLICAT

Use CLEANUP REPLICAT to delete run history for a specified Replicat group. The cleanup keeps the last run record intact so that Replicat can resume processing from where it left off.

Before using this command, stop Replicat by issuing the STOP REPLICAT <group name> command.

Syntax

```
CLEANUP REPLICAT <group name> [, SAVE <count>]
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* cleans up all Replicat groups whose names begin with T.
`SAVE <count>`	Excludes the specified number of the most recent records from the cleanup.

Example 1 The following deletes all but the last record.

```
CLEANUP REPLICAT finance
```

Example 2 The following deletes all but the most recent five records.

```
CLEANUP REPLICAT *, SAVE 5
```

DELETE REPLICAT

Use DELETE REPLICAT to delete a Replicat group. This command deletes the checkpoint file but leaves the parameter file intact. Then you can re-create the group or delete the parameter file as needed. This command frees up trail files for purging by Manager, because the checkpoints used by the deleted group are removed (assuming no other processes are reading the file).

Before using DELETE REPLICAT, do the following:

1. Stop Replicat.

   ```
   STOP REPLICAT <group name>
   ```

2. If this group uses a database checkpoint table, log into the database by using the DBLOGIN command, so that the checkpoints can be deleted from the table.

Syntax

```
DELETE REPLICAT <group name> [!]
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* deletes all Replicat groups whose names begin with T.
`!`	Use this option to delete the group's checkpoints from the checkpoint file on disk, but not from the checkpoint table in the database. This option can be used to ignore the prompt that occurs when a wildcard specifies multiple groups.

Example

```
DELETE REPLICAT finance
```

INFO REPLICAT

Use INFO REPLICAT to retrieve the processing history of a Replicat group. The output of this command includes:

- The status of Replicat (STARTING, RUNNING, STOPPED or ABENDED).
- Approximate Replicat lag.
- The trail from which Replicat is reading.
- Replicat run history, including checkpoints in the trail.
- Information about the Replicat environment.

The basic command, without the TASKS or ALLPROCESSES argument, displays information only for online (continuous) Replicat groups. Tasks are excluded.

Replicat can be stopped or running when INFO REPLICAT is issued. In the case of a running process, the status of RUNNING can mean one of the following:

- Active: Running and processing (or able to process) data. This is the normal state of a process after it is started.
- Suspended: The process is running, but suspended due to an EVENTACTIONS SUSPEND action. In a suspended state, the process is not active, and no data can be processed, but the state of the current run is preserved and can be continued by issuing the RESUME command in GGSCI. The RBA in the INFO command reflects the last checkpointed position before the suspend action. To determine whether the state is active or suspended, issue the SEND REPLICAT command with the STATUS option.

About lag

Checkpoint Lag is the lag, in seconds, at the time the last checkpoint was written to the trail. For example, if the following is true...

- Current time = 15:00:00
- Last checkpoint = 14:59:00
- Timestamp of the last record processed =14:58:00

...then the lag is reported as 00:01:00 (one minute, the difference between 14:58 and 14:59).

A lag value of UNKNOWN indicates that Replicat could be running but has not yet processed records, or that the source system’s clock is ahead of the target system’s clock (due to clock imperfections, not time zone differences). For more precise lag information, use LAG REPLICAT (see page 61).

Showing detail

To show detailed information, use the DETAIL option. The following is sample output.

Figure 6 Detailed INFO REPLICAT output

```
REPLICAT   DELTPCC          Last Started 2011-01-21 11:40 Status RUNNING
Checkpoint Lag              00:00:00 (updated 232:39:41 ago)
Log Read Checkpoint File    C:\GGS\DIRDAT\RT000000
                            2011-01-21 18:54:33.000000 RBA 4735245

Extract Source             Begin                      End
C:\GGS\DIRDAT\RT000000     2011-01-21 18:54           2011-01-21 18:54
C:\GGS\DIRDAT\RT000000     * Initialized *            2011-01-21 18:54

Current directory          C:\GGS
Report file                C:\GGS\dirrpt\DELTPCC.rpt
Parameter file             dirprm\DELTPCC.prm
Checkpoint file            C:\GGS\dirchk\DELTPCC.cpr
Checkpoint table           GG.CHECKPT
Process file               C:\GGS\dirpcs\DELTPCC.pcr
Error log                  C:\GGS\ggserr.log
```

Showing checkpoints

Replicat makes checkpoints in the trail file to mark its last read position. To view process checkpoints, use the SHOWCH option. The basic command shows current checkpoints. To view a specific number of previous checkpoints, type the value after the SHOWCH entry.

Figure 7 INFO REPLICAT, SHOWCH

```
REPLICAT   JC108RP   Last Started 2011-01-12 13:10   Status RUNNING
Checkpoint Lag       00:00:00 (updated 111:46:54 ago)
Log Read Checkpoint  File ./dirdat/eh000000
                     First Record  RBA 3702915
Current Checkpoint Detail:
   Read Checkpoint #1
   GGS Log Trail
   Startup Checkpoint(starting position in data source):
      Sequence #: 0
      RBA: 3702915
      Timestamp: Not Available
      Extract Trail: ./dirdat/eh
   Current Checkpoint (position of last record read in the data source):
      Sequence #: 0
      RBA: 3702915
      Timestamp: Not Available
      Extract Trail: ./dirdat/eh
   Header:
      Version = 2
      Record Source = A
      Type = 1
      # Input Checkpoints = 1
      # Output Checkpoints = 0
   File Information:
      Block Size = 2048
      Max Blocks = 100
      Record Length = 2048
      Current Offset = 0
   Configuration:
      Data Source = 0
      Transaction Integrity = -1
      Task Type = 0
   Status:
      Start Time = 2011-01-12 13:10:13
      Last Update Time = 2011-01-12 21:23:31
      Stop Status = A
      Last Result = 400
```

About Replicat checkpoints

Extract makes checkpoints in the trail.

Startup checkpoint

The startup checkpoint is the first checkpoint made in the trail when the process starts. Comprising this statistic are:

- Sequence #: The sequence number of the trail file where the checkpoint was written.
- RBA: The relative byte address of the record at which the checkpoint was made.
- Timestamp: The timestamp of the record at which the checkpoint was made.
- Extract Trail: The relative path name of the trail.

Current checkpoint

The current checkpoint is the position of the last record read by Replicat in the trail. This should match the Log Read Checkpoint statistic shown in the summary and in the basic INFO REPLICAT command without options. The fields for this statistic are the same as those of the Startup Checkpoint.

Other SHOWCH information

The Header, File Information, Configuration, and Status statistics at the end of the SHOWCH display are for use by Oracle Support analysts. They contain internal information that is useful when resolving a support case.

Syntax

```
INFO REPLICAT <group name>
[, DETAIL]
[, SHOWCH [<n>]]
[, TASKS | ALLPROCESSES]
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* shows all Replicat groups whose names begin with T.
`DETAIL`	Displays detail information.
`SHOWCH`	Displays current checkpoint details, including those recorded to the checkpoint file and those recorded to the checkpoint table, if one is being used. The database checkpoint display includes the table name, the hash key (unique identifier), and the create timestamp. Specify a value for <n> to include the specified number of previous checkpoints as well as the current one.
`TASKS`	Displays only Replicat tasks. Tasks that were specified by a wildcard argument are not displayed by INFO REPLICAT .
`ALLPROCESSES`	Displays all Replicat groups, including tasks.

Example 1 `INFO REPLICAT *, DETAIL, ALLPROCESSES`

Example 2 `INFO REPLICAT *, TASKS`

Example 3 `INFO REPLICAT finance, SHOWCH`

KILL REPLICAT

Use KILL REPLICAT to kill a Replicat process. Killing a process leaves the most recent checkpoint in place, and the current transaction is rolled back by the database, guaranteeing that no data is lost when the process is restarted. The Manager process will not attempt to restart a killed Replicat process. Use this command only if Replicat cannot be stopped gracefully with the STOP REPLICAT command.

Syntax
```
KILL REPLICAT <group name>
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* kills all Replicat processes whose group names begin with T.

Example
```
KILL REPLICAT finance
```

LAG REPLICAT

Use LAG REPLICAT to determine a true lag time between Replicat and the trail. LAG REPLICAT estimates the lag time more precisely than INFO REPLICAT because it communicates with Replicat directly rather than reading a checkpoint position.

About Replicat lag

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.

Syntax
```
LAG REPLICAT <group name>
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* shows lag for all Replicat groups whose names begin with T.

Example 1
```
LAG REPLICAT *
```

Example 2
```
LAG REPLICAT *fin*
```

SEND REPLICAT

Use SEND REPLICAT to communicate with a starting or running Replicat process. The request is processed as soon as Replicat is ready to accept commands from users.

Syntax

```
SEND REPLICAT <group name>,{
FORCESTOP |
GETLAG |
HANDLECOLLISIONS [<table spec>] |
NOHANDLECOLLISIONS [<table spec>] |
REPORT [HANDLECOLLISIONS [<table spec>]] |
RESUME |
STATUS |
STOP |
TRACE[2] [DDLINCLUDE | DDL[ONLY]] [FILE] <file name> |
TRACE[2] OFF |
TRACE OFF <file name> |
TRACEINIT
}
```

Argument	Description
`<group name>`	The name of the Replicat group. If Replicat is not running, an error is returned.
`FORCESTOP`	Forces Replicat to stop, bypassing any notifications. This command will roll back any active transaction and stop the process immediately.
`GETLAG`	Shows a true lag time between Replicat and the trail. Lag time is the difference, in seconds, between the time that the last record was processed by Replicat and the timestamp of the record in the trail. The results are the same as LAG REPLICAT.
`HANDLECOLLISIONS [<table spec>]`	Turns on the HANDLECOLLISIONS parameter. Instead of using this option, you can specify the HANDLECOLLISIONS parameter in the Replicat parameter file. HANDLECOLLISIONS is used for automatic error handling when performing initial data loads while the source database is active. Make certain to disable HANDLECOLLISIONS (either with SEND REPLICAT or by removing the parameter from the parameter file) after the initial load is complete and online data changes have been applied to the target tables. <table spec> restricts HANDLECOLLISIONS to a specific target table or a group of target tables specified with a standard wildcard (*).
`NOHANDLECOLLISIONS [<table spec>]`	Turns off the HANDLECOLLISIONS parameter but does not remove it from the parameter file. To avoid enabling HANDLECOLLISIONS the next time Replicat starts, remove it from the parameter file. <table spec> restricts NOHANDLECOLLISIONS to a specific target table or a group of target tables specified with a standard wildcard (*).

Argument	Description
`REPORT [HANDLECOLLISIONS [<table spec>]]`	Generates an interim statistical report to the Extract report file. The statistics that are displayed depend upon the configuration of the STATOPTIONS parameter when used with the RESETREPORTSTATS | NORESETREPORTSTATS option. See page 368. HANDLECOLLISIONS shows tables for which HANDLECOLLISIONS has been enabled. <table spec> restricts the output to a specific target table or a group of target tables specified with a standard wildcard (*).
`RESUME`	Resumes (makes active) a process that was suspended by an EVENTACTIONS SUSPEND event. The process resumes normal processing from the point at which it was suspended.
`STATUS`	Returns the current location within the trail and information regarding the current transaction. Fields output are: ◆ Processing status ◆ Position in the trail file ◆ Trail sequence number ◆ RBA in trail ◆ Trail name Possible processing status messages are: ◆ Delaying – waiting for more data ◆ Suspended – waiting to be resumed ◆ Waiting on deferred apply – delaying processing based on the DEFERAPPLYINTERVAL parameter. ◆ Processing data – processing data ◆ Skipping current transaction – START REPLICAT with SKIPTRANSACTION was used. ◆ Searching for START ATCSN <csn> – START REPLICAT with ATCSN was used. ◆ Searching for START AFTERCSN <csn> – START REPLICAT with AFTERCSN was used. ◆ Performing transaction timeout recovery – Aborting current incomplete transaction and repositioning to start new one (see TRANSACTIONTIMEOUT parameter). ◆ Waiting for data at logical EOF after transaction timeout recovery – Waiting to receive remainder of incomplete source transaction after a TRANSACTIONTIMEOUT termination. ◆ At EOF (end of file) – no more records to process
`STOP`	Stops Replicat gracefully.

Argument	Description

`TRACE[2] [DDLINCLUDE | DDL[ONLY]] [FILE] <tracefile>`	Turns tracing on and off. Tracing captures information to the specified file to reveal processing bottlenecks.
	◆ TRACE captures step-by-step processing information .
	◆ TRACE2 identifies code segments rather than specific steps.
	If a trace is already in progress, the existing trace file is closed and the trace resumes to the file specified with <tracefile> .
	Contact Oracle Support for assistance if the trace reveals significant processing bottlenecks. For more information, go to http://support.oracle.com.
	Tracing also can be enabled by means of the Replicat parameters TRACE and TRACE2.
	DDLINCLUDE | DDLONLY
	(Replicat only) Enables DDL tracing and specifies how DDL tracing is included in the trace report.
	◆ DDLINCLUDE includes DDL tracing in addition to regular tracing of transactional data processing.
	◆ DDL[ONLY] excludes the tracing of transactional data processing and only traces DDL. This option can be abbreviated to DDL.
	[FILE] <file name>
	The relative or fully qualified name of a file to which Oracle GoldenGate logs the trace information. The FILE keyword is optional, but must be used if other parameter options will follow the file name, for example:
	`SEND REPLICAT <group> TRACE FILE <file name> DDLINCLUDE`
	If no other options will follow the file name, the FILE keyword can be omitted, for example:
	`SEND REPLICAT <group> TRACE DDLINCLUDE <file name>`

`TRACE[2] OFF`	Turns off tracing.
`TRACE OFF <file name>`	Turns tracing off only for the specified trace file. This option supports the EVENTACTIONS feature, where there can be multiple trace files due to multiple EVENTACTIONS statements.
`TRACEINIT`	Resets tracing statistics back to 0 and then starts accumulating statistics again. Use this option to track the current behavior of processing, as opposed to historical.

Example 1 `SEND REPLICAT finance, HANDLECOLLISIONS`

Example 2 `SEND REPLICAT finance, REPORT HANDLECOLLISIONS fin_*`

Example 3 `SEND REPLICAT finance, GETLAG`

START REPLICAT

Use START REPLICAT to start Replicat. To confirm that Replicat has started, use the INFO REPLICAT or STATUS REPLICAT command.

About Replicat start options

Normal start point

START REPLICAT, without any options, causes Replicat to start processing at one of the following points to maintain data integrity:

- After graceful or abnormal termination: At the last unprocessed transaction in the trail from the previous run, as represented by the current read checkpoint.
- First-time startup after the group was created: From the beginning of the active trail file (seqno 0, rba 0).

Alternate start point

The SKIPTRANSACTION, ATCSN, and AFTERCSN options of START REPLICAT cause Replicat to begin processing at a transaction in the trail other than the normal start point. Use them to:

- Specify a logical recovery position after an error that prevents Replicat from moving forward in the trail. Replicat can be positioned to skip the offending transaction or transactions, with the understanding that the data will not be applied to the target.
- Specify a start position at which to begin applying transactional changes that were replicated during an initial load procedure. Whenever a transaction changes data in a database, the database engine assigns a change identifier that represents the state of the data at that point in time. This type of identifier, generically known as the *commit sequence number* (CSN) in Oracle GoldenGate terminology, helps the database to keep track of changing data states throughout different transactions. If you know the CSN that corresponds to the completion of a backup, you can start Replicat to apply replicated transactions from that point forward. This allows Replicat to bypass any replicated changes that represent states that are older than the ones included in the backup. The purpose of skipping the older data changes is to avoid duplicate-record and missing-record errors.

NOTE Skipping a transaction, or starting at or after a CSN, might cause Replicat to start more slowly than normal, depending on how much data in the trail must be read before arriving at the appropriate transaction record. To view the startup progress, use the SEND REPLICAT command with the STATUS option.

Starting Replicat from the command line

Replicat also can be started from the operating system's command line for certain synchronization configurations. For more information on the proper configuration and startup method to use for your purposes, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Syntax

```
START REPLICAT <group name>
[SKIPTRANSACTION | ATCSN <csn> | AFTERCSN <csn>]
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* starts all Replicat groups whose names begin with T.
`SKIPTRANSACTION`	Causes Replicat to skip the first transaction after its expected startup position in the trail. All operations from that first transaction are excluded. If the MAXTRANSOPS parameter is also being used for this Replicat, it is possible that the process will start to read the trail file from somewhere in the middle of a transaction. In that case, the remainder of the partial transaction is skipped, and Replicat resumes normal processing from the next begin-transaction record in the file. The skipped records are written to the discard file if the DISCARDFILE parameter is being used; otherwise, a message is written to the report file that is similar to: `User requested START SKIPTRANSACTION. The current transaction will be skipped. Transaction ID <txid>, position Seqno <seqno>, RBA <rba>` **Limitations:** ◆ Valid only when the trail that Replicat is reading is part of an online change synchronization configuration (with checkpoints). Not valid for task-type initial loads (where SPECIALRUN is used with ADD REPLICAT).

`ATCSN <csn> |`
`AFTERCSN <csn>`
	◆ ATCSN <csn> causes Replicat to skip transactions in the trail until it finds a begin-transaction indicator that contains the specified commit sequence number (CSN). This transaction and subsequent ones are applied to the target. All transactions with a CSN less than the specified one are skipped.
	◆ AFTERCSN <csn> causes Replicat to skip transactions in the trail until it finds the first transaction after the one that contains the specified CSN. All transactions whose begin-transaction record contains a CSN less than, or equal to, the specified one are skipped.
	For <csn>, see the appendix “About the commit sequence number”. The CSN must be in the format that is native to the database; otherwise, Replicat will abend and write a message to the report file.
	To determine the appropriate CSN to use, view the Replicat report file with the VIEW REPORT <group> command in GGSCI. If more thorough investigation is required to determine the correct CSN, an experienced Oracle GoldenGate user can use the Logdump utility. For more information about using Logdump, see the Oracle GoldenGate *Windows and UNIX Troubleshooting and Tuning Guide*.
	When ATCSN or AFTERCSN is used, a message similar to the following is written to the report file:
	`User requested start at commit sequence number (CSN) <csn-string>`
	or ...
	`User requested start after commit sequence number (CSN) <csn-string>`

Argument	Description
	Limitations: ◆ Valid only when the trail that Replicat is reading is part of an online change synchronization configuration (with checkpoints). Not valid for task-type initial loads (where SPECIALRUN is used with ADD REPLICAT).To support starting at, or after, a CSN, the trail must be of Oracle GoldenGate version 10.0.0 or later, because the CSN is stored in the file header. If Replicat is started with AFTERCSN against an earlier trail version, Replicat will abend and write an error to the report stating that the trail format is not supported.

NOTE When a record that is specified with a CSN is found, Replicat issues a checkpoint to ensure that subsequent restarts of the process that occur before the next checkpoint will start from the requested location, and not from a point prior to the requested CSN.

Example 1

```
START REPLICAT finance
```

Example 2 The following starts Replicat at an Oracle-specific CSN.

```
START REPLICAT finance, ATCSN 6488359
```

Example 3 The following starts Replicat at a SQL Server-specific CSN.

```
START REPLICAT finance, AFTERCSN 0X000004D2:0000162E:0009
```

STATS REPLICAT

Use STATS REPLICAT to display statistics for one or more Replicat groups.

Syntax

```
STATS REPLICAT <group name>
[, <statistic>]
[, TABLE <table>]
[, TOTALSONLY <table spec>]
[, REPORTCDR]
[, REPORTDETAIL | NOREPORTDETAIL]
[, REPORTRATE <time units>]
[, ... ]
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* shows statistics for all Replicat groups whose names begin with T.
`<statistic>`	The statistic to be displayed. More than one statistic can be specified by separating each with a comma, for example STATS REPLICAT finance, TOTAL, DAILY.

Argument	**Description**
	Valid values are: TOTAL Displays totals since process startup. DAILY Displays totals since the start of the current day. HOURLY Displays totals since the start of the current hour. LATEST Displays totals since the last RESET command. RESET Resets the counters in the LATEST statistical field.
`TABLE <table>`	Displays statistics only for the specified table or a group of tables specified with a wildcard (*).
`TOTALSONLY <table spec>`	Summarizes the statistics for the specified table or a group of tables specified with a wildcard (*).
`REPORTCDR`	Shows statistics for Conflict Detection and Resolution. Statistics include: ◆ Total CDR conflicts ◆ CDR resolutions succeeded ◆ CDR resolutions failed ◆ CDR INSERTROWEXISTS conflicts ◆ CDR UPDATEROWEXISTS conflicts ◆ CDR DELROWEXISTS conflicts ◆ CDR DELROWMISSING conflicts
`REPORTDETAIL | NOREPORTDETAIL`	Controls whether or not the output includes operations that were not replicated as the result of collision errors. These operations are reported in the regular statistics (inserts, updates, and deletes performed) plus as statistics in the detail display, if enabled. For example, if 10 records were insert operations and they were all ignored due to duplicate keys, the report would indicate that there were 10 inserts and also 10 discards due to collisions. The default is REPORTDETAIL. See also “STATOPTIONS” on page 335.
`REPORTRATE <time units>`	Displays statistics in terms of processing rate rather than absolute values. Valid values: ◆ HR ◆ MIN ◆ SEC

Example The following example displays total and hourly statistics per minute for a specific table, and it also resets the latest statistics. Statistics for discarded operations are not reported.

```
STATS REPLICAT finance, TOTAL, HOURLY, TABLE acct,
REPORTRATE MIN, RESET, NOREPORTDETAIL
```

STATUS REPLICAT

Use STATUS REPLICAT to determine whether or not Replicat is running. A status of RUNNING can mean one of the following:

- Active: Running and processing (or able to process) data. This is the normal state of a process after it is started.
- Suspended: The process is running, but suspended due to an EVENTACTIONS SUSPEND action. In a suspended state, the process is not active, and no data can be processed, but the state of the current run is preserved and can be continued by issuing the RESUME command in GGSCI. The RBA in the INFO command reflects the last checkpointed position before the suspend action. To determine whether the state is active or suspended, issue the SEND REPLICAT command with the STATUS option.

Syntax

```
STATUS REPLICAT <group name>
[, TASKS]
[, ALLPROCESSES]
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* shows status for all Replicat groups whose names begin with T.
`TASKS`	Displays status only for Replicat tasks. By default, tasks are not displayed unless you specify a single Replicat group (without wildcards).
`ALLPROCESSES`	Displays status for all Replicat groups, including tasks.

Example 1 `STATUS REPLICAT finance`

Example 2 `STATUS REPLICAT fin*`

STOP REPLICAT

Use STOP REPLICAT to stop Replicat gracefully. This command preserves the state of synchronization for the next time Replicat starts, and it ensures that Manager does not automatically start Replicat.

Syntax

```
STOP REPLICAT <group name> [!]
```

Argument	Description
`<group name>`	The name of a Replicat group or a wildcard (*) to specify multiple groups. For example, T* stops all Replicat groups whose names begin with T.

Argument	Description
`!`	(Exclamation point) Stops Replicat immediately. The transaction is aborted and the process terminates.

Example

```
STOP REPLICAT finance
```

ER commands

Use the ER commands to control multiple Extract and Replicat groups as a unit. Use them with wildcards to affect every Extract and Replicat group that satisfies the wildcard.

Syntax

```
<command> ER <group wildcard specification>
```

Argument	Description

`<command>`	Can be any of the following:
	`INFO`
	`KILL`
	`LAG`
	`SEND`
	`START`
	`STATS`
	`STATUS`
	`STOP`
	For descriptions and optional parameters for these commands, refer to the Extract or Replicat command equivalent in this chapter.

`<group wildcard specification>`	The wildcard specification for the groups that you want to affect with the command. Oracle GoldenGate will automatically increase internal storage to track up to 100,000 wildcard entries.

Example The following example starts and then stops the Extract and Replicat groups whose names contain the letter X.

```
GGSCI (ggs3) > START ER *X*
GGSCI (ggs3) > STOP ER *X*
```

Trail commands

Use trail commands to create and manage Oracle GoldenGate trails. A trail is a series of files in which Oracle GoldenGate temporarily stores extracted data on disk until it has been applied to the target location.

Command summary

ADD EXTTRAIL

ADD RMTTRAIL

ALTER EXTTRAIL

ALTER RMTTRAIL

DELETE EXTTRAIL

DELETE RMTTRAIL

INFO EXTTRAIL

INFO RMTTRAIL

ADD EXTTRAIL

Use ADD EXTTRAIL to create a trail for online processing on the local system and:

- Associate it with an Extract group.
- Assign a maximum file size.

Syntax

```
ADD EXTTRAIL <trail name>, EXTRACT <group name>
[, MEGABYTES <n>]
[SEQNO <n>]
```

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail. The trail name can contain only two characters. Oracle GoldenGate appends this name with a six-digit sequence number whenever a new file is created. For example, a trail named dirdat/tr would have files named dirdat/tr000001, dirdat/tr000002, and so forth.
`<group name>`	The name of the Extract group to which the trail is bound. Only one Extract process can write data to a trail.
`MEGABYTES <n>`	The maximum size, in megabytes, of a file in the trail. The default is 100.
`SEQNO <n>`	Specifies that the first file in the trail will start with the specified trail sequence number. Do not include any zero padding. For example, to start at sequence 3 of a trail named "tr," specify SEQNO 3. The actual file would be named /ggs/dirdat/tr000003. This option can be used during troubleshooting when Replicat needs to be repositioned to a certain trail sequence number. It eliminates the need to alter Replicat to read the required sequence number.

Example

```
ADD EXTTRAIL dirdat\aa, EXTRACT finance, MEGABYTES 200
```

Example

```
ADD EXTTRAIL /ggs/dirdat/tr000003
```

ADD RMTTRAIL

Use ADD RMTTRAIL to create a trail for online processing on a remote system and:

- Assign a maximum file size.
- Associate the trail with an Extract group.

In the parameter file, specify a RMTHOST entry before any RMTTRAIL entries to identify the remote system and TCP/IP port for the Manager process.

Syntax

```
ADD RMTTRAIL <trail name>, EXTRACT <group name>
[, MEGABYTES <n>]
[, SEQNO <n>]
```

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail. The actual trail name can contain only two characters. Oracle GoldenGate appends this name with a six-digit sequence number whenever a new file is created. For example, a trail named ./dirdat/tr would have files named ./dirdat/tr000001, ./dirdat/tr000002, and so forth.
`<group name>`	The name of the Extract group to which the trail is bound. Only one Extract process can write data to a trail.
`MEGABYTES <n>`	The maximum size, in megabytes, of a file in the trail. The default is 100.
`SEQNO <n>`	Specifies that the first file in the trail will start with the specified trail sequence number. Do not include any zero padding. For example, to start at sequence 3 of a trail named “tr,” specify SEQNO 3. The actual file would be named /ggs/dirdat/tr000003. This option can be used during troubleshooting when Replicat needs to be repositioned to a certain trail sequence number. It eliminates the need to alter Replicat to read the required sequence number.

Example

```
ADD RMTTRAIL dirdat\aa, EXTRACT finance, MEGABYTES 200
```

Example

```
ADD RMTTRAIL /ggs/dirdat/tr000003
```

ALTER EXTTRAIL

Use ALTER EXTTRAIL to change the attributes of a trail that was created with the ADD EXTTRAIL command (a trail on the local system). The change takes effect the next time that Extract starts.

Syntax

```
ALTER EXTTRAIL <trail name>, EXTRACT < group name>
[, MEGABYTES <n>]
```

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail, for example dirdat\aa.

Argument	Description
`<group name>`	The name of the Extract group to which the trail is bound.
`MEGABYTES <n>`	The maximum size of a file, in megabytes. The default is 100. After using this option, issue the SEND EXTRACT command with the ROLLOVER option to close the current trail file and open a new one.

Example

```
ALTER EXTTRAIL dirdat\aa, EXTRACT finance,
MEGABYTES 200
```

ALTER RMTTRAIL

Use ALTER RMTTRAIL to change the attributes of a trail that was created with the ADD RMTTRAIL command (a trail on a remote system). The change takes effect the next time that Extract starts.

Syntax

```
ALTER RMTTRAIL <trail name>, EXTRACT <group name>
[, MEGABYTES <n>]
```

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail, for example dirdat\aa.
`<group name>`	The name of the Extract group to which the trail is bound.
`MEGABYTES <n>`	The maximum size of a file, in megabytes. The default is 100. The default is 100. After using this option, issue the SEND EXTRACT command with the ROLLOVER option to close the current trail file and open a new one.

Example

```
ALTER RMTTRAIL dirdat\aa, EXTRACT finance,
MEGABYTES 200
```

DELETE EXTTRAIL

Use DELETE EXTTRAIL to delete the record of checkpoints associated with a trail on a local system. Checkpoints are maintained in a file bearing the same name as the group in the dirchk sub-directory of the Oracle GoldenGate directory.

This command only deletes references to the specified trail from the checkpoint file. It does not delete the trail files themselves. To delete the trail files, use standard operating system commands for removing files.

Syntax

```
DELETE EXTTRAIL <trail name>
```

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail, including the two-character trail prefix.

Example `DELETE EXTTRAIL dirdat/et`

DELETE RMTTRAIL

Use DELETE RMTTRAIL to delete the record of checkpoints associated with a trail on a remote system. Checkpoints are maintained in a file bearing the same name as the group in the dirchk sub-directory of the Oracle GoldenGate directory.

This command only deletes references to the specified trail from the checkpoint file. It does not delete the trail files themselves. To delete the trail files, use standard operating system commands for removing files.

Syntax `DELETE RMTTRAIL <trail name>`

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail, including the two-character trail prefix.

Example `DELETE RMTTRAIL dirdat/et`

INFO EXTTRAIL

Use INFO EXTTRAIL to retrieve configuration information for a local trail. It shows the name of the trail, the Extract that writes to it, the position of the last data processed, and the assigned maximum file size.

Figure 8 Sample INFO EXTTRAIL output

```
Extract Trail: c:\gg_81\dirdat\md
      Extract: GGSEXT8
        Seqno: 2
          RBA: 51080
    File Size: 100M
```

Syntax `INFO EXTTRAIL <trail name>`

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail or a wildcard designating multiple trails.

Example 1 `INFO EXTTRAIL dirdat\aa`

Example 2 `INFO EXTTRAIL *`

INFO RMTTRAIL

Use INFO RMTTRAIL to retrieve configuration information for a remote trail. It shows the name of the trail, the Extract that writes to it, the position of the last data processed, and the assigned maximum file size.

Figure 9 Sample INFO RMTTRAIL output

```
Extract Trail: /gg_81/dirdat/rt
      Extract: GGSEXT
        Seqno: 4
          RBA: 78066
    File Size: 100M
```

Syntax `INFO RMTTRAIL <trail name>`

Argument	Description
`<trail name>`	The relative or fully qualified path name of the trail or a wildcard designating multiple trails.

Example 1 `INFO RMTTRAIL dirdat\aa`

Example 2 `INFO RMTTRAIL *`

Parameter commands

Use parameter commands to view and manage Oracle GoldenGate parameter files.

Command summary

EDIT PARAMS

SET EDITOR

VIEW PARAMS

EDIT PARAMS

Use EDIT PARAMS to create or change a parameter file. By default, this command launches Notepad on Windows systems or the vi editor on UNIX systems. You can change the editor with the SET EDITOR command.

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

Syntax
```
EDIT PARAMS {MGR | <group> | <file name>}
```

Argument	Description
`MGR`	Opens a parameter file for the Manager process.
`<group>`	Opens a parameter file for the specified Extract or Replicat group.
`<file name>`	Opens the specified file. 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 must also specify the full path name with the PARAMS option of the ADD EXTRACT or ADD REPLICAT command when you create the process group.

Example 1
```
EDIT PARAMS finance
```

Example 2
```
EDIT PARAMS c:\lpparms\replp.prm
```

SET EDITOR

Use SET EDITOR to change the default text editor for the current session of GGSCI. The default editors are Notepad for Windows and vi for UNIX. GGSCI input, including to create parameter files, takes the character set of the local operating system.

Syntax
```
SET EDITOR <program name>
```

Argument	Description
`<program name>`	Any text editor.

Example The following example changes the default editor to Wordpad.

```
SET EDITOR wordpad
```

VIEW PARAMS

Use VIEW PARAMS to view the contents of a parameter file.

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

Syntax
```
VIEW PARAMS {MGR | <group> | <file name>}
```

Argument	Description
`MGR`	Shows the Manager parameter file.

Argument	Description
`<group>`	Shows the parameter file for the specified Extract or Replicat group.
`<file name>`	Shows the specified file. If the parameter file resides in a directory other than dirprm, specify the full path name.

Example 1 `VIEW PARAMS finance`

Example 2 `VIEW PARAMS c:\lpparms\replp.prm`

Database commands

Use the database commands to interact with the database.

Command Summary

DBLOGIN

ENCRYPT PASSWORD

LIST TABLES

MININGDBLOGIN

FLUSH SEQUENCE

DBLOGIN

Use DBLOGIN to establish a database connection through GGSCI in preparation to issue other Oracle GoldenGate commands that affect the database. The user who issues DBLOGIN should have the appropriate database privileges to perform the functions that are enacted by those commands. Any other special privileges that are required for a GGSCI command are listed with the reference documentation for that command.

Requirements when configuring Oracle integrated capture mode

If using DBLOGIN to issue REGISTER EXTRACT to initiate integrated capture against an Oracle database, the user who issues DBLOGIN must:

- Have privileges granted through the Oracle dbms_goldengate_auth.grant_admin_privilege procedure.
- Be the user that is specified with the USERID parameter for the Extract group that is associated with this DBLOGIN.
- Not be changed while Extract is in integrated capture mode.

Special database privileges to use log retention in classic capture mode

To enable the log-retention feature in classic capture mode for an Oracle database, DBLOGIN must be issued with special privileges before using REGISTER EXTRACT with the LOGRETENTION option. For simplicity, you can log in as the Extract database user if the correct privileges were granted to that user when Oracle GoldenGate was installed. Otherwise, log in as a user with the following privileges.

Table 1 Oracle EE 10.2 and later privileges for TRANLOG option

Oracle EE version	Privileges

10.2	1. Run package to grant Oracle Streams admin privilege.
	`exec dbms_streams_auth.grant_admin_privilege('<user>')`
	2. Grant INSERT into logmnr_restart_ckpt$.
	`grant insert on system.logmnr_restart_ckpt$ to <user>;`
	3. Grant UPDATE on streams$_capture_process.
	`grant update on sys.streams$_capture_process to <user>;`
	4. Grant the 'become user' privilege.
	`grant become user to <user>;`

11.1 and 11.2.0.1	1. Run package to grant Oracle Streams admin privilege.
	`exec dbms_streams_auth.grant_admin_privilege('<user>')`
	2. Grant the 'become user' privilege.
	`grant become user to <user>;`

11.2.0.2 and later	Run package to grant Oracle Streams admin privilege. `exec dbms_goldengate_auth.grant_admin_privilege('<user>')`

Syntax

```
DBLOGIN {
[SOURCEDB <dsn>] |
[, <database>@<host>:<port>] |
USERID {/ | <user id>}[, PASSWORD <password>]
[<algorithm> ENCRYPTKEY {<keyname> | DEFAULT}] [SYSDBA | SQLID <sqlid>]
[SESSIONCHARSET <character set>]
}
```

Argument	Description
`SOURCEDB <dsn>`	A datasource name. Required for Sybase, MySQL, and databases that use ODBC.
`<database>@<host>:<port>`	(MySQL) Specifies a connection string that contains the database name, host name, and database port number. Can be used to specify a port other than the default that is specified in the database configuration.
`USERID <user id>`	Use if database credentials are required. Specifies the name of a database user, a schema (SQL/MX) or a SQL*Net connect string (Oracle).

Argument	Description
`PASSWORD <password>`	Use when authentication is required to specify the password for the database user. If the password was encrypted by means of the ENCRYPT PASSWORD command, supply the encrypted password; otherwise, supply the clear-text password. If the password is case-sensitive, type it that way. If the PASSWORD clause is omitted, you are prompted for a password, and the password is not echoed. If either the user ID or password changes, the change must be made in the Oracle GoldenGate parameter files, including the re-encryption of the password if necessary.
`<algorithm>`	If the password was encrypted with ENCRYPT PASSWORD, specify the encryption algorithm that was used: AES128 AES192 AES256 BLOWFISH
`ENCRYPTKEY {<keyname> | DEFAULT}`	Specifies the encryption key that was specified with ENCRYPT PASSWORD. ◆ ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option. ◆ ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option.
`SYSDBA`	(Oracle) Specifies that the user logs in as sysdba.
`SQLID <sqlid>`	(DB2 on z/OS) Issues the SQL command SET CURRENT SQLID = 'sqlid' after the USERID login (with PASSWORD, if applicable) is completed. If the SET command fails, the entire DBLOGIN command fails as a unit.
`SESSIONCHARSET <database character set>`	(Sybase, Teradata and MySQL) Sets a database session character set for the GGSCI connection to the database. All subsequent commands will use the specified session character set. This command option overrides any SESSIONCHARSET that is specified in the GLOBALS file.

Example 1

```
DBLOGIN USERID ogg@ora1.ora, PASSWORD
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC AES128, ENCRYPTKEY
securekey1
```

Example 2

```
DBLOGIN SOURCEDB msqldb@host1:3305, USERID ogg@ora1.ora, PASSWORD
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC AES128, ENCRYPTKEY
securekey1
```

Example 3

```
DBLOGIN SOURCEDB msqldb@host1:3305, USERID ogg@ora1.ora, PASSWORD
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC AES128, ENCRYPTKEY
securekey1, SESSIONCHARSET ISO-8859-11
```

ENCRYPT PASSWORD

Use ENCRYPT PASSWORD to encrypt a password that is used in an Oracle GoldenGate parameter file or command.

Syntax

```
ENCRYPT PASSWORD <password>
[AES128 | AES192 | AES256 | BLOWFISH]
ENCRYPTKEY {<keyname> | DEFAULT}
```

Argument	Description
`<password>`	The login password. Do *not* enclose the password within quotes. If the password is case-sensitive, type it that way.
`AES128 | AES192 | AES256 | BLOWFISH`	Specifies the encryption algorithm to use. ◆ AES128 uses the AES-128 cipher, which has a key size of 128 bits. ◆ AES192 uses the AES-192 cipher, which has a key size of 192 bits. ◆ AES256 uses the AES-256 cipher, which has a key size of 256 bits. ◆ BLOWFISH uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32 bits to 128 bits. Use BLOWFISH only for backward compatibility with earlier Oracle GoldenGate versions. If no algorithm is specified, AES128 is the default for all database types except DB2 on z/OS and NonStop SQL/MX, where BLOWFISH is the default. All of the AES ciphers have a 128-bit block size. For more information about these ciphers, see the Advanced Encryption Security publication online. To use AES encryption for any database other than Oracle, the path of the lib sub-directory of the Oracle GoldenGate installation directory must be specified as an environment variable before starting any processes: ◆ UNIX: Specify the path as an entry to the LD_LIBRARY_PATH or SHLIB_PATH variable. For example: `setenv LD_LIBRARY_PATH ./lib:$LD_LIBRARY_PATH` ◆ Windows: Add the path to the PATH variable. You can use the SETENV parameter to set it as a session variable for the process.

Argument	Description
`ENCRYPTKEY {<keyname> | DEFAULT}`	Specifies the encryption key. ◆ <keyname> specifies the logical name of a user-created encryption key in a local ENCKEYS lookup file. The key name is used to look up the actual key in the ENCKEYS file. A user-created key and an associated ENCKEYS file is required when using AES encryption; optional, but recommended, for Blowfish encryption. To use <keyname>, generate the key with KEYGEN or another utility, then store it in an ENCKEYS file on the source and target systems. For more information, see the security guidelines in the Oracle GoldenGate *Windows and UNIX Administrator's Guide*. ◆ DEFAULT directs Oracle GoldenGate to generate a random key that is stored in the trail so that decryption can be performed by the downstream process. This type of key is insecure and should not be used in a production environment. Use this option only when BLOWFISH is specified. ENCRYPT PASSWORD returns an error if DEFAULT is used with any AES algorithm.

Example 1
```
ENCRYPT PASSWORD ny14072 AES192 ENCRYPTKEY superkey2
```

Example 2
```
ENCRYPT PASSWORD ny14072 BLOWFISH ENCRYPTKEY superkey3
```

Example 3
```
ENCRYPT PASSWORD ny14072 BLOWFISH ENCRYPTKEY DEFAULT
```

FLUSH SEQUENCE

Use FLUSH SEQUENCE immediately after you start Extract for the first time during an initial synchronization or a re-synchronization. This command updates an Oracle sequence so that initial redo records are available at the time that Extract starts to capture transaction data. Normally, redo is not generated until the current cache is exhausted. The flush gives Replicat an initial start point with which to synchronize to the correct sequence value on the target system. From then on, Extract can use the redo that is associated with the usual cache reservation of sequence values.

To use FLUSH SEQUENCE

1. The following Oracle procedures are used by FLUSH SEQUENCE:

Table 2 Procedures that support FLUSH SEQUENCE

Database	Procedure	User and Privileges
Source	updateSequence	Grant EXECUTE to the owner of the Oracle GoldenGate DDL objects, or other selected user if not using DDL support.
Target	replicateSequence	Grant EXECUTE to the Oracle GoldenGate Replicat user.

The sequence.sql script installs these procedures. Normally, this script is run as part of the Oracle GoldenGate installation process, but make certain that was done before using FLUSH SEQUENCE. If sequence.sql was not run, the flush fails and an error message similar to the following is generated:

```
Cannot flush sequence {0}. Refer to the Oracle GoldenGate for Oracle
documentation for instructions on how to set up and run the sequence.sql
script. Error {1}.
```

2. The GLOBALS file must contain a GGSCHEMA parameter that specifies the schema in which the procedures are installed. This user must have CONNECT, RESOURCE, and DBA privileges.
3. Before using FLUSH SEQUENCE, issue the DBLOGIN command as the database user that has EXECUTE privilege on the updateSequence procedure.

NOTE For full instructions on configuring Oracle GoldenGate to support FLUSH SEQUENCE, see the Oracle GoldenGate *Oracle Installation and Setup Guide*.

Syntax `FLUSH SEQUENCE <owner.sequence>`

Argument	Description
`<owner.sequence>`	The owner and name of an Oracle sequence. The schema name cannot be null. You can use an asterisk (*) wildcard for the sequence name but not for the owner name.

Example `FLUSH SEQUENCE scott.seq*`

LIST TABLES

Use LIST TABLES to list all tables in the database that match the specification provided with the command argument. Use the DBLOGIN command to establish a database connection before using this command.

Syntax `LIST TABLES <table>`

Argument	Description
`<table>`	The name of a table or a group of tables specified with a wildcard (*).

Example The following shows a LIST TABLES command and sample output.

```
GGSCI (sysa) 3> list tables tcust*
TCUSTMER
TCUSTORD
```

MININGDBLOGIN

Use MININGDBLOGIN to establish a connection to a downstream Oracle database logmining server in preparation to issue other Oracle GoldenGate commands that affect this database, such as REGISTER EXTRACT. Use this command only if establishing Extract in integrated capture mode.

To log into a source Oracle database that serves as the database logmining server, use the DBLOGIN command. MININGDBLOGIN is reserved for login to a downstream mining database.

The user who issues MININGDBLOGIN must:

- Have privileges granted through the Oracle dbms_goldengate_auth.grant_admin_privilege procedure.
- Be the user that is specified with the TRANLOGOPTIONS MININGUSER parameter for the Extract group that is associated with this MININGDBLOGIN.
- Not be changed while Extract is in integrated capture mode.

For support and configuration information for integrated capture, see the Oracle GoldenGate *Oracle Installation and Setup Guide*.

Syntax

```
MININGDBLOGIN {
USERID {/ | <user id>}[, PASSWORD <password>]
[<algorithm> ENCRYPTKEY {<keyname> | DEFAULT}] [SYSDBA]
}
```

Argument	Description
`<user id>`	Specifies the name of a database user or a SQL*Net connect string.
`<password>`	Use when authentication is required to specify the password for the database user. If the password was encrypted by means of the ENCRYPT PASSWORD command, supply the encrypted password; otherwise, supply the clear-text password. If the password is case-sensitive, type it that way. If the PASSWORD clause is omitted, you are prompted for a password, and the password is not echoed. If either the user ID or password changes, the change must be made in the Oracle GoldenGate parameter files, including the re-encryption of the password if necessary. However, it is recommended that this user remains constant.
`<algorithm>`	If the password was encrypted with ENCRYPT PASSWORD, specify the encryption algorithm that was used: AES128 AES192 AES256 BLOWFISH
`ENCRYPTKEY {<keyname> | DEFAULT}`	Specifies the encryption key that was specified with ENCRYPT PASSWORD. ◆ ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option. ◆ ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option.

Argument	Description
SYSDBA	Specifies that the user logs in as sysdba.

Example

```
MININGDBLOGIN USERID ogg@ora2.ora, PASSWORD
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC AES128, ENCRYPTKEY
securekey1
```

Trandata commands

Use trandata commands to configure the appropriate database components to provide the transaction information that Oracle GoldenGate needs to replicate source data operations.

Command summary

ADD SCHEMATRANDATA

ADD TRANDATA

DELETE SCHEMATRANDATA

DELETE TRANDATA

INFO SCHEMATRANDATA

INFO TRANDATA

ADD SCHEMATRANDATA

Use ADD SCHEMATRANDATA to enable schema-level supplemental logging for Oracle tables. ADD SCHEMATRANDATA acts on all of the current and future tables in a given schema to automatically log a superset of available keys that Oracle GoldenGate needs for row identification.

ADD SCHEMATRANDATA does the following:

- Enables Oracle supplemental logging for new tables created with a CREATE TABLE.
- Updates supplemental logging for tables affected by an ALTER TABLE to add or drop columns.
- Updates supplemental logging for tables that are renamed.
- Updates supplemental logging for tables for which unique or primary keys are added or dropped.

ADD SCHEMATRANDATA logs the key columns of a table in the following order of priority:

- Primary key
- In the absence of a primary key, all of the unique keys of the table, including those that are disabled, unusable or invisible. Unique keys that contain ADT member columns are also logged. Only unique keys on virtual columns (function-based indexes) are not logged.

- If none of the preceding exists, all scalar columns of the table are logged. (System-generated row-OIDs are always logged.)

When to use ADD SCHEMATRANDATA

ADD SCHEMATRANDATA must be used in the following cases:

- For all tables that are part of an Extract group that is to be configured for integrated capture. ADD SCHEMATRANDATA ensures that the correct key is logged by logging all of the keys.
- When DDL replication is active and DML is concurrent with DDL that creates new tables or alters key columns. It best handles scenarios where DML can be applied to objects very shortly after DDL is issued on them. ADD SCHEMATRANDATA causes the appropriate key values to be logged in the redo log atomically with each DDL operation, thus ensuring metadata continuity for the DML when it is captured from the log, despite any lag in Extract processing.

Additional requirements for using ADD SCHEMATRANDATA

- Minimal supplemental logging must be enabled at the *database level* in order for Oracle GoldenGate to process updates to primary keys and chained rows. This must be done through the database interface, not through Oracle GoldenGate. You can enable minimal supplemental logging by issuing the following DDL statement:

  ```
  SQL> alter database add supplemental log data;
  ```

  To verify that supplemental logging is enabled at the database level, issue the following statement:

  ```
  SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE;
  ```

  The output of the query must be YES or IMPLICIT. LOG_DATA_MIN must be explicitly set, because it is not enabled automatically when other LOG_DATA options are set.
- Before using ADD SCHEMATRANDATA, issue the DBLOGIN command. The user who issues the command must be granted the Oracle Streams administrator privilege.

  ```
  SQL> exec dbms_streams_auth.grant_admin_privilege('<user>')
  ```

Additional considerations for using ADD SCHEMATRANDATA

- ADD SCHEMATRANDATA can be used instead of the ADD TRANDATA command when DDL replication is not enabled. Note, however, that if a table has no primary key but has multiple unique keys, ADD SCHEMATRANDATA causes the database to log all of the unique keys. In such cases, ADD SCHEMATRANDATA causes the database to log more redo data than does ADD TRANDATA. To avoid the extra logging, designate one of the unique keys as a primary key, if possible.
- For tables with a primary key, with a single unique key, or without a key, ADD SCHEMATRANDATA adds no additional logging overhead, as compared to ADD TRANDATA. For more information, see “ADD TRANDATA” on page 86.
- If you must log additional, non-key columns of a specific table (or tables) for use by Oracle GoldenGate, such as those needed for FILTER statements and KEYCOLS clauses in the TABLE and MAP parameters, issue an ADD TRANDATA command for those columns. That command has a COLS option to issue table-level supplemental logging for the columns, and it can be used in conjunction with ADD SCHEMATRANDATA.

Syntax `ADD SCHEMATRANDATA <schema>`

Argument	Description
`<schema>`	The schema for which you want the supplementary key information to be logged. Do not use a wildcard.

Example `ADD SCHEMATRANDATA SCOTT`

ADD TRANDATA

Use ADD TRANDATA to enable Oracle GoldenGate to acquire the transaction information that it needs from the transaction records. Use the DBLOGIN command to establish a database connection before using this command.

ADD TRANDATA is valid only for the databases that are listed here. For other supported databases, this functionality may exist already or must be configured through the database interface. See the Oracle GoldenGate installation guide for your database for any special requirements that apply to making transaction information available.

DB2 databases

Use ADD TRANDATA to enable DATA CAPTURE CHANGES on specified tables. This command supports DB2 LUW and DB2 z/OS. By default, ADD TRANDATA issues one of the following commands to the database:

DB2 z/OS:

```
ALTER TABLE <name> DATA CAPTURE CHANGES;
```

DB2 LUW:

```
ALTER TABLE <name> DATA CAPTURE CHANGES INCLUDE LONGVAR COLUMNS;
```

For DB2 LUW, you can exclude the LONGVAR clause by using ADD TRANDATA with the EXCLUDELONG option.

SQL Server databases

Use ADD TRANDATA to enable the supplemental logging information that Oracle GoldenGate needs to reconstruct SQL operations. The SQL Server transaction log does not provide enough information by default.

Sybase databases

ADD TRANDATA marks a Sybase table for replication by executing the Sybase sp_setreptable and sp_setrepcol system procedures. ADD TRANDATA options employ database features to control how the database propagates LOB data for the specified table. See the ADD TRANDATA options list.

NOTE The ADD TRANDATA command will overwrite the LOB setting that is currently set for the table.

Oracle databases

ADD TRANDATA by default enables table-level supplemental logging. The command issues an ALTER TABLE command with an ADD SUPPLEMENTAL LOG DATA clause that is appropriate for the type of unique constraint (or lack of one) that is defined for the table.

Unless a KEYCOLS clause is used in the TABLE or MAP statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically with no virtual columns, no UDTs, no function-based columns, and no nullable columns. A key cannot contain a column that is part of an invisible index.
3. First unique key alphanumerically with no virtual columns, no UDTs, and no function-based columns, but can include nullable columns. A key cannot contain a column that is part of an invisible index.
4. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding virtual columns, UDTs, function-based columns, and any columns that are explicitly excluded from the Oracle GoldenGate configuration.

NOTE If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient WHERE clause.

Use ADD TRANDATA only if you are *not* using the Oracle GoldenGate DDL replication feature. If you are using the Oracle GoldenGate DDL replication feature, use the ADD SCHEMATRANDATA command to log the required supplemental data. It is possible to use ADD TRANDATA when DDL support is enabled, but only if you can guarantee one of the following:

- You can stop DML activity on any and all tables before users or applications perform DDL on them.
- You cannot stop DML activity before the DDL occurs, but you can guarantee that:
  - There is no possibility that users or applications will issue DDL that adds new tables whose names satisfy an explicit or wildcarded specification in a TABLE or MAP statement.
  - There is no possibility that users or applications will issue DDL that changes the key definitions of any tables that are already in the Oracle GoldenGate configuration.

ADD SCHEMATRANDATA ensures replication continuity should DML ever occur on an object for which DDL has just been performed. For more information, see “ADD SCHEMATRANDATA” on page 84.

You can use ADD TRANDATA even when using ADD SCHEMATRANDATA if you need to use the COLS option to log any non-key columns, such as those needed for FILTER statements and KEYCOLS clauses in the TABLE and MAP parameters.

Besides table-level logging, minimal supplemental logging must be enabled at the database level in order for Oracle GoldenGate to process updates to primary keys and chained rows. This must be done through the database interface, not through Oracle GoldenGate. You can enable minimal supplemental logging by issuing the following DDL statement:

```
SQL> alter database add supplemental log data;
```

To verify that supplemental logging is enabled at the database level, issue the following statement:

```
SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE;
```

The output of the query must be YES or IMPLICIT. LOG_DATA_MIN must be explicitly set, because it is not enabled automatically when other LOG_DATA options are set.

The following are additional options for Oracle ADD TRANDATA supplemental logging:

- Use the COLS option to log non-key columns as needed, such as those required for a KEYCOLS clause or for filtering and manipulation requirements. A KEYCOLS clause is checked when processing starts and prevents ADD TRANDATA from logging all of the columns of the table when it determines there is no primary or unique key.
- Use the NOKEY option to prevent the logging of key columns when needed.

Take the following into account when using ADD TRANDATA for an Oracle database:

- If any of the logging details change after Oracle GoldenGate starts extracting data, you must stop and start the Extract process that is reading from the affected table before any data is changed.
- When creating a supplemental log group with ADD TRANDATA, Oracle GoldenGate appends the table name, an underscore, and object ID to a prefix of GGS_. Because Oracle limits an object name to 30 characters, Oracle GoldenGate truncates long table names as needed so the prefix and object ID can be included.

Syntax

```
ADD TRANDATA <owner.table>
[, COLS (<column list>)]
[, INCLUDELONG | EXCLUDELONG]
[, LOBSNEVER | LOBSALWAYS | LOBSIFCHANGED | LOBSALWAYSNOINDEX]
[, NOKEY]
```

Argument	Description
`<owner.table>`	The owner and name of the table or file for which to log transaction data. A wildcard can be used for the object name but not the owner name. Used with a wildcard, ADD TRANDATA filters out names that match the names of system objects. To use ADD TRANDATA for objects that are not system objects but have names that match those of system objects in a wildcard pattern, issue ADD TRANDATA for those objects without using a wildcard.
`COLS (<column list>)`	Adds specific non-key column(s) to the supplemental logging. Can be used to log columns specified in a KEYCOLS clause and to log columns that will be needed for filtering or manipulation purposes, which might be more efficient than fetching those values with a FETCHCOLS clause in a TABLE statement. Separate multiple columns with commas, for example `NAME, ID, DOB`.

Argument	Description

`INCLUDELONG | EXCLUDELONG`	(DB2 LUW) Controls whether or not the ALTER TABLE issued by ADD TRANDATA includes the "INCLUDE LONGVAR COLUMNS" attribute. INCLUDELONG is the default. When ADD TRANDATA is issued with this option, Oracle GoldenGate issues the following statement:
	`ALTER TABLE <name> DATA CAPTURE CHANGES INCLUDE LONGVAR COLUMNS;`
	When EXCLUDELONG is used, the following is the command:
	`ALTER TABLE <name> DATA CAPTURE CHANGES;`
	When EXCLUDELONG is used, Oracle GoldenGate does not support functionality that requires before images of tables that include LONGVAR columns. Examples of this functionality are the GETUPDATEBEFORES, NOCOMPRESSUPDATES, and NOCOMPRESSDELETES parameters. To support this functionality, changes to LONGVAR columns in the transaction logs must include both the before and after images of the column value.

`LOBSNEVER | LOBSALWAYS | LOBSIFCHANGED | LOBSALWAYSNOINDEX`	(Sybase) Controls how the database propagates LOB data for the specified table. **Note:** The ADD TRANDATA command will overwrite the LOB setting that is currently set for the table. To change the setting afterwards, you must use the sp_setrepcol script. ◆ LOBSNEVER prevents LOB data from being propagated. Note this exception: If the LOB column is inserted with a NULL value, or if it is skipped in an INSERT operation, then Extract will write that column to the trail with NULL data. ◆ LOBSALWAYS does two things: it uses sp_setrepcol to set LOB replication to ALWAYS_REPLICATE (always replicate LOB data whether or not it has changed in a transaction), and it marks the table to use an index on replication (by means of the USE_INDEX option of sp_setreptable). Because a LOB is marked for replication in a single transaction, this can take a long time, and USE_INDEX reduces that time by creating a global nonclustered index for every LOB. A shared-table lock is held while the global nonclustered index is created. ◆ LOBSIFCHANGED replicates LOB data only if it was changed during a transaction. This reduces replication overhead but does not protect against inconsistencies that could occur on the target outside the replication environment. This is the default. ◆ LOBSALWAYSNOINDEX sets LOB replication to ALWAYS_REPLICATE (always replicate LOB data whether or not it has changed in a transaction). This adds overhead, but protects against inconsistencies that could occur on the target outside the replication environment. LOBSALWAYSNOINDEX does not mark the table to use an index on replication. The benefit is that no lock is held while ADD TRANDATA is being executed. LOBSALWAYSNOINDEX is the default for Sybase databases earlier than version 15.

Argument	Description
	Note: When using the ALWAYS_REPLICATE option, if a LOB column contains a NULL value, and then another column in the table gets updated (but not the LOB), that LOB will not be captured even though ALWAYS_REPLICATE is enabled. You can check the LOB settings of a table with the INFO TRANDATA command, after ADD TRANDATA has been used for that table. It shows the LOB settings for all of the LOB columns. You can use the Sybase system procedures to change the LOB settings for any given column as needed.
`NOKEY`	Suppresses the supplemental logging of primary key columns. If using NOKEY, use the COLS option to log alternate columns that can serve as keys, and designate those columns as substitute keys by using the KEYCOLS option of the TABLE or MAP parameter.

Example 1 The following example causes one of the following: the primary key to be logged for an Oracle table; supplemental data to be logged for a SQL Server table; or a Sybase table to be marked for replication.

```
ADD TRANDATA finance.acct
```

Example 2 The following Oracle example causes the primary key to be logged plus the non-key columns name and address.

```
ADD TRANDATA finance.acct, COLS (name, address)
```

Example 3 The following Oracle example prevents the primary key from being logged, but logs the non-key columns name and pid instead.

```
ADD TRANDATA finance.acct, NOKEY, COLS (name, pid)
```

Example 4 The following Sybase example marks the acct table for replication and specifies to log LOB data only if it was changed during a transaction

```
ADD TRANDATA finance.acct, LOBSIFCHANGED
```

DELETE SCHEMATRANDATA

Use DELETE SCHEMATRANDATA to remove the Oracle schema-level supplemental logging that was added with the ADD SCHEMATRANDATA command. Use the DBLOGIN command to establish a database connection before using this command. The user that is specified with this command must have the privilege to remove supplemental log groups.

Syntax

```
DELETE SCHEMATRANDATA <schema>
```

Argument	Description
`<schema>`	The schema for which you want supplemental logging to be removed. Do not use a wildcard.

Example

```
DELETE SCHEMATRANDATA SCOTT
```

DELETE TRANDATA

Use DELETE TRANDATA to do one of the following:

- DB2 LUW and DB2 on z/OS: Alters the table to DATA CAPTURE NONE.
- Oracle: Disable supplemental logging.
- Sybase: Disable replication.
- SQL Server: Stop extended logging.

Use the DBLOGIN command to establish a database connection before using this command. The user specified with this command must have the same privileges that are required for ADD TRANDATA.

Syntax

```
DELETE TRANDATA <owner.table>
```

Argument	Description
`<owner.table>`	The owner and name of the table or file. A wildcard can be used for the table name but not the owner name.

Example 1

```
DELETE TRANDATA finance.acct
```

Example 2

```
DELETE TRANDATA finance.ac*
```

INFO SCHEMATRANDATA

Use INFO SCHEMATRANDATA to determine whether Oracle schema-level supplemental logging is enabled for the specified schema. Use the DBLOGIN command to establish a database connection before using this command.

Syntax

```
INFO SCHEMATRANDATA <schema>
```

Argument	Description
`<schema>`	The schema for which you want to confirm supplemental logging. Do not use a wildcard.

Example

```
INFO SCHEMATRANDATA scott
```

INFO TRANDATA

Use INFO TRANDATA to get the following information:

- DB2 LUW and DB2 on z/OS: Determine whether DATA CAPTURE is enabled or not.
- Oracle: Determine whether supplemental logging is enabled, and to show the names of columns that are being logged supplementally. If all columns are being logged, the notation “ALL” is displayed instead of individual column names.
- Sybase: Determine whether replication is enabled or not, and whether all LOB columns have identical logging settings (as specified with the ADD TRANDATA LOB options.

- SQL Server: Determine whether or not extended logging is enabled.

Use the DBLOGIN command to establish a database connection before using this command.

```
INFO TRANDATA <owner.table>
```

Argument	Description
`<owner.table>`	The owner and name of the table or file for which you want to view trandata information. The owner is not required if it is the same as the user specified by the DBLOGIN command. A wildcard can be used for the table name but not the owner name.

Example 1 `INFO TRANDATA finance.acct`

Example 2 `INFO TRANDATA finance.ac*`

Checkpoint table commands

Use the checkpoint table commands to manage the checkpoint table that is used by Oracle GoldenGate to track the current position of Replicat in the trail. For more information about using a checkpoint table, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Command summary

ADD CHECKPOINTTABLE

CLEANUP CHECKPOINTTABLE

DELETE CHECKPOINTTABLE

INFO CHECKPOINTTABLE

ADD CHECKPOINTTABLE

Use ADD CHECKPOINTTABLE to create a checkpoint table in the target database. Replicat uses the table to maintain a record of its read position in the trail for recovery purposes.

A checkpoint table is optional; checkpoints are also maintained in a file on disk. The use of a checkpoint table causes checkpoints to be part of the Replicat transaction. This allows Replicat to recover better in certain circumstances than when checkpoints alone are used.

One table can serve as the default checkpoint table for all Replicat groups in an Oracle GoldenGate instance if you specify it with the CHECKPOINTTABLE parameter in a GLOBALS file. More than one instance of Oracle GoldenGate (multiple installations) can use the same checkpoint table. Oracle GoldenGate keeps track of the checkpoints even when the same Replicat group name exists in different instances. For more information, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Use the DBLOGIN command to establish a database connection before using this command. Do not change the names or attributes of the columns in this table. You may, however, change table storage attributes.

Syntax

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

Argument	Description
`<owner.table>`	The owner and name of the checkpoint table to be created. The name cannot contain any special characters, such as quotes, backslash, dollar sign, percent symbol, and so forth. The owner and name can be omitted if you are using this table as the default checkpoint table and it is listed with CHECKPOINTTABLE in the GLOBALS file. It is recommended, but not required, that the table be created in a schema dedicated to Oracle GoldenGate. If an owner and name are not specified, a default table is created based on the CHECKPOINTTABLE parameter in the GLOBALS parameter file. Record the name of the table, because you will need it to view statistics or delete the table if needed.

Example 1 The following adds a checkpoint table with the default name specified in the GLOBALS file.

```
ADD CHECKPOINTTABLE
```

Example 2 The following adds a checkpoint table with a user-defined name.

```
ADD CHECKPOINTTABLE ggs.fin_check
```

CLEANUP CHECKPOINTTABLE

Use CLEANUP CHECKPOINTTABLE to remove checkpoint records from the checkpoint table when there is no checkpoint file associated with it in the working Oracle GoldenGate directory (from which GGSCI was started). The purpose of this command is to remove checkpoint records that are not needed any more, either because groups were changed or files were moved.

Use the DBLOGIN command to establish a database connection before using this command.

Syntax

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

Argument	Description
`<owner.table>`	The owner and name of the checkpoint table to be cleaned up. If an owner and name are not specified, the table that is affected is the one specified with the CHECKPOINTTABLE parameter in the GLOBALS parameter file.

Example

```
CLEANUP CHECKPOINTTABLE ggs.fin_check
```

DELETE CHECKPOINTTABLE

Use DELETE CHECKPOINTTABLE to drop a checkpoint table from the database. Use the DBLOGIN command to establish a database connection before using this command.

To stop using a checkpoint table while the associated Replicat group remains active, follow these steps:

1. Run GGSCI.
2. Stop Replicat.

   ```
   STOP REPLICAT <group>
   ```

3. Delete the Replicat group and then add it back with the following commands.

   ```
   DELETE REPLICAT <group>
   ADD REPLICAT <group>, EXTTRAIL <trail>, NODBCHECKPOINT
   ```

4. Exit GGSCI, then start it again.
5. Start Replicat again.

   ```
   START REPLICAT <group>
   ```

6. Log into the database with the DBLOGIN command, using the appropriate authentication options for the database. See page 77.
7. Delete the checkpoint table with DELETE CHECKPOINTTABLE.

If the checkpoint table is deleted while Replicat is still running and transactions are occurring, Replicat will abend with an error that the checkpoint table could not be found. However, the checkpoints are still maintained on disk in the checkpoint file. To resume processing, add the checkpoint table back under the same name. Data in the trail resumes replicating. Then, if you still want to delete the checkpoint table, follow the recommended steps.

Syntax

```
DELETE CHECKPOINTTABLE [<owner.table>] [!]
```

Argument	Description
`<owner.table>`	The owner and name of the checkpoint table to be deleted. An owner and name are not required if they are the same as those specified with the CHECKPOINTTABLE parameter in the GLOBALS file.
`!`	Bypasses the prompt that confirms intent to delete the table.

Example

```
DELETE CHECKPOINTTABLE ggs.fin_check
```

INFO CHECKPOINTTABLE

Use INFO CHECKPOINTTABLE to confirm the existence of a checkpoint table and view the date and time that it was created. It returns a message similar to the following:

```
Checkpoint table HR.CHKPT_TBLE created 2011-01-06 11:51:53.
```

Use the DBLOGIN command to establish a database connection before using this command.

Syntax

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

Argument	Description
`<owner.table>`	The owner and name of the checkpoint table. An owner and name are not required if they are the same as those specified with the CHECKPOINTTABLE parameter in the GLOBALS file.

Example `INFO CHECKPOINTTABLE ggs.fin_check`

Oracle trace table commands

Use the trace table commands to manage the Oracle GoldenGate trace table that is used with bidirectional synchronization of Oracle databases. Replicat generates an operation in the trace table at the start of each transaction. Extract ignores all transactions that begin with an operation to the trace table. Ignoring Replicat's operations prevents data from looping back and forth between the source and target tables.

For more information about bidirectional synchronization, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Command summary

ADD TRACETABLE

DELETE TRACETABLE

INFO TRACETABLE

ADD TRACETABLE

Use ADD TRACETABLE to create a trace table in the Oracle database. The trace table must reside in the schema of the Oracle GoldenGate Extract user, as configured with the USERID parameter. The trace table prevents Replicat transactions from being extracted again in a bidirectional synchronization configuration.

Use the DBLOGIN command to establish a database connection before using this command.

The trace table has the following description.

Table 3 Description of trace table

Name	Null?	Type	Description
GROUP_ID	NOT NULL	VARCHAR2(8)	The name of the Replicat group or special run process.
DB_USER		VARCHAR2(30)	The user ID of the Replicat group or special run process.
LAST_UPDATE		DATE	The timestamp of the transaction.

Syntax

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

Argument	Description
`<owner>.<table>`	Optional, use only to specify a trace table with a name that is different from the default of GGS_TRACE. The owner must be the same owner that is specified with the USERID parameter in the Extract parameter file. To use the default name, omit this argument. Whenever possible, use the default table name. When using a trace table name other than the default of GGS_TRACE, specify it with the TRACETABLE parameter in the Extract and Replicat parameter files. Record the name, because you will need it for the parameter files and to view statistics or delete the table. For more information, see "TRACETABLE | NOTRACETABLE" on page 382.

Example 1 The following adds a trace table with the default name of GGS_TRACE.

```
ADD TRACETABLE
```

Example 2 The following adds a trace table with a user-defined name of ora_trace.

```
ADD TRACETABLE ora_trace
```

DELETE TRACETABLE

Use DELETE TRACETABLE to delete a trace table. Use the DBLOGIN command to establish a database connection before using this command.

Syntax

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

Argument	Description
`<owner.table>`	The owner and name of the trace table to be deleted. An owner and name are not required if the owner is the same as that specified with the USERID parameter and the trace table has the default name of GGS_TRACE.

Example

```
DELETE TRACETABLE ora_trace
```

INFO TRACETABLE

Use the INFO TRACETABLE command to verify the existence of the specified trace table in the local instance of the database. If the table exists, Oracle GoldenGate displays the name and the date and time that it was created; otherwise Oracle GoldenGate displays a message stating that the table does not exist. Use the DBLOGIN command to establish a database connection before using this command.

Syntax

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

Argument	Description
`<owner.table>`	The owner and name of the trace table to be verified. An owner and name are not required if the owner is the same as that specified with the USERID parameter and the trace table has the default name of GGS_TRACE.

Example

```
INFO TRACETABLE ora_trace
```

DDL commands

The following commands control aspects of DDL replication.

DUMPDDL

Use the DUMPDDL command to view the data in the Oracle GoldenGate DDL history table. This information is the same information that is used by the Extract process. It is stored in proprietary format, but can be exported in human-readable form to the screen or to a series of SQL tables that can be queried by using regular SQL.

DUMPDDL always dumps all of the records in the DDL history table. Use SQL queries or search redirected standard output to view information about particular objects and the operations you are interested in. Because the history table contains large amounts of data, only the first 4000 bytes (approximately) of a DDL statement are displayed in order to maintain efficient performance. The format of the metadata is string based. It is fully escaped and supports table and column names in their native character set.

Because the information is historical data that is provided by the DDL *before* trigger, it reflects the state of an object before a DDL change. Consequently, there will not be any data for CREATE operations.

NOTE The default name of the before trigger is GGS_DDL_TRIGGER_BEFORE.

Before using DUMPDDL, log into the database as the owner of the history table by using the DBLOGIN command.

Basic DUMPDDL

The basic DUMPDDL command outputs metadata to the following tables.

Table 4 DUMPDDL tables

Table	Description
GGS_DDL_OBJECTS	Information about the objects for which DDL operations are being synchronized. SEQNO is the primary key. All of the other tables listed here contain a SEQNO column that is the foreign key to GGS_DDL_OBJECTS.
GGS_DDL_COLUMNS	Information about the columns of the objects involved in DDL synchronization.
GGS_DDL_LOG_GROUPS	Information about the supplemental log groups involved in DDL synchronization.
GGS_DDL_PARTITIONS	Information about the partitions for objects involved in DDL synchronization.
GGS_DDL_PRIMARY_KEYS	Information about the primary keys of the objects involved in DDL synchronization.

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 (default name is GGS_DDL_HIST).

All of these tables are owned by the schema that was designated as the Oracle GoldenGate DDL schema during the installation of the DDL objects (see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*). To view the structure of these tables, use the DESC command in SQL*Plus.

DUMPDDL with SHOW

DUMPDDL with the SHOW option dumps the information contained in the history table to the screen in standard output format. No output tables are produced. All records in the DDL history table are shown.

Syntax

```
DUMPDDL [SHOW]
```

Argument	Description
SHOW	Dumps the DDL information to the screen in standard output format.

Miscellaneous commands

The following commands control various other aspects of Oracle GoldenGate.

Command summary

! command

Use the ! command to execute a previous GGSCI command without modifications. To modify a command before executing it again, use the FC command (see page 100). To display a list of previous commands, use the HISTORY command (see page 102).

The ! command without arguments executes the most recent command. Options enable you to execute any previous command by specifying its line number or a text substring. Previous commands can be executed again only if they were issued during the current session of GGSCI, because command history is not maintained from session to session.

Syntax

```
! [<n> | -<n> | <string>]
```

Argument	Description
`<n>`	Executes the command from the specified GGSCI line. Each GGSCI command line is sequenced, beginning with 1 at the start of the session.
`-<n>`	Executes the command issued <n> lines before the current line.
`<string>`	Executes the last command that starts with the specified text string.

Example 1

```
! 9
```

Example 2

```
! -3
```

Example 3

```
! sta
```

ALLOWNESTED

Use the ALLOWNESTED command to enable the use of nested OBEY files. A nested OBEY file is one that contains another OBEY file. The maximum number of nested levels is 16. An attempt to run a nested OBEY file in the default mode of NOALLOWNESTED will cause an error that is similar to the following:

```
ERROR: Nested OBEY scripts not allowed. Use ALLOWNESTED to allow nested
scripts.
```

When you exit your GGSCI session, the next GGSCI session will revert back to NOALLOWNESTED.

For more information, see “OBEY” on page 104.

Syntax

```
ALLOWNESTED | NOALLOWNESTED
```

CREATE SUBDIRS

Use CREATE SUBDIRS when installing Oracle GoldenGate. This command creates the default directories within the Oracle GoldenGate home directory. Use CREATE SUBDIRS before any other configuration tasks.

Syntax

```
CREATE SUBDIRS
```

FC

Use FC to display edit a previously issued GGSCI command and then execute it again. Previous commands are stored in the memory buffer and can be displayed by issuing the HISTORY command (see page 102).

Displaying previous commands

Issuing FC without arguments displays the most recent command. Options enable you to execute any previous command by specifying its line number or a text substring. Previous commands can be edited only if they were issued during the current GGSCI session, because history is not maintained from one session to another.

Editing commands

The FC command displays the specified command and then opens an editor with a prompt containing a blank line starting with two dots. To edit a command, use the space bar to position the cursor beneath the character in the displayed command where you want to begin editing, and then use one of the following arguments. Arguments are not case-sensitive and can be combined.

Table 5 FC editor commands

Argument	Description

`i <text>`	Inserts text. For example:
	`GGSCI (SysA) 24> fc 9`
	`GGSCI (SysA) 24> send mgr`
	`GGSCI (SysA) 24..        i childstatus`
	`GGSCI (SysA) 24> send mgr childstatus`

`r <text>`	Replaces text. For example:
	`GGSCI (SysA) 25> fc 9`
	`GGSCI (SysA) 25> info mgr`
	`GGSCI (SysA) 25..     rextract extjd`
	`GGSCI (SysA) 25> info extract extjd`

`d`	Deletes a character. To delete multiple characters, enter a d for each one. For example:
	`GGSCI (SysA) 26> fc 10`
	`GGSCI (SysA) 26> info extract extjd, detail`
	`GGSCI (SysA) 26..                  dddddddd`
	`GGSCI (SysA) 26> info extract extjd`

`<replacement text>`	Replaces the displayed command with the text that you enter on a one-for-one basis. For example:
	`GGSCI (SysA) 26> fc 10`
	`GGSCI (SysA) 26> info mgr`
	`GGSCI (SysA) 26..     extract extjd`
	`GGSCI (SysA) 26> info extract extjd`

To execute the command, press **Enter** twice, once to exit the editor and once to issue the command. To cancel an edit, type a forward slash (/) twice.

Syntax `FC [<n> | -<n> | <string>]`

Argument	Description
`<n>`	Displays the command from the specified line. Each GGSCI command line is sequenced, beginning with 1 at the start of the session.
`-<n>`	Displays the command that was issued <n> lines before the current line.
`<string>`	Displays the last command that starts with the specified text string.

Example 1 `FC 9`

Example 2 `FC -3`

Example 3 `FC sta`

HELP

Use HELP to obtain information about an Oracle GoldenGate command. The basic command returns a list of command categories and the associated commands. The <command> option restricts the output to that of a specific command.

Syntax
```
HELP [<command>]
```

Argument	Description
`<command>`	The command for which you want help.

Example
```
HELP add replicat
```

HISTORY

Use HISTORY to view a list of the most recently issued GGSCI commands since the GGSCI session started. You can use the ! command (page 99) or the FC command (page 100) to re-execute a command in the list.

Syntax
```
HISTORY [<n>]
```

Example
```
HISTORY 7
```

Argument	Description
`<n>`	Returns a specific number of recent commands, where <n> is any positive number.

The result of this command would be similar to:

```
1: start manager
2: status manager
3: info manager
4: send manager childstatus
5: start extract extjd
6: info extract extjd
7: history
```

INFO ALL

Use INFO ALL to display the status and lag (where relevant) for all Manager, Extract, and Replicat processes on a system. The basic command, without options, displays only online (continuous) processes. To display tasks, use either INFO ALL TASKS or INFO ALL ALLPROCESSES.

The **Status** and **Lag at Chkpt** (checkpoint) fields display the same process status and lag as the INFO EXTRACT and INFO REPLICAT commands.

Figure 10 Sample INFO ALL output

```
Program       Status      Group       Lag at Chkpt    Time Since Chkpt
MANAGER       RUNNING
EXTRACT       ABENDED     EXTCUST     00:00:00        96:56:14
EXTRACT       STOPPED     INITDL
EXTRACT       STOPPED     INITDBL
```

Syntax `INFO ALL [TASKS | ALLPROCESSES]`

Argument	Description
`TASKS`	Displays information only for tasks.
`ALLPROCESSES`	Displays information for online processes and tasks.

Example 1 `INFO ALL TASKS`

Example 2 `INFO ALL ALLPROCESSES`

INFO MARKER

Use INFO MARKER to review recently processed markers from a NonStop system. A record is displayed for each occasion on which GGSCI, Logger, Extract, or Replicat processed the marker.

Markers can only be added on a NonStop system, using Oracle GoldenGate for NonStop for HP NonStop software.

The following is an example of the output.

```
Processed             Added               Diff     Prog    Group    Node
2012-02-16:14:41:15   2012-02-16:14:41:08 00:00:07 Extract PQACMD   \QAMD
                      GROUPCMD REPLICAT RQACMD CLOSEFILES
2012-02-16:14:41:13   2012-02-16:14:41:08 00:00:05 Extract PQACMD   \QAMD
                      TACLCMD REPLICAT RQACMD FUP PURGEDATA $QA16.QAETAR
```

Where:

- **Processed** is the local time that a program processed the marker.
- **Added** is the local time at which the marker was inserted into the NonStop audit trails or log trails.
- **Diff** is the time difference between the Processed and Added values. Diff can serve as an indicator of the lag between the user application and Extract and Replicat activities.
- **Prog** shows which process processed the marker, such as GGSCI, Logger, Extract or Replicat.
- **Group** shows the Extract or Replicat group or Logger process that processed the marker. N/A is displayed if GGSCI processed the marker.
- **Node** shows the node where the marker was inserted into the audit trails.
- There might be an additional column if user-defined text was included in the ADD MARKER statement.

Syntax

```
INFO MARKER [COUNT <num items>]
```

Argument	Description
`COUNT <num items>`	Restricts the list to a specified number of the most recent markers.

OBEY

Use OBEY to process a file that contains a list of Oracle GoldenGate commands. OBEY is useful for executing commands that are frequently used in sequence.

You can call one OBEY file from another one. This is called a nested OBEY file. You can nest up to 16 OBEY files. To use nested OBEY files, you must enable the functionality by first issuing the ALLOWNESTED command. See page 100.

Syntax

```
OBEY <file name>
```

Argument	Description
`<file name>`	The relative or fully qualified path name of the file that contains the list of commands.

Example 1

```
obey ./mycommands.txt
```

The preceding command executes a file that might look something like this:

```
add extract fin, tranlog, begin now
add exttrail dirdat/aa, extract fin
add extract hr, tranlog, begin now
add exttrail dirdat/bb, extract hr
start extract *
info extract *, detail
```

Example 2 The following illustrates a nested OBEY file. Assume an OBEY file named addcmds.txt. Inside this file, there is another OBEY command that calls the OBEY file named startcmds.txt, which executes another set of commands.

```
obey ./addcmds.txt
```

(This executes the following:)

```
add extract fin, tranlog, begin now
add exttrail ggs/dirdat/aa, extract fin
add extract hr, tranlog, begin now
add exttrail ggs/dirdat/bb, extract hr
add replicat fin2, exttrail ggs/dirdat/aa, begin now
add replicat hr2, exttrail ggs/dirdat/bb, begin now
obey ./startcmds.txt
```

(The nested startcmds.txt file executes the following:)

```
start extract *
info extract *, detail
start replicat *
info replicat *, detail
```

SHELL

Use SHELL to execute shell commands from within the GGSCI interface.

Syntax `SHELL <command>`

Argument	Description
`<command>`	The system command to execute.

Example 1 `SHELL dir dirprm\*`

Example 2 `SHELL rm ./dat*`

SHOW

Use SHOW to display the Oracle GoldenGate environment.

Figure 11 Sample SHOW display

```
Parameter settings:
SET SUBDIRS    ON
SET DEBUG      OFF
Current directory: C:\GG_81
Using subdirectories for all process files
Editor:  notepad
Reports (.rpt)                  C:\GG_81\dirrpt
Parameters (.prm)               C:\GG_81\dirprm
Replicat Checkpoints (.cpr)     C:\GG_81\dirchk
Extract Checkpoints (.cpe)      C:\GG_81\dirchk
Process Status (.pcs)           C:\GG_81\dirpcs
SQL Scripts (.sql)              C:\GG_81\dirsql
Database Definitions (.def)     C:\GG_81\dirdef
```

Syntax `SHOW`

VERSIONS

Use VERSIONS to display operating system and database version information. For ODBC connections, driver version information is also displayed. To display database information, issue a DBLOGIN command first to establish a database connection.

Syntax

```
VERSIONS
```

VIEW GGSEVT

Use VIEW GGSEVT to view the Oracle GoldenGate error log (ggserr.log file). This file contains information about Oracle GoldenGate events, such as process startup, shutdown, and exception conditions. This information is recorded in the system error log, too, but viewing the Oracle GoldenGate error log sometimes is more convenient and may retain events further back in time.

The display can be lengthy. To exit the display before reaching the end, use the operating system's standard methods for terminating screen output.

Figure 12 Sample VIEW GGSEVT output

```
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 ).
```

Syntax

```
VIEW GGSEVT
```

VIEW REPORT

Use VIEW REPORT to view the process report that is generated by Extract or Replicat. The report lists process parameters, run statistics, error messages, and other diagnostic information.

The command displays only the current report. Reports are aged whenever a process starts. Old reports are appended with a sequence number, for example finance0.rpt, finance1.rpt, and so forth. To view old reports, use the [<n>] option.

Figure 13 Sample report

```
********************************************************************
** Running with the following parameters **
********************************************************************
sourceisfile
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128, ENCRYPTKEY securekey1
rmthost sys1, mgrport 8040
rmtfile /home/jdad/ggsora/dirdat/tcustord.dat, purge
table tcustord;
rmtfile /home/jdad/ggsora/dirdat/tcustmer.dat, purge
table tcustmer;
```

```
Processing table TCUSTORD
Processing table TCUSTMER

*********************************************************************
* ** Run Time Statistics ** *
*********************************************************************

Report at 2011-01-13 11:07:36 (activity since 2011-01-13 11:07:31)

Output to /home/jdad/ggsora/dirdat/tcustord.dat:

From Table TCUSTORD:
       #       inserts:         2
       #       updates:         0
       #       deletes:         0
       #      discards:         0

Output to /home/jdad/ggsora/dirdat/tcustmer.dat:

From Table TCUSTMER:
       #       inserts:         2
       #       updates:         0
       #       deletes:         0
       #      discards:         0
```

Syntax

```
VIEW REPORT {<group name>[<n>] | <file name>}
```

Argument	Description
`<group name>`	The name of the group. The command assumes the report file named <group>.rpt in the Oracle GoldenGate dirrpt sub-directory.
`<n>`	The number of an old report. Report files are numbered from 0 (the most recent) to 9 (the oldest).
`<file name>`	The relative or full path name of the file, such as c:\ggs\dirrpt\orders.rpt.

Example 1 The following displays an old report file (number 3) for the orders group.

```
VIEW REPORT orders3
```

Example 2 The following displays a specific report identified by file name.

```
VIEW REPORT dirrpt\orders.rpt
```

Oracle GoldenGate Parameters Summary

This chapter contains a summary of all of the Oracle GoldenGate configuration parameters. The parameter names link to the detailed parameter documentation when viewed online.

Parameter Categories

The following are the categories of Oracle GoldenGate parameters.

GLOBALS parameters

Manager parameters

Parameters common to Extract and Replicat

Extract parameters

Replicat parameters

DEFGEN parameters

GLOBALS parameters

The GLOBALS file stores parameters that relate to the Oracle GoldenGate instance as a whole, as opposed to runtime parameters for a specific process.

Table 6 All GLOBALS parameters

Parameter	Description
CHARSET	Specifies a multibyte character set for the process to use instead of the operating system default when reading the parameter file.
CHECKPOINTTABLE	Specifies a default checkpoint table.
DDLTABLE	Specifies a non-default name for the DDL history table that supports DDL synchronization for Oracle.
ENABLEMONITORING	Enables Oracle GoldenGate Monitor to view and monitor Oracle GoldenGate instances.
GGSCHEMA	Specifies the name of the schema that contains the database objects that support DDL synchronization for Oracle.

Table 6 All GLOBALS parameters

Parameter	Description
MARKERTABLE	Specifies a non-default name for the DDL marker table that supports DDL synchronization for Oracle.
MGRSERVNAME	Specifies the name of the Manager process when it is installed as a Windows service.
OUTPUTFILEUMASK	Specifies a umask that can be used by Oracle GoldenGate processes to create trail files and discard files.
USEANSISQLQUOTES	Enables SQL-92 rules for quoted object names and literals.
SYSLOG	Filters the types of Oracle GoldenGate messages that are written to the system logs.
TRAILCHARSET	Specifies the character set of the source data when the trail is of an older version that does not store the source character set, or to override the character set that is stored in the trail.
USEIPV6	Forces Oracle GoldenGate to use IPv6 for TCP/IP connections.

Manager parameters

Manager is the parent process of Oracle GoldenGate and is responsible for the management of its processes, resources, user interface, and the reporting of thresholds and errors. In most cases default settings for Manager suffice.

Table 7 Manager parameters: General

Parameter	Description
CHARSET	Specifies a multibyte character set for the process to use instead of the operating system default when reading the parameter file.
COMMENT | --	Allows insertion of comments in a parameter file.
SOURCEDB	Specifies a data source name as part of the login information.
USERID	Provides login information for Manager when it needs to access the database.

Table 7 Manager parameters: General (continued)

Parameter	Description
SYSLOG	Filters the types of Oracle GoldenGate messages that are written to the system logs.

Table 8 Manager parameters: Port management

Parameter	Description
DYNAMICPORTLIST	Specifies the ports that Collector can dynamically allocate.
PORT	Establishes the TCP/IP port number on which Manager listens for requests.

Table 9 Manager parameters: Process management

Parameter	Description
AUTORESTART	Specifies processes to be restarted by Manager after a failure.
AUTOSTART	Specifies processes to be started when Manager starts.
BOOTDELAYMINUTES	Determines how long after system boot time Manager delays until performing main processing activities. This parameter supports Windows.
UPREPORT	Determines how often process heartbeat messages are reported.

Table 10 Manager parameters: Event management

Parameter	Description
DOWNCRITICAL	Reports processes that stopped gracefully or abnormally.
DOWNREPORT	Controls the frequency for reporting stopped processes.
LAGCRITICAL	Specifies a lag threshold that is considered critical and generates a warning to the error log.
LAGINFO	Specifies a lag threshold at which an informational message is reported to the error log.
LAGREPORT	Sets an interval for reporting lag time to the error log.

Table 11 Manager parameters: Maintenance

Parameter	Description
CHECKMINUTES	Determines how often Manager cycles through maintenance activities.
PURGEDDLHISTORY	Purges rows from the Oracle DDL history table when they are no longer needed.
PURGEDDLHISTORYALT	Purges rows from the alternate Oracle DDL history table that keeps track of partition IDs that are associated with a table ID.
PURGEMARKERHISTORY	Purges Oracle marker table rows that are no longer needed.
PURGEOLDEXTRACTS	Purges trail data that is no longer needed.
PURGEOLDTASKS	Purges Extract and Replicat tasks after a specified period of time.
STARTUPVALIDATIONDELAY[CSECS]	Sets a delay time after which Manager checks that processes are still running after startup.

Parameters common to Extract and Replicat

These parameters are available for both the Extract and Replicat processes.

Table 12 Parameters common to Extract and Replicat: General

Parameter	Description
CHARSET	Specifies a multibyte character set for the process to use instead of the operating system default when reading the parameter file.
CHECKPARAMS	Verifies parameter file syntax.
COMMENT | --	Denotes comments in a parameter file.
GETENV	Retrieves variables that were set with the SETENV parameter.
OBEY	Processes parameter statements contained in a different parameter file.
SETENV	Specifies a value for a UNIX environment variable from within the GGSCI interface.

Table 12 Parameters common to Extract and Replicat: General (continued)

Parameter	Description
TRACETABLE | NOTRACETABLE	Specifies a trace table to which Replicat adds a record whenever it updates the target database. Causes Extract to ignore database changes generated by Replicat Supports Oracle.. Supports Oracle bi-directional replication.
USERID	Specifies database connection information.

Table 13 Parameters common to Extract and Replicat: Selection, Converting, and Mapping Data

Parameter	Description
ASCIITOEBCDIC	Converts ASCII text to EBCDIC for DB2 on z/OS systems running UNIX System Services.
COLMATCH	Establishes global column-mapping rules.
DDL	Enables and filters the capture of DDL operations.
BINARYCHARS | NOBINARYCHARS	Controls whether or not binary characters are treated as a null-terminated string.
DDLSUBST	Enables string substitution in DDL processing.
GETDELETES | IGNOREDELETES	Controls the extraction of delete operations.
GETINSERTS | IGNOREINSERTS	Controls the extraction of insert operations.
GETTRUNCATES | IGNORETRUNCATES	Controls the extraction of truncate statements.
GETUPDATEAFTERS | IGNOREUPDATEAFTERS	Controls the extraction of update after images.
GETUPDATEBEFORES | IGNOREUPDATEBEFORES	Controls the extraction of update before images.
GETUPDATES | IGNOREUPDATES	Controls the extraction of update operations.
REPLACEBADCHAR	Replaces invalid character values with another value.
SOURCEDEFS	Specifies a file that contains source data definitions created by the DEFGEN utility.
TRIMSPACES | NOTRIMSPACES	Controls whether trailing spaces are trimmed or not when mapping CHAR to VARCHAR columns.

Table 13 Parameters common to Extract and Replicat: Selection, Converting, and Mapping Data (continued)

Parameter	Description
VARWIDTHNCHAR | NOVARWIDTHNCHAR	Controls whether length information is written to the trail for NCHAR columns.
WILDCARDRESOLVE	Defines rules for processing wildcard table specifications in a TABLE statement.

Table 14 Parameters common to Extract and Replicat: Custom Processing

Parameter	Description
CUSEREXIT	Invokes a user exit routine during processing.
INCLUDE	Invokes a macro library.
MACRO	Defines an Oracle GoldenGate macro.
MACROCHAR	Defines a macro character other than the default of #.
SQLEXEC	Executes a stored procedure or query during Extract processing.

Table 15 Parameters common to Extract and Replicat: Reporting

Parameter	Description
CMDTRACE	Displays macro expansion steps in the report file.
LIST | NOLIST	Displays or suppresses the listing of macros in the report file.
REPORT	Schedules a statistical report.
STATOPTIONS	Specifies information to include in statistical displays.
REPORTCOUNT	Reports the number of records processed.
TRACE | TRACE2	Shows processing information to assist in revealing processing bottlenecks.

Table 16 Parameters common to Extract and Replicat: Tuning

Parameter	Description
ALLOCFILES	Controls the number of incremental memory structures allocated when the value of NUMFILES is reached.
CHECKPOINTSECS	Controls how often the process writes a checkpoint.
DBOPTIONS	Specifies database options.
DDLOPTIONS	Specifies DDL processing options.
DYNAMICRESOLUTION | NODYNAMICRESOLUTION	Suppresses the metadata lookup for a table until Extract encounters transactional data for it. Makes Extract start faster when there are numerous tables specified for synchronization.
EOFDELAY | EOFDELAYCSECS	Determines how long the process delays before searching for more data to process in its data source.
FUNCTIONSTACKSIZE	Controls the size of the memory stack that is used for processing Oracle GoldenGate functions.
NUMFILES	Controls the initial allocation of memory dedicated to storing information about tables to be processed by Oracle GoldenGate.

Table 17 Parameters common to Extract and Replicat: Error Handling

Parameter	Description
DDLERROR	Controls error handling for DDL extraction.
DISCARDFILE	Contains records that could not be processed.

Table 18 Parameters common to Extract and Replicat: Maintenance

Parameter	Description
DISCARDROLLOVER	Controls how often to create a new discard file.
PURGEOLDEXTRACTS	Purges obsolete trail files.
REPORTROLLOVER	Specifies when to create new report files.
DECRYPTTRAIL	Decrypts data in a trail or extract file.

Extract parameters

The Extract process captures either full data records or transactional data changes, depending on configuration parameters, and then sends the data to a target system to be applied to target tables or processed further by another process, such as a load utility.

Table 19 Extract parameters: General

Parameter	Description
ETOLDFORMAT	Generates trails in a format that is compatible with Replicat versions prior to Oracle GoldenGate version 6.0.
RECOVERYOPTIONS	Controls the recovery mode of the Extract process.
SOURCEDB	Specifies the data source as part of the login information.
TCPSOURCETIMER | NOTCPSOURCETIMER	Adjusts timestamps of records transferred to other systems when those systems reflect different times.

Table 20 Extract parameters: Processing method

Parameter	Description
DSOPTIONS	Specifies Extract processing options when a Teradata Access Module (TAM) is being used.
EXTRACT	Defines an Extract group as an online process.
GETAPPLOPS | IGNOREAPPLOPS	Controls whether or not operations from all processes except Replicat are written to a trail or file.
GETREPLICATES | IGNOREREPLICATES	Controls whether or not replicated operations are captured by an Extract on the same system.
PASSTHRU | NOPASSTHRU	Controls whether tables will be processed by a data-pump Extract in pass-through mode or whether data definitions will be required.
RMTTASK	Creates a processing task on a remote system.
SOURCEISTABLE	Extracts entire records from source tables.
VAM	Indicates that a Teradata Access Module (TAM) is being used to provide transactional data to the Extract process.

Table 21 Extract parameters: Selecting, converting, and mapping data

Parameter	Description
COMPRESSDELETES | NOCOMPRESSDELETES	Controls whether Oracle GoldenGate writes only the key or all columns to the trail for delete operations.
COMPRESSUPDATES | NOCOMPRESSUPDATES	Causes only primary key columns and changed columns to be logged for updates.
FETCHOPTIONS	Controls certain aspects of the way that Oracle GoldenGate fetches data.
SEQUENCE	Specifies sequences for synchronization.
TABLE for Extract	Specifies tables for extraction and controls column mapping and conversion.
TABLEEXCLUDE	Excludes tables from the extraction process.
TARGETDEFS	Specifies a file containing target table definitions for target databases that reside on the NonStop platform.
TRAILCHARSETASCII	Specifies the ASCII character set for data captured from DB2 on z/OS, when both ASCII and EBCDIC tables are present.
TRAILCHARSETEBCDIC	Specifies the EBCDIC character set for data captured from DB2 on z/OS, when both ASCII and EBCDIC tables are present.

Table 22 Extract parameters: Routing data

Parameter	Description
EXTFILE	Specifies an extract file to which extracted data is written on the local system.
EXTTRAIL	Specifies a trail to which extracted data is written on the local system.
RMTFILE	Specifies an extract file to which extracted data is written on a remote system.
RMTHOST	Specifies the target system and Manager port number.
RMTTRAIL	Specifies a trail to which extracted data is written on a remote system.

Table 23 Extract parameters: Formatting data

Parameter	Description
FORMATASCII	Formats extracted data in external ASCII format.
FORMATSQL	Formats extracted data into equivalent SQL statements.
FORMATXML	Formats extracted data into equivalent XML syntax.
NOHEADERS	Prevents record headers from being written to the trail.

Table 24 Extract parameters: Tuning

Parameter	Description
BR	Controls the Bounded Recovery feature of Extract.
CACHEMGR	Controls the virtual memory cache manager.
FLUSHSECS | FLUSHCSECS	Determines the amount of time that record data remains buffered before being written to the trail.
LOBMEMORY	Controls the amount of memory and temporary disk space available for caching transactions that contain LOBs.
MAXFETCHSTATEMENTS	Controls the maximum number of prepared queries that Extract can use to fetch data from the database.
RMTHOSTOPTIONS	Specifies connection attributes other than host information for a TCP/IP connection used by a passive Extract group.
THREADOPTIONS	Controls aspects of the way that Extract operates in an Oracle Real Application Cluster environment. Supports Oracle.
TRANLOGOPTIONS	Supplies capture processing options.
TRANSMEMORY	Controls the amount of memory and temporary disk space available for caching uncommitted transaction data.
WARNLONGTRANS	Defines a long-running transaction and controls the frequency of checking for and reporting them.

Table 25 Extract parameters: Maintenance

Parameter	Description
ROLLOVER	Specifies the way that trail files are aged.

Table 26 Extract parameters: Security

Parameter	Description
ENCRYPTTRAIL | NOENCRYPTTRAIL	Controls encryption of data in a trail or extract file.

Replicat parameters

The Replicat process reads data extracted by the Extract process and applies it to target tables or prepares it for use by another application, such as a load utility.

Table 27 Replicat parameters: General

Parameter	Description
TARGETDB	Specifies the data source as part of the login information.

Table 28 Replicat parameters: Processing method

Parameter	Description
BEGIN	Specifies a starting point for Replicat processing. Required when SPECIALRUN is specified.
BULKLOAD	Loads data directly into the interface of the Oracle SQL*Loader utility.
END	Specifies a stopping point for Replicat processing. Required when using SPECIALRUN.
GENLOADFILES	Generates run and control files that are compatible with a database load utility.
REPLICAT	Specifies a Replicat group for online change synchronization.
SPECIALRUN	Used for one-time processing that does not require checkpointing from run to run.

Table 29 Replicat parameters: Selecting, converting, and mapping data

Parameter	Description
ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP	Allows the same source-target MAP statement to appear more than once in the parameter file.
ALLOWNOOPUPDATES | NOALLOWNOOPUPDATES	Controls how Replicat responds to a “no-op” operation. A no-op operation is one in which there is no effect on the target table.
APPLYNOOPUPDATES | NOAPPLYNOOPUPDATES	Force a “no-op” update to be applied using all columns in both the SET and WHERE clauses.
ASSUMETARGETDEFS	Assumes that source and target tables have the same column structure.
CHARSETCONVERSION | NOCHARSETCONVERSION	Controls whether or not Replicat converts the character set of the source data to the character set of the target database when applying the data.
INSERTALLRECORDS	Inserts a new record into the target table for every change operation made to a record.
INSERTDELETES | NOINSERTDELETES	Converts deletes to inserts.
INSERTMISSINGUPDATES | NOINSERTMISSINGUPDATES	Converts an update to an insert when the target row does not exist.
INSERTUPDATES | NOINSERTUPDATES	Converts updates to inserts.
MAP for Replicat	Specifies a relationship between one or more source and target tables and controls column mapping and conversion.
MAPEXCLUDE	Excludes tables from being processed by a wildcard specification supplied in MAP statements.
SPACESTONULL | NOSPACESTONULL	Controls whether or not a target column containing only spaces is converted to NULL.
TABLE for Replicat	Specifies a table or tables for which event actions are to take place when a row satisfies the given filter criteria.
UPDATEDELETES | NOUPDATEDELETES	Converts deletes to updates.

Table 29 Replicat parameters: Selecting, converting, and mapping data (continued)

Parameter	Description
UPDATEINSERTS | NOUPDATEINSERTS	Converts inserts to updates.

Table 30 Replicat parameters: Routing data

Parameter	Description
EXTFILE	Defines the name of an extract file on the local system that contains data to be replicated. Used for one-time processing.
EXTTRAIL	Defines a trail containing data to be replicated. Used for one-time processing.

Table 31 Replicat parameters: Error handling and reportingg

Parameter	Description
HANDLECOLLISIONS | NOHANDLECOLLISIONS	Handles errors for duplicate and missing records.
HANDLETPKUPDATE	Prevents constraint errors associated with replicating transient primary key updates.
OVERRIDEDUPS | NOOVERRIDEDUPS	Overlays a replicated insert record onto an existing target record whenever a duplicate-record error occurs.
RESTARTCOLLISIONS | NORESTARTCOLLISIONS	Controls whether or not Replicat applies HANDLECOLLISIONS logic after Oracle GoldenGate has abended because of a conflict.
REPERROR	Determines how Replicat responds to database errors.
REPFETCHEDCOLOPTIONS	Determines how Replicat responds to operations for which a fetch from the source database was required.
SHOWSYNTAX	Causes Replicat to print its SQL statements to the report file.
SQLDUPERR	Specifies the database error number that indicates a duplicate record. Use with OVERRIDEDUPS.
WARNRATE	Determines how often database errors are reported.

Table 32 Replicat parameters: Tuning

Parameter	Description
BATCHSQL	Increases the throughput of Replicat processing by arranging similar SQL statements into arrays and applying them at an accelerated rate.
DEFERAPPLYINTERVAL	Specifies a length of time for Replicat to wait before applying replicated operations to the target database.
DYNSQL | NODYNSQL	Causes Replicat to use literal SQL statements rather than a compile-once, execute-many strategy.
GROUPTRANSOPS	Controls the number of records that are grouped into a Replicat transaction.
INSERTAPPEND | NOINSERTAPPEND	Controls whether or not Replicat uses an APPEND hint when applying INSERT operations to Oracle target tables.
MAXDISCARDRECS	Limits the number of discarded records reported to the discard file.
MAXSQLSTATEMENTS	Controls the number of prepared SQL statements that can be used by Replicat.
MAXTRANSOPS	Divides large source transactions into smaller ones on the target system.
NUMFILES	Controls the initial allocation of memory that is dedicated to storing information about tables to be processed by Oracle GoldenGate.
RETRYDELAY	Specifies the delay between attempts to retry a failed SQL operation.
TRANSACTIONTIMEOUT	Specifies a time interval after which Replicat will commit its open target transaction and roll back any incomplete source transactions that it contains, saving them for when the entire source transaction is ready to be applied.

DEFGEN parameters

DEFGEN creates a file with data definitions for source or target tables. Data definitions are needed when the source and target tables have different definitions or the databases are of different types.

Table 33 All DEFGEN parameters

Parameters	Description
CHARSET	Specifies a multibyte character set for the process to use instead of the operating system default when reading the parameter file.
DEFSFILE	Identifies the name of the file to which DEFGEN writes the definitions
SOURCEDB	Specifies the data source as part of the login information.
TABLE for DEFGEN	Identifies a table for which you want to capture a definition.
USERID	Specifies database connection information.

DDL parameters

These parameters control Oracle GoldenGate DDL support. Other parameters may be required with DDL support, but the ones here deal specifically with the DDL feature.

Table 34 All DDL parameters

Parameter	Description
DDL	Enables DDL support and filters DDL.
DDLERROR	Handles errors that occur during DDL replication.
DDLOPTIONS	Configures aspects of DDL replication other than filtering and string substitution.
DDLSUBST	Enables the substitution of strings in DDL operations.
DDLTABLE	Specifies an alternate name for the DDL history table.
GGSCHEMA	Specifies the name of the schema that contains the objects that support DDL replication.
PURGEDDLHISTORY	Controls the size of the DDL history table.
PURGEMARKERHISTORY	Controls the size of the DDL marker table.

CHAPTER 2

Oracle GoldenGate Parameters

This chapter describes all of the Oracle GoldenGate parameters that are available to users for configuring, running, and managing Oracle GoldenGate processes.

For instructions on creating, changing, and storing Oracle GoldenGate parameter files, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

ALLOCFILES

Valid for Extract and Replicat

Use the ALLOCFILES parameter to control the incremental number of memory structures allocated once the initial memory allocation specified by the NUMFILES parameter is reached (see page 285). Together, these parameters control how process memory is allocated for storing information about the source and target tables being processed.

The default values should be sufficient for both NUMFILES and ALLOCFILES, because memory is allocated by the process as needed, system resources permitting.

ALLOCFILES must occur before any TABLE or MAP entries to have any effect.

Default 500

Syntax

```
ALLOCFILES <number of structures>
```

Argument	Description
`<number of structures>`	The additional number of memory structures to be allocated. Do not set ALLOCFILES to an arbitrarily high number, or memory will be consumed unnecessarily. The memory structures of Oracle GoldenGate support up to two million tables.

Example

```
ALLOCFILES 1000
```

ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP

Valid for Replicat

Use the ALLOWDUPTARGETMAP and NOALLOWDUPTARGETMAP parameters to control whether or not duplicate MAP statements for the same source and target objects are accepted in a parameter file. For example, the following parameter file would be permissible with ALLOWDUPTARGETMAP enabled.

```
REPLICAT repcust
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1
SOURCEDEFS /ggs/dirdef/source.def
ALLOWDUPTARGETMAP
GETINSERTS
GETUPDATES
IGNOREDELETES
MAP ggs.tcustmer, TARGET ggs.tcustmer, COLMAP (USEDEFAULTS, deleted_row
= "N");
IGNOREINSERTS
IGNOREUPDATES
GETDELETES
UPDATEDELETES
MAP ggs.tcustmer, TARGET ggs.tcustmer, COLMAP (USEDEFAULTS,
deleted_row = "Y");
```

If ALLOWDUPTARGETMAP is not specified and the same source and target tables are mapped more than once, only the first MAP statement is used and the others are ignored.

Default NOALLOWDUPTARGETMAP

Syntax ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP

ALLOWNONVALIDATEDKEYS

Valid for GLOBALS

Use ALLOWNONVALIDATEDKEYS to allow Extract, Replicat, and GGSCI commands to allow primary keys that are not validated or keys that are invalid to be used by Oracle GoldenGate as a unique identifier. This parameter overrides the key selection criteria that is used by Oracle GoldenGate. When it is enabled, Oracle GoldenGate will use NON VALIDATED and NOT VALID primary keys as a unique identifier.

A key can become invalid as the result of an object reorgnization or a number of other actions, but if you know the keys are valid, ALLOWNONVALIDATEDKEYS saves the downtime of re-validating them, especially in a testing environment. However, when using ALLOWNONVALIDATEDKEYS, whether in testing or in production, you accept the risk that the target data may not be maintained accurately through replication: If a key proves to be non-valid and the table on which it is defined contains more than one record with the same key value, Oracle GoldenGate might choose the wrong target row to update.

To enable ALLOWNONVALIDATEDKEYS functionality when only DML support is active

To enable ALLOWNONVALIDATEDKEYS if only DML replication is in use, stop all processes, then add ALLOWNONVALIDATEDKEYS to the GLOBALS parameter file, and then restart the processes. To disable it, remove it from the GLOBALS file and then restart the processes.

To enable ALLOWNONVALIDATEDKEYS functionality when DDLsupport is active

1. Rebuild the DDL trigger by executing the ddl_setup.sql script from the Oracle GoldenGate installation directory. This script requires that you know the schema where the Oracle GoldenGate DDL objects are installed.

2. Add the ALLOWNONVALIDATEDKEYS parameter to the GLOBALS parameter file.
3. Update the GGS_SETUP table in the DDL schema by using the following SQL.

```
UPDATE <owner>.GGS_SETUP SET value='1' WHERE
property='ALLOWNONVALIDATEDKEYS';
COMMIT;
```

4. Restart all GoldenGate processes including Manager. From this point on, Oracle GoldenGate will select non-validated or non-valid primary keys as a unique identifier.

To disable ALLOWNONVALIDATEDKEYS functionality when DDL support is active

1. Remove ALLOWNONVALIDATEDKEYS from the GLOBALS parameter file.
2. Update the record that you added to the GGS_SETUP table to 0.
3. Restart all of the Oracle GoldenGate processes.

Default None (Disabled)

Syntax `ALLOWNONVALIDATEDKEYS`

ALLOWNOOPUPDATES | NOALLOWNOOPUPDATES

Valid for Replicat

Use ALLOWNOOPUPDATES and NOALLOWNOOPUPDATES to control how Replicat responds to a “no-op” operation. A no-op operation is one in which there is no effect on the target table. The following are some examples of how this can occur.

- The source table has a column that does not exist in the target table, or has a column that was excluded from replication (with a COLSEXCEPT clause). In either case, if that source column is updated, there will be no target column name to use in the SET clause within the Replicat SQL statement.
- An update is made that sets a column to the same value as the current one. The database does not log the new value, because it did not really change. However, Oracle GoldenGate extracts the operation as a change record because the primary key was logged — but there is no column value for the SET clause in the Replicat SQL statement.

By default (NOALLOWNOOPUPDATES), Replicat abends with an error because these types of operations do not update the database. With ALLOWNOOPUPDATES, Replicat ignores the operation instead of abending. The statistics reported by Replicat will show that an update was made, but the database will not be updated.

You can use the internal parameter APPLYNOOPUPDATES to force the update to be applied. APPLYNOOPUPDATES overrides ALLOWNOOPUPDATES. If both are specified then updates with only key columns will be applied. By default, Oracle GoldenGate will abend with the following message if it only has source key columns but there is no key defined for the target table.

```
2011-01-25 02:28:42 GGS ERROR    160 Encountered an update for target
table TELLER, which has no unique key defined. KEYCOLS can be used to
define a key. Use ALLOWNOOPUPDATES to process the update without applying
it to the target database. Use APPLYNOOPUPDATES to force the update to
be applied using all columns in both the SET and WHERE clause.
```

Exceptions when error-handling is in place

If ALLOWNOOPUPDATES is specified when the HANDLECOLLISIONS or INSERTMISSINGUPDATES parameters are being used, and if Oracle GoldenGate has all of the target key values, then Oracle GoldenGate will not ignore the update, but instead will apply it using all key columns in the SET clause and the WHERE clause (invoking APPLYNOOPUPDATES behavior). This is necessary so Oracle GoldenGate can detect if the row being updated is missing. If it is, then Oracle GoldenGate turns the update into an insert.

Default NOALLOWNOOPUPDATES (only applies if the table does not have a key)

Syntax `ALLOWNOOPUPDATES | NOALLOWNOOPUPDATES`

APPLYNOOPUPDATES | NOAPPLYNOOPUPDATES

Valid for Replicat

Use APPLYNOOPUPDATES to force a “no-op” update to be applied using all columns in both the SET and WHERE clauses. See ALLOWNOOPUPDATES | NOALLOWNOOPUPDATES for a description of “no-op.”

APPLYNOOPUPDATES uses whatever data is in the trail. If there is a primary key update record, it uses the before columns from the source. If there is a regular (non-key) update, it assumes that the after value is the same as the before value (otherwise it would be a primary key update). The preceding assumes source and target keys are identical. If they are not, you must use a KEYCOLS clause in the TABLE statement on the source.

Default `NOAPPLYNOOPUPDATES`

Syntax `APPLYNOPUPDATES | NOAPPLYNOPUPDATES`

ASCIITOEBCDIC

Valid for Extract data pump and Replicat

Use the ASCIITOEBCDIC parameter to control the conversion of data in the input trail file from ASCII to EBCDIC format. This parameter should only be used to support backward compatibility in cases where the input trail file was created by an Extract version prior to v10.0. It is ignored for all other cases, because ASCII to EBCDIC conversion is currently the default.

As of version 11.2.1, conversion is not allowed by a data pump.

Default None

Syntax `ASCIITOEBCDIC`

ASSUMETARGETDEFS

Valid for Replicat

Use the ASSUMETARGETDEFS parameter when the source and target tables specified with a MAP statement have identical column structure, such as when synchronizing a hot site. It directs Oracle GoldenGate not to look up source structures from a source-definitions file.

If source and target tables do not have the same structure, use the SOURCEDEFS parameter instead of ASSUMETARGETDEFS. See "SOURCEDEFS" on page 330.

To find out what makes source and target tables identical or different in Oracle GoldenGate terms, see "Associating replicated data with metadata" in the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Default None

Syntax

```
ASSUMETARGETDEFS
```

AUTORESTART

Valid for Manager

Use the AUTORESTART parameter to start one or more Extract and Replicat processes automatically after they fail. AUTORESTART provides fault tolerance when something temporary interferes with a process, such as intermittent network outages or programs that interrupt access to transaction logs.

You can use multiple AUTORESTART statements in the same parameter file.

To apply this parameter to an Extract group that is created in PASSIVE mode, use it for the Manager that is on the target system where the associated alias Extract group resides. Oracle GoldenGate will send the start command to the source system. If AUTORESTART is used locally for a passive Extract group, it is ignored.

If Manager encounters an out-of-order transaction upon restart, it will not restart Extract. Instead, it will log a warning that notifies you to use the ETROLLOVER option of SEND EXTRACT to advance the trail to skip the transaction that caused the error.

Default Do not auto-restart

Syntax

```
AUTORESTART <process type> <group name>
[, RETRIES <max retries>]
[, WAITMINUTES <wait minutes>]
[, RESETMINUTES <reset minutes>]
```

Argument	Description

`<process type>`	Specify one of the following:
	`EXTRACT`
	`REPLICAT`
	`ER` (Extract and Replicat)

Argument	Description
`<group name>`	A group name or wildcard specification for multiple groups. When wildcarding is used, Oracle GoldenGate starts all groups of the specified <process type> on the local system that satisfy the wildcard, except those in PASSIVE mode.

`RETRIES`
`<max retries>`
	The maximum number of times that Manager should try to restart a process before aborting retry efforts. The default number of tries is 2.

`WAITMINUTES`
`<wait minutes>`
	The amount of time, in minutes, to pause between discovering that a process has terminated abnormally and restarting the process. Use this option to delay restarting until a necessary resource becomes available or some other event occurs. The default delay is 2 minutes.

`RESETMINUTES`
`<reset minutes>`
	The window of time, in minutes, during which retries are counted. The default is 120 minutes (2 hours). After the time expires, the number of retries reverts to zero.

Example In the following example, Manager tries to start all Extract processes three times after failure within a one hour time period, and it waits five minutes before each attempt.

```
AUTORESTART EXTRACT *, RETRIES 3, WAITMINUTES 5, &
RESETMINUTES 60
```

AUTOSTART

Valid for Manager

Use the AUTOSTART parameter to start one or more Extract and Replicat processes automatically when Manager starts. AUTOSTART ensures that no process groups are overlooked and that synchronization activities start immediately.

You can use multiple AUTOSTART statements in the same parameter file.

To apply this parameter to an Extract group that is created in PASSIVE mode, use it for the Manager that is on the target system where the associated alias Extract group resides. Oracle GoldenGate will send the start command to the source system. If AUTOSTART is used locally for a passive Extract group, it is ignored.

If Manager encounters an out-of-order transaction upon restart, it will not restart Extract. Instead, it will log a warning that notifies you to use the ETROLLOVER option of SEND EXTRACT to advance the trail to skip the transaction that caused the error.

Default Do not autostart

Syntax

```
AUTOSTART <process type> <group name>
```

Argument	Description

`<process type>`	Specify one of the following:
	`EXTRACT`
	`REPLICAT`
	`ER` (Extract and Replicat)

`<group name>`	A group name or wildcard specification for multiple groups. When wildcarding is used, Oracle GoldenGate starts all groups of the specified <process type> that satisfy the wildcard on the local system, except those in PASSIVE mode.

Example `AUTOSTART ER *`

BATCHSQL

Valid for Replicat

Use the BATCHSQL parameter to increase the performance of Replicat. BATCHSQL causes Replicat to organize similar SQL statements into arrays and apply them at an accelerated rate. In its normal mode, Replicat applies one SQL statement at a time.

BATCHSQL is valid for:

- DB2 LUW
- DB2 on z/OS
- Oracle
- NonStop SQL/MX
- PostgreSQL
- SQL Server
- Teradata

How BATCHSQL works

In BATCHSQL mode, Replicat organizes similar SQL statements into batches within a memory queue, and then it applies each batch in one database operation. A batch contains SQL statements that affect the same table, operation type (insert, update, or delete), and column list. For example, each of the following is a batch:

- Inserts to table A
- Inserts to table B
- Updates to table A
- Updates to table B
- Deletes from table A
- Deletes from table B

NOTE Oracle GoldenGate analyzes foreign-key referential dependencies in the batches before executing them. If dependencies exist among statements that are in different batches, more than one SQL statement per batch might be required to maintain the referential integrity.

Controlling the number of cached statements

The MAXSQLSTATEMENTS parameter controls the number of statements that are cached (see page 282). Old statements are recycled using a least-recently-used algorithm. The batches are executed based on a specified threshold (see "Managing memory").

Usage restrictions

SQL statements that are treated as exceptions include:

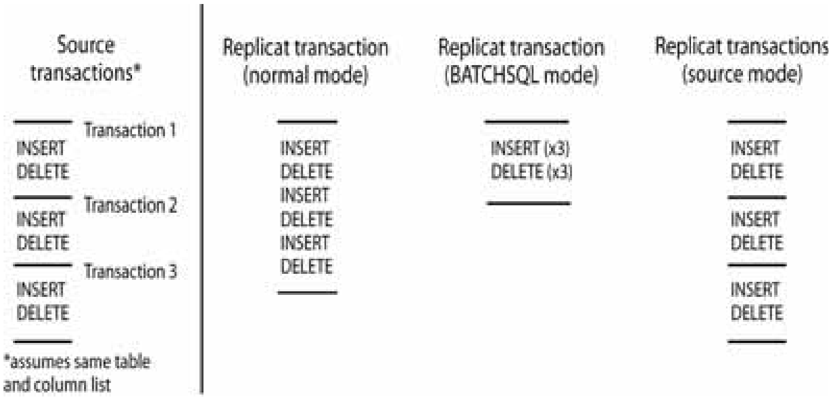
- Statements that contain LOB or LONG data.
- Statements that contain rows longer than 25k in length.
- Statements where the target table has one or more unique keys besides the primary key. Such statements cannot be processed in batches because BATCHSQL does not guarantee the correct ordering for non-primary keys if their values could change.
- (SQL Server) Statements where the target table has a trigger.
- Statements that cause errors.

When Replicat encounters exceptions in batch mode, it rolls back the batch operation and then tries to apply the exceptions in the following ways, always maintaining transaction integrity:

- First Replicat tries to use normal mode: one SQL statement at a time within the transaction boundaries that are set with the GROUPTRANSOPS parameter (see page 216).
- If normal mode fails, Replicat tries to use source mode: apply the SQL within the same transaction boundaries that were used on the source.

When finished processing exceptions, Replicat resumes BATCHSQL mode.

Figure 14 Replicat modes



Determining when to use BATCHSQL

When Replicat is in BATCHSQL mode, smaller row changes will show a higher gain in performance than larger row changes. At 100 bytes of data per row change, BATCHSQL has been known to improve the performance of Replicat by up to 300 percent, but actual performance benefits will vary, depending on the mix of operations. At around 5,000 bytes of data per row change, the benefits of using BATCHSQL diminish.

Managing memory

The gathering of SQL statements into batches improves efficiency but also consumes memory. To maintain optimum performance, use the following BATCHSQL options:

BATCHESPERQUEUE

BYTESPERQUEUE

OPSPERBATCH

OPSPERQUEUE

As a benchmark for setting values, assume that a batch of 1,000 SQL statements at 500 bytes each would require less than 10 megabytes of memory.

Default Disabled (Process in normal Replicat mode)

Syntax

```
BATCHSQL
[BATCHERRORMODE | NOBATCHERRORMODE]
[BATCHESPERQUEUE <n>]
[BATCHTRANSOPS <n>]
[BYTESPERQUEUE <n>]
[OPSPERBATCH <n>]
[OPSPERQUEUE <n>]
[TRACE]
```

Option	Description
BATCHERRORMODE | NOBATCHERRORMODE	Sets the response of Replicat to errors that occur during BATCHSQL processing mode. ◆ BATCHERRORMODE causes Replicat to try to resolve errors without leaving BATCHSQL mode. It converts inserts that fail on duplicate-record errors to updates, and it ignores missing-record errors for deletes. When using BATCHERRORMODE, use the HANDLECOLLISIONS parameter to prevent Replicat from abending. ◆ NOBATCHERRORMODE, the default, causes Replicat to disable BATCHSQL processing temporarily when there is an error, and then retry the transaction first in normal mode and then, if normal mode fails, in source mode (same transaction boundaries as on the source).
BATCHESPERQUEUE <n>	Controls the maximum number of batches that one memory queue can contain. After BATCHESPERQUEUE is reached, a target transaction is executed. ◆ Minimum value is 1. ◆ Maximum value is 1000. ◆ Default is 50.

Option	Description
`BATCHTRANSOPS <n>`	Controls the maximum number of batch operations that can be grouped into a transaction before requiring a commit. When BATCHTRANSOPS is reached, the operations are applied to the target. Set BATCHTRANSOPS to the default of 1000 or higher. ◆ Minimum value is 1. ◆ Maximum value is 100000. ◆ Default is 1000.
`BYTESPERQUEUE <n>`	Sets the maximum number of bytes that one queue can contain. After BYTESPERQUEUE is reached, a target transaction is executed. ◆ Minimum value is 1000000 bytes (1 megabyte). ◆ Maximum value is 1000000000 bytes (1 gigabyte). ◆ Default is 2000000 bytes (20 megabytes).
`OPSPERBATCH <n>`	Sets the maximum number of row operations that one batch can contain. After OPSPERBATCH is reached, a target transaction is executed. ◆ Minimum value is 1. ◆ Maximum value is 100000. ◆ Default is 1200.
`OPSPERQUEUE <n>`	Sets the maximum number of row operations in all batches that one queue can contain. After OPSPERQUEUE is reached, a target transaction is executed. ◆ Minimum value is 1. ◆ Maximum value is 100000. ◆ Default is 1200.
`TRACE`	Enables detailed tracing of BATCHSQL activity to the console and to the report file. Do not set tracing without the guidance of an Oracle Support analyst.

Example

```
BATCHSQL BATCHESPERQUEUE 100, OPSPERBATCH 2000
```

BEGIN

Valid for Replicat

Use the BEGIN parameter to start processing with the first record in the database transaction log or Oracle GoldenGate trail that has a timestamp greater than, or equal to, the time specified with BEGIN. All subsequent records, including records where the timestamp is less than the specified time, are processed. Use BEGIN when SPECIALRUN is specified for the same Replicat group.

Default None

Syntax

```
BEGIN <date>[<time>]
```

Argument	Description
`<date>[<time>]`	Specifies when to begin processing. Valid values: ◆ A date and optional time in the format of yyyy-mm-dd:hh:mi[:ss[.cccccc]] based on a 24-hour clock. Seconds and centiseconds are optional.

Example

```
BEGIN 2011-01-01:04:30:00
```

BINARYCHARS | NOBINARYCHARS

Valid for Extract and Replicat

Use BINARYCHARS and NOBINARYCHARS to control whether character data is treated as binary data or null-terminated strings.

BINARYCHARS, the default, maintains data the way it was entered in the source table. This ensures proper processing in cases when a column in the source or target database is defined as a character column and it is possible that binary characters could be entered into that column. BINARYCHARS is not compatible with the BULKLOAD parameter (direct-bulk load); use NOBINARYCHARS.

NOBINARYCHARS can cause Oracle GoldenGate to interpret a binary character to be the end of the data in that column. If there is more data after the binary data, it is not processed by Oracle GoldenGate, compromising data integrity. NULL characters cause this to happen, as well as any character defined with the DELIMITER option of FORMATASCII. Unless there is good reason to use NOBINARYCHARS, leaving the default set to BINARYCHARS is recommended so that data is maintained the way it was entered in the source table. Before using NOBINARYCHARS, contact Oracle Support.

BINARYCHARS and NOBINARYCHARS are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements until the other is encountered.

Default BINARYCHARS

Syntax

```
BINARYCHARS | NOBINARYCHARS
```

BLOBMEMORY

This parameter is an alias for LOBMEMORY (see page 226).

BOOTDELAYMINUTES

Valid for Manager

Use the BOOTDELAYMINUTES parameter on a Windows system to delay the activities that Manager performs when it starts, such as executing parameters. For example, BOOTDELAYMINUTES can be used to delay AUTOSTART parameters until database services are started.

Specify BOOTDELAYMINUTES before other parameter entries. This parameter only supports Windows.

Default None (no delay)

Syntax `BOOTDELAYMINUTES <minutes>`

Argument	Description
`<minutes>`	The number of minutes to delay after system startup before starting Oracle GoldenGate processing.

Example `BOOTDELAYMINUTES 5`

BR

Valid for Extract (Oracle only)

Use the BR parameter to control the Bounded Recovery (BR) feature. This feature currently supports Oracle databases.

Bounded Recovery is a component of the general Extract checkpointing facility. It guarantees an efficient recovery after Extract stops for any reason, planned or unplanned, no matter how many open (uncommitted) transactions there were at the time that Extract stopped, nor how old they were. Bounded Recovery 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.

WARNING Before changing this parameter from its default settings, contact Oracle Support for guidance. Most production environments will not require changes to this parameter. You can, however, specify the directory for the Bounded Recovery checkpoint files without assistance.

How Extract recovers open transactions

When Extract encounters the start of a transaction in the redo log (in Oracle, this is the first executable SQL statement) it starts caching to memory all of the data that is specified to be captured for that transaction. Extract must cache a transaction even if it contains no captured data, because future operations of that transaction might contain data that is to be captured.

When Extract encounters a commit record for a transaction, it writes the entire cached transaction to the trail and clears it from memory. When Extract encounters a rollback record for a transaction, it discards the entire transaction from memory. Until Extract processes a commit or rollback, the transaction is considered *open* and its information continues to be collected.

If Extract stops before it encounters a commit or rollback record for a transaction, all of the cached information must be recovered when Extract starts again. This applies to all transactions that were open at the time that Extract stopped.

Extract performs this recovery as follows:

- If there were no open transactions when Extract stopped, the recovery begins at the current Extract read checkpoint. This is a normal recovery.

- If there were open transactions whose start points in the log were very close in time to the time when Extract stopped, Extract begins recovery by re-reading the logs from the *beginning of the oldest open transaction*. This requires Extract to do redundant work for transactions that were already written to the trail or discarded before Extract stopped, but that work is an acceptable cost given the relatively small amount of data to process. This also is considered a normal recovery.
- If there were one or more transactions that Extract qualified as *long-running open transactions*, Extract begins its recovery with a *Bounded Recovery*.

How Bounded Recovery works

A transaction qualifies as long-running if it has been open longer than one *Bounded Recovery interval*, which is specified with the BRINTERVAL option of the BR parameter. For example, if the Bounded Recovery interval is four hours, a long-running open transaction is any transaction that started more than four hours ago.

At each Bounded Recovery interval, Extract makes a *Bounded Recovery checkpoint*, which persists the current state and data of Extract to disk, including the state and data (if any) of long-running transactions. If Extract stops after a Bounded Recovery checkpoint, it will recover from a position within the previous Bounded Recovery interval or at the last Bounded Recovery checkpoint, instead of processing from the log position where the oldest open long-running transaction first appeared.

The *maximum Bounded Recovery time* (maximum time for Extract to recover to where it stopped) is never more than twice the current Bounded Recovery checkpoint interval. The actual recovery time will be a factor of the following:

- the time from the last valid Bounded Recovery interval to when Extract stopped.
- the utilization of Extract in that period.
- the percent of utilization for transactions that were previously written to the trail. Bounded Recovery processes these transactions much faster (by discarding them) than Extract did when it first had to perform the disk writes. This constitutes most of the reprocessing that occurs for transactional data.

When Extract recovers, it restores the persisted data and state that were saved at the last Bounded Recovery checkpoint (including that of any long running transactions).

For example, suppose a transaction has been open for 24 hours, and suppose the Bounded Recovery interval is four hours. In this case, the maximum recovery time will be no longer than eight hours worth of Extract processing time, and is likely to be less. It depends on when Extract stopped relative to the last valid Bounded Recovery checkpoint, as well as Extract activity during that time.

Problems solved by Bounded Recovery

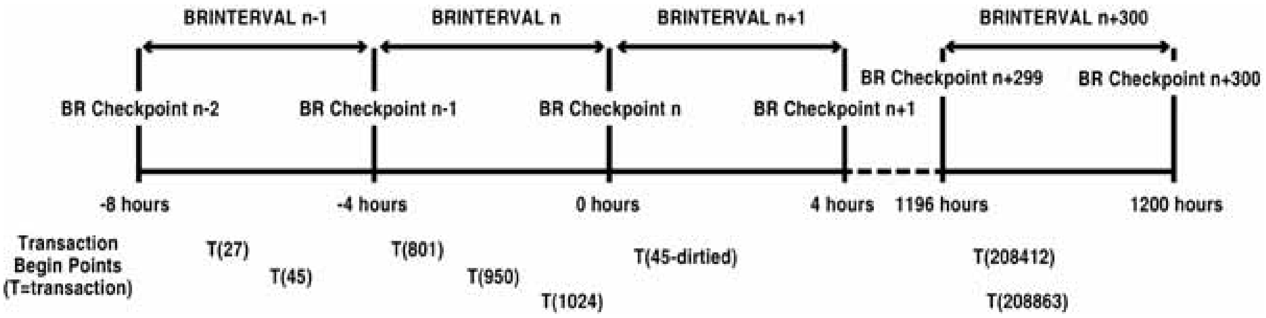
The use of disk persistence to store and then recover long-running transactions enables Extract to manage a situation that rarely arises but would otherwise significantly (adversely) affect performance if it occurred. The beginning of a long-running transaction is often very far back in time from the place in the log where Extract was processing when it stopped. A long-running transaction can span numerous old logs, some of which might no longer reside on accessible storage or might even have been deleted. Not only would it take an unacceptable amount of time to read the logs again from the start of a long-running transaction but, since long-running transactions are rare, most of that work would be the redundant capture of other transactions that were already written to the trail or discarded.

Being able to restore the state and data of persisted long-running transactions eliminates that work.

Bounded Recovery example

The following diagram illustrates a timeline over which a series of transactions were started. It shows how long-running open transactions are persisted to disk after a specific interval and then recovered after a failure. It will help to understand the terminology used in the example:

- A *persisted object* is any object in the cache that was persisted at a Bounded Recovery checkpoint. Typically this is the transactional state or data, but the cache is also used for objects that are internal to Extract. These are all collectively referred to as objects.
- The *oldest non-persisted object* is the oldest open object in the cache in the interval that immediately precedes the current Bounded Recovery checkpoint. Typically this is the oldest open transaction in that interval. Upon the restart of Bounded Recovery, runtime processing resumes from the position of the oldest non-persisted object, which, in the typical case of transactions, will be the position in the redo log of that transaction.



In this example, the Bounded Recovery interval is four hours. An open transaction is persisted at the current Bounded Recovery checkpoint if it has been open for more than one Bounded Recovery interval from the current Bounded Recovery checkpoint.

At BR Checkpoint n:

- There are five open transactions: T(27), T(45), T(801), T(950), T(1024). All other transactions were either committed and sent to the trail or rolled back. Transactions are shown at their start points along the timeline.
- The transactions that have been open for more than one Bounded Recovery interval are T(27) and T(45). At **BR Checkpoint n**, they are persisted to disk.
- The oldest non-persisted object is T(801). It is not eligible to be persisted to disk, because it has not been open across at least one Bounded Recovery interval. As the oldest non-persisted object, its start position in the log is stored in the **BR Checkpoint n** checkpoint file. If Extract stops unexpectedly after **BR Checkpoint n**, it will recover to that log position and start to re-read the log there. If there is no oldest non-persisted object in the preceding Bounded Recovery interval, Extract will start re-reading the log at the log position of the current Bounded Recovery checkpoint.

At BR Checkpoint n+1:

- T(45) was dirtied (updated) in the previous Bounded Recovery interval, so it gets written to a new persisted object file. The old file will be deleted after completion of **BR Checkpoint n+1**.
- If Extract fails while writing **BR Checkpoint n+1** or at any time within that Bounded Recovery checkpoint interval between **BR Checkpoint n** and **BR Checkpoint n+1**, it will recover from **BR Checkpoint n**, the last valid checkpoint. The restart position for **BR Checkpoint n** is the start of the oldest non-persisted transaction, which is T(801). Thus, the worst-case recovery time is always no more than two Bounded Recovery intervals from the point where Extract stopped, in this case no more than eight hours.

At BR Checkpoint n+3000

- The system has been running for a long time. T(27) and T(45) remain the only persisted transactions. T(801) and T(950) were committed and written to the trail sometime before **BR Checkpoint n+2999**. Now, the only open transactions are T(208412) and T(208863).
- **BR Checkpoint n+3000** is written.
- There is a power failure in the interval after **BR Checkpoint n+3000**.
- The new Extract recovers to **BR Checkpoint n+3000**. T(27) and T(45) are restored from their persistence files, which contain the state from **BR Checkpoint n**. Log reading resumes from the beginning of T(208412).

Managing long-running transactions

Oracle GoldenGate provides the following parameters and commands to manage long-running transactions:

- Use the WARNLONGTRANS parameter to specify a length of time that a transaction can be open before Extract generates a warning message that the transaction is long-running. Also use WARNLONGTRANS to control the frequency with which Oracle GoldenGate checks for long-running transactions. Note that this setting is independent of, and does not affect, the Bounded Recovery interval.
- Use the SEND EXTRACT command with the SKIPTRANS option to force Extract to skip a specified transaction.
- Use the SEND EXTRACT command with the FORCETRANS option to force Extract to write a specified transaction to the trail as a committed transaction.
- Use the TRANLOGOPTIONS parameter with the PURGEORPHANEDTRANSACTIONS option to enable the purging of orphaned transactions that occur when a node fails and Extract cannot capture the rollback.

About the files that are written to disk

At the expiration of a Bounded Recovery interval, Extract always creates a Bounded Recovery checkpoint file. Should there be long-running transactions that require persistence, they each are written to their own persisted-object files. A persisted-object file contains the state and data of a single transaction that is persisted to disk.

Field experience has shown that the need to persist long running transactions is rare, and that the transaction is empty in most of those cases.

If a previously persisted object is still open and its state and/or data have been modified during the just-completed Bounded Recovery interval, it is re-persisted to a new persisted object file. Otherwise, previously persisted object files for open transactions are not changed.

It is theoretically possible that more than one persisted file might be required to persist a long-running transaction.

NOTE The Bounded Recovery files cannot be used to recover the state of Extract if moved to another system, even with the same database, if the new system is not identical to the original system in all relevant aspects. For example, checkpoint files written on an Oracle 11g Solaris platform cannot be used to recover Extract on an Oracle 11g Linux platform.

Circumstances that change Bounded Recovery to normal recovery

Most of the time, Extract uses normal recovery and not Bounded Recovery, the exception being the rare circumstance when there is a persisted object or objects. Certain abnormal circumstances may prevent Extract from switching from Bounded Recovery back to normal recovery mode. These include, but are not limited to, such occurrences as physical corruption of the disk (where persisted data is stored for long-running transactions), inadvertent deletion of the Bounded Recovery checkpoint files, and other actions or events that corrupt the continuity of the environment. There may also be more correctable reasons for failure.

In all but a very few cases, if Bounded Recovery fails during a recovery, Extract switches to normal recovery. After completing the normal recovery, Bounded Recovery is turned on again for runtime.

Bounded Recovery is not invoked under the following circumstances:

- The Extract start point is altered by CSN or by time.
- The Extract I/O checkpoint altered.
- The Extract parameter file is altered during recovery, such as making changes to the TABLE specification.

After completion of the recovery, Bounded Recovery will be in effect again for the next run.

What to do if Extract abends during Bounded Recovery

If Extract abends in Bounded Recovery, examine the error log to determine the reason. It might be something that is quickly resolved, such as an invalid parameter file or incorrect privileges on the directory that contains the Bounded Recovery files. In such cases, after the problem is fixed, Extract can be restarted with Bounded Recovery in effect.

If the problem is not correctable, restart Extract with the BRRESET command in GGSCI. Extract will recover in normal recovery mode and then turn on Bounded Recovery again.

Modifying the BR parameter

Bounded Recovery is enabled by default with a default Bounded Recovery interval of four hours (as controlled with the BRINTERVAL option). This interval should be appropriate for most environments. Do not alter the BR parameter without first obtaining guidance from an Oracle support analyst. Bounded Recovery runtime statitistics are available to help Oracle GoldenGate analysts evaluate the Bounded Recovery usage profile to determine the proper setting for BRINTERVAL in the unlikely event that the default is not sufficient.

Should you be requested to alter BR, be aware that the Bounded Recovery interval is a multiple of the regular Extract checkpoint interval. The Extract checkpoint is controlled by the CHECKPOINTSECS parameter. Thus, the BR parameter controls the ratio of the Bounded Recovery checkpoint to the regular Extract checkpoint. You might need to change both parameters, if so advised by your Oracle representative.

Supported databases

This parameter applies to Oracle databases. For other databases, Extract recovers by reading the old logs from the start point of the oldest open transaction at the point of failure and does not persist long-running transactions.

Default `BR BRINTERVAL 4, BRDIR BR`

Syntax

```
BR
[, BRDIR <directory>]
[, BRINTERVAL <interval><unit>]
[, BRKEEPSTALEFILES]
[, BROFF]
[, BROFFONFAILURE]
[, BRRESET]
```

Argument	Description
`BRDIR <directory>`	Specifies the relative or full path name of the parent directory that will contain the BR directory. The BR directory contains the Bounded Recovery checkpoint files, and the name of this directory cannot be changed. The default parent directory for the BR directory is a directory named BR in the root directory that contains the Oracle GoldenGate installation files. Each Extract group within a given Oracle GoldenGate installation will have its own sub-directory under the directory that is specified with BRDIR. Each of those directories is named for the associated Extract group. For <directory>, do not use any name that contains the string "temp" or "tmp" (case-independent). Temporary directories are subject to removal during internal or external cleanup procedures.

Argument	Description
`BRINTERVAL <interval><unit>`	Specifies the time between Bounded Recovery checkpoints. This is known as the *Bounded Recovery interval*. This interval is an integral multiple of the standard Extract checkpoint interval, as controlled by the CHECKPOINTSECS parameter. However, it need not be set exactly. Bounded Recovery will adjust any legal BRINTERVAL parameter internally as it requires. The minimum for <interval> is 20 minutes. The maximum is 96 hours. <unit> can be: ◆ M for minutes ◆ H for hours The default interval is 4 hours.
`BRKEEPSTALEFILES`	Causes old Bounded Recovery checkpoint files to be retained. By default, only current checkpoint files are retained. Extract cannot recover from old Bounded Recovery checkpoint files. Retain old files only at the request of an Oracle support analyst.
`BROFF`	Turns off Bounded Recovery for the run and for recovery. Consult Oracle Support before using this option. In most circumstances, when there is a problem with Bounded Recovery, it turns itself off.
`BROFFONFAILURE`	Disables Bounded Recovery after an error. By default, if Extract encounters an error during Bounded Recovery processing, it reverts to normal recovery, but then enables Bounded Recovery again after recovery completes. BROFFONFAILURE turns Bounded Recovery off for the runtime processing.
`BRRESET`	**Note:** To use this option, you must start Extract from the command line. To run Extract from the command line: `replicat paramfile <name>.prm reportfile <name>.rpt` Where: paramfile <name>.prm is the relative or fully qualified name of the Extract parameter file. The command name can be abbreviated to pf. reportfile <name>.rpt is the relative or fully qualified name of the Extract report file, if you want it in a place other than the default. The command name can be abbreviated to rf. BRRESET forces Extract to use normal recovery for the current run, and then turn Bounded Recovery back on after the recovery is complete. Its purpose is for the rare cases when Bounded Recovery does not revert to normal recovery if it encounters an error. Bounded Recovery will be enabled during runtime. Consult Oracle before using this option.

Example BR BRDIR /user/checkpt/br specifies that the Bounded Recovery checkpoint files will be created in the /user/checkpt/br directory.

BULKLOAD

Valid for Replicat

Use the BULKLOAD parameter for an initial load Replicat when using the direct bulk load to SQL*Loader method. This method passes initial-load data directly to the interface of Oracle's SQL*Loader utility to perform a direct load. A Collector process and trails are not used.

When using BULKLOAD, use the NOBINARYCHARS parameter in the Extract parameter file. Contact Oracle Support before using NOBINARYCHARS. For more information, go to http://support.oracle.com.

For a complete guide to the methods of loading data with Oracle GoldenGate, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Default None

Syntax `BULKLOAD`

CACHEMGR

Valid for Extract (all databases except DB2 on z/OS and NonStop SQL/MX)

Use the CACHEMGR parameter to control the amount of virtual memory and temporary disk space that is available for caching uncommitted transaction data.

> **WARNING** Before changing this parameter from its default cache settings, contact Oracle Support for guidance. The cache manager of Oracle GoldenGate is internally self-configuring and self-adjusting, and most production environments will not require changes to this parameter. You can, however, specify the directory for the page files without assistance.

About memory management

> **NOTE** While described as accurately as possible here, the underlying design of the memory management component is always subject to changes that may be required by ongoing product improvements.

Because Oracle GoldenGate replicates only committed transactions, it stores the operations of each transaction in a managed virtual-memory pool known as a *cache* until it receives either a commit or a rollback for that transaction. One global cache operates as a shared resource of an Extract process. The following sub-pools of virtual memory are allocated from the global cache:

- One sub-pool per log reader thread for most transaction row data.
- One sub-pool for BLOB data and possibly other large items.

Within each sub-pool, individual buffers are allocated from the global cache. Each buffer contains information that is relative to a transaction that is being processed by Oracle GoldenGate.

The Oracle GoldenGate cache manager takes advantage of the memory management functions of the operating system to ensure that Oracle GoldenGate processes work in a sustained and efficient manner. Within its cache, it makes use of modern virtual memory techniques by:

- Allocating and managing active buffers efficiently.
- Recycling old buffers instead of paging to disk, when possible.
- Paging less-used information to disk, when necessary.

The actual amount of physical memory that is used by any Oracle GoldenGate process is controlled by the operating system, not the Oracle GoldenGate program.

The cache manager keeps an Oracle GoldenGate process working within the soft limit of its global cache size, only allocating virtual memory on demand. (The cache manager does not allocate physical memory, over which it has no control.) System calls to increase the cache size are made only as a last resort and, when used, are always followed by the release of virtual memory back to the system.

The system must have sufficient swap space for each Oracle GoldenGate Extract and Replicat process that will be running. To determine the required swap space:

1. Start one Extract process and one Replicat process.
2. Run GGSCI.
3. View the report file of each running process and find the line PROCESS VM AVAIL FROM OS (min).
4. Round up each value to the next full gigabyte if needed. For example, round up 1.76GB to 2 GB.
5. Multiply the rounded-up Extract value by the number of Extract processes.
6. Multiply the rounded-up Replicat value by the number of Replicat processes.
7. Add the two results, plus any additional swap space required by other Oracle GoldenGate processes and other processes on the system.

   ```
   (<PROCESS VM> x <n_Extracts>) + (<PROCESS VM> x <n_Replicats>) +
   (<swap for other processes>) = Max swap space on system
   ```

   This sum is the maximum amount of swap space that could be needed for those processes. To determine the number of processes you will need, consult the configuration chapters in the *Oracle GoldenGate Windows and UNIX Administrator's Guide*.

The actual amount of physical memory that is used by any Oracle GoldenGate process is controlled by the operating system, not the Oracle GoldenGate process. The global cache size is controlled by the CACHESIZE option of CACHEMGR.

NOTE The cache manager is also used internally by Oracle GoldenGate for other purposes besides the Extract BLOB sub-pool and the sub-pool for other transaction data. You may see these additional memory pools when you view the statistics.

When to adjust CACHEMGR

The memory manager generates statistics that can be viewed with the SEND EXTRACT or SEND REPLICAT command when used with the CACHEMANAGER option. The statistics show the size of the memory pool, the paging frequency, the size of the transactions, and other information that creates a system profile.

Based on this profile, you might need to make adjustments to the memory cache if you see performance problems that appear to be related to file caching. The first step is to modify the CACHESIZE and CACHEPAGEOUTSIZE parameters. You might need to use a higher or lower cache size, a higher or lower page size, or a combination of both, based on the size and type of transactions that are being generated.

It is possible, however, that operating system constraints could limit the effect of modifying any components of the CACHEMGR parameter. In particular, if the operating system has a small per-process virtual memory limit, it will force more file caching, regardless of the CACHEMGR configuration.

For more information about using the cache manager statistics, see “SEND EXTRACT” on page 36.

Viewing basic statistics in the report file

Upon completing its initialization, the cache manager writes the following statistics to the Extract report file:

```
CACHEMGR virtual memory values (may have been adjusted)
CACHESIZE: 1G
CACHEPAGEOUTSIZE (normal): 4M
PROCESS VM AVAIL FROM OS (min): 1.79G
CACHESIZEMAX (strict force to disk): 1.58G
```

Where:

- CACHESIZE shows the soft limit of virtual memory that is available to Extract for caching transaction data. It is determined dynamically, based on the value of PROCESS VM AVAIL FROM OS (min). It can be controlled with the CACHESIZE option of CACHEMGR.
- CACHEPAGEOUTSIZE (normal) shows the threshold above which data from a transaction can be paged to disk, if needed. It can be controlled with the CACHEPAGEOUTSIZE option of CACHEMGR.
- PROCESS VM AVAIL FROM OS (min) shows the approximate amount of virtual memory that the process has determined it can use. For internal reasons, this amount may be less than what the operating system shows as being available.
- CACHESIZEMAX (strict force to disk) is derived from PROCESS VM AVAIL FROM OS and CACHESIZE. It can be understood in terms of how the cache manager determines which transactions are eligible to be paged out to disk. Normally, only those whose current virtual memory buffers exceed CACHEPAGEOUTSIZE are eligible to be paged. When the total memory requested exceeds CACHESIZE, the cache manager looks for transactions to write to disk and chooses them from the list of eligible ones. If the eligible ones have been paged to disk already, and the virtual memory in use now exceeds CACHESIZEMAX (strict force to disk), then any transaction that requires

additional buffers can be eligible for paging. This guarantees that virtual memory will always be available. Once the use of memory drops below CACHESIZEMAX, the CACHEPAGEOUTSIZE rule applies again.

Identifying the paging directory

By default, Oracle GoldenGate maintains data that it swaps to disk in the dirtmp subdirectory of the Oracle GoldenGate installation directory. The cache manager assumes that all of the free space on the file system is available. You can assign a directory by using the CACHEDIRECTORY option of the CACHEMGR parameter.

Guidelines for using CACHEMGR

- This parameter is valid for all databases except DB2 on z/OS and NonStop SQL/MX.
- At least one argument must be supplied. CACHEMGR by itself is invalid.
- Parameter options can be listed in any order.
- Only one CACHEMGR parameter is permitted in a parameter file.
- To use this parameter correctly (other than specifying the directory for the page files), you must know the profile of the system and the kinds of transactions that are being propagated from your applications. In normal environments, you should not need to change this parameter, because the cache manager is self-adjusting. If you feel that an adjustment is warranted, please open an Oracle service request for assistance. For more information, go to http://support.oracle.com.

Default None

Syntax

```
CACHEMGR {
    [, CACHESIZE <size>]
    [, CACHEDIRECTORY <path> [<size>] [, ...]]
    [, CACHEPAGEOUTSIZE <size>]
    }
```

Argument	Description
CACHESIZE <size>	Sets a soft limit for the amount of virtual memory (cache size) that is available for caching transaction data. On 64-bit systems, the default is 64 GB. On 32-bit systems, the cache size is determined dynamically by the cache manager. The memory is allocated on demand. By default, the cache manager dynamically determines the amount of virtual memory that is available to it from the operating system and determines the appropriate cache size. The available virtual memory is reported with the PROCESS VM AVAIL FROM OS value in the report file. The CACHESIZE value will be sized down if it is larger than, or sufficiently close to, the amount of virtual memory that is available to the process. However, for systems with large address spaces, the cache manager does no further determination once an internal limit is reached.

Argument	Description
	The CACHESIZE value will always be a power of two, rounded down from the value of PROCESS VM AVAIL FROM OS, unless the latter is itself a power of two, in which case it is halved. After the specified size is consumed by data, the memory manager will try to free up memory by paging data to disk or by reusing aged buffers, before requesting more memory from the system. If a transaction's cache virtual memory requirements grow beyond the initial buffer allocation, the additional amount of cache memory that is allocated by the cache manager is determined dynamically based on factors such as the current size of the cached data of this transaction, the size needed for the new data, and the amount of virtual memory that is being used conjointly by all the transactions.
`CACHEPAGEOUTSIZE <size>`	Note: Before changing this parameter, contact Oracle Support. This parameter sets a threshold above which data from a transaction can be paged to disk, if needed. To avoid using system overhead unnecessarily, the cache manager will page a transaction to the file system only if the following are true: ◆ the transaction is eligible to be paged. ◆ there is not sufficient free virtual memory, based on CACHESIZE, to satisfy new memory requests. In exceptional cases, other transactions may temporarily become eligible for file caching. The default size is approximately 8 MB. Avoid using values below 1 MB. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k If you believe that there are performance problems on the system, changing this parameter might help; otherwise, it should be left to its default. The paging threshold might become a performance issue in environments with many concurrent large transactions, or with one or two very large and long-running transactions. An example would be a large transaction with much LOB data in a system with limited memory. In these cases, the cache manager statistics should be consulted to determine: ◆ the cached transaction size distribution in the BLOB memory pool. ◆ the log reader memory pool(s). ◆ the statistics on file caching.

Argument	Description
`CACHEDIRECTORY <path> [<size>]`	Specifies the name of the directory to which Oracle GoldenGate writes transaction data to disk temporarily when necessary. The default without this parameter is the dirtmp sub-directory of the Oracle GoldenGate installation directory. Any directory for temporary files can be on an Oracle Database File System, but cannot be on a direct I/O or concurrent I/O mounted file system that does not support the mmap() system call, such as AIX. ◆ <path> is a fully qualified directory name. ◆ <size> sets a maximum amount of disk space that can be allocated to the specified directory. The upper limit is that which is imposed by the file system, such as maximum file size or number of files. The minimum size is 2 GB, which is enforced. There is no default. Do not use this option unless you must constrain the swap space that is used by Oracle GoldenGate because of resource limitations. You can specify more than one directory by using a CACHEDIRECTORY clause for each one. The maximum number of directories is 100. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k

Example

```
CACHEMGR CACHESIZE 500MB, CACHEDIRECTORY /ggs/temp 2GB, CACHEDIRECTORY
/ggs2/temp 2GB
```

CHARSET

Valid for Extract, Replicat, DEFGEN, Manager, GLOBALS

Use the CHARSET parameter to specify the character set of the parameter files in the local Oracle GoldenGate instance. By default, the parameter file is created in the default character set of the local operating system. CHARSET specifies an alternative character set to use in the event that the local platform does not support a required character or characters.

CHARSET allows operating-system incompatible characters, including multi byte characters, to be used in the parameter file without the need for an escape sequence (\uXXXX) when the local platform does not support multibyte characters as the default character set of the operating system.

CHARSET can also be used when the parameter file is being created on one system but will be used on a different system with a different character set. To avoid possible incompatibilities between different character sets, you should create parameter files on the same system where they will be used by Oracle GoldenGate.

Placement in the parameter file

CHARSET must be placed on the first line of the parameter file.

Usage in the GLOBALS file

CHARSET used in a GLOBALS file sets a default character set for the parameter files of all local processes. CHARSET used in any individual parameter file overrides the default that is set in GLOBALS.

Usage in nested parameter files

You can use CHARSET in a parameter file that includes an OBEY or INCLUDE parameter, but the referenced parameter file does not inherit the CHARSET character set. The CHARSET character set is used to read wildcarded object names in the referenced file, but you must use an escape sequence (\uXXXX) to specify all other incompatible characters in the referenced file.

Default None

Syntax `CHARSET <character_set>`

Argument	Description
`<character_set>`	Any supported character set.

Example `CHARSET UTF-8`

CHARSETCONVERSION | NOCHARSETCONVERSION

Valid for Replicat

Use the CHARSETCONVERSION and NOCHARSETCONVERSION parameters to control whether Replicat performs character-set conversion when replicating character-type data between databases that have different character sets. Replicat performs the conversion by default in most, but not all, cases. For more information, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

NOCHARSETCONVERSION prevents Replicat from performing the conversion of character sets and normally should not be used, because it raises the risk of data-integrity errors. One circumstance where NOCHARSETCONVERSION might be appropriate is when you are certain that all of the data is ASCII, and that no ASCII-incompatible characters exist in character columns. If you are not sure whether NOCHARSETCONVERSION is appropriate in your environment, contact Oracle Support before using it.

NOTE For an Oracle target to perform the conversion instead of Replicat, the Replicat parameter file must contain a SETENV parameter that sets the NLS_LANG environment variable to the character set of the *source* database.

Default For non-Oracle targets, the default is CHARSETCONVERSION. For Oracle targets, the default is NOCHARSETCONVERSION unless the captured source data is EBCDIC from DB2 z/OS: In the case of EBCDIC, the default is CHARSETCONVERSION

Syntax `CHARSETCONVERSION | NOCHARSETCONVERSION`

CHECKMINUTES

Valid for Manager

Use the CHECKMINUTES parameter to control how often Manager performs maintenance activities. Decreasing this parameter can significantly affect performance if trail files roll over frequently. Other events, such as processes ending abnormally, also trigger the maintenance cycle.

Default Every 10 minutes

Syntax `CHECKMINUTES <minutes>`

Argument	Description
`<minutes>`	The frequency, in minutes, to perform maintenance activities.

Example `CHECKMINUTES 15`

CHECKPARAMS

Valid for Extract and Replicat

Use the CHECKPARAMS parameter to test the syntax of a parameter file. To start the test:

1. Edit the parameter file to add CHECKPARAMS.
2. (Optional) To verify the tables, add the NODYNAMICRESOLUTION parameter.
3. Start the process. Without processing data, Oracle GoldenGate audits the syntax. If NODYNAMICRESOLUTION exists, Oracle GoldenGate connects to the database to verify that the tables specified with TABLE or MAP exist. If there is a syntax failure, the process abends with error 190. If the syntax succeeds, the process stops and writes a message to the report file that the parameters processed successfully.
4. Do one of the following:
   - If the test succeeds, edit the file to remove the CHECKPARAMS parameter and the NODYNAMICRESOLUTION parameter, if used, and then start the process again to begin processing.
   - If the test fails, edit the parameter file to fix the syntax based on the report's findings, and then remove NODYNAMICRESOLUTION and start the process again.

CHECKPARAMS can be positioned anywhere within the parameter file.

Default None

Syntax `CHECKPARAMS`

CHECKPOINTSECS

Valid for Extract and Replicat

Use the CHECKPOINTSECS parameter to control how often Extract and Replicat make their routine checkpoints.

- Decreasing the value causes more frequent checkpoints. This reduces the amount of data that must be reprocessed if the process fails, but it could cause performance degradation because data is written to disk more frequently.
- Increasing the value causes less frequent checkpoints. This might improve performance, but it increases the amount of data that must be reprocessed if the process fails. When using less frequent Extract checkpoints, make certain that the transaction logs remain available in case the data has to be reprocessed.

NOTE In addition to its routine checkpoints, Replicat also makes a checkpoint when it commits a transaction.

Avoid changing CHECKPOINTSECS unless you first open an Oracle service request. For more information, go to http://support.oracle.com.

Default 10 seconds

Syntax `CHECKPOINTSECS <seconds>`

Argument	Description
`<seconds>`	The number of seconds to wait before issuing a checkpoint.

Example `CHECKPOINTSECS 20`

CHECKPOINTTABLE

Valid for GLOBALS

Use the CHECKPOINTTABLE parameter in a GLOBALS parameter file to specify the name of a default checkpoint table that can be used by all Replicat groups in one or more Oracle GoldenGate instances. All Replicat groups created with the ADD REPLICAT command will default to this table unless it is overridden by using the CHECKPOINTTABLE option of that command.

To create the checkpoint table, use the ADD CHECKPOINTTABLE command in GGSCI.

Default None

Syntax `CHECKPOINTTABLE <owner.table>`

Argument	Description
`owner.table`	The owner and name of the checkpoint table.

Example `CHECKPOINTTABLE ggs.chkpt`

CMDTRACE

Valid for Extract and Replicat

Use the CMDTRACE parameter to display macro expansion steps in the report file. You can use this parameter more than once in the parameter file to set different options for different macros.

Default `OFF`

Syntax `CMDTRACE [ON | OFF | DETAIL]`

Argument	Description
`ON`	Enables the display of macro expansion.
`OFF`	Disables the display of macro expansion.
`DETAIL`	Produces a verbose display of macro expansion.

Example In the following example, tracing is enabled before #testmac is invoked, and then disabled after the macro's execution.

```
MACRO #testmac
BEGIN
col1 = col2,
col3 = col4
END;
...
CMDTRACE ON
MAP test.table2 , TARGET test.table2,
COLMAP (#testmac);
CMDTRACE OFF
```

COLMATCH

Valid for Extract and Replicat

Use the COLMATCH parameter to create global rules for column mapping. COLMATCH rules apply to all TABLE or MAP statements that follow the COLMATCH statement. Global rules can be turned off for subsequent TABLE or MAP entries with the RESET option.

With COLMATCH, you can map between tables that are similar in structure but 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 a COLMAP clause in individual TABLE or MAP statements.

With COLMATCH, you can:

- Map explicitly based on column names.
- Ignore name prefixes or suffixes.

Either COLMATCH or a COLMAP clause of a TABLE or MAP statement is required when mapping differently named source and target columns.

See the Oracle GoldenGate *Windows and UNIX Administrator's Guide* for more information about mapping columns.

Default None

Syntax

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

Argument	Description
`NAMES <target column> = <source column>`	Specifies the name of a target and source column, for example CUSTOMER_CODE and CUST_CODE.
`PREFIX <prefix>`	Specifies a column name prefix to ignore. For example, to map a target column named ORDER_ID to a source column named P_ORDER_ID, specify: `COLMATCH PREFIX P_`
`SUFFIX <suffix>`	Specifies a column name suffix to ignore. For example, to map a target column named CUST_CODE_K to a source column named CUST_CODE, specify: `COLMATCH SUFFIX _K`
`RESET`	Turns off previously defined COLMATCH rules for subsequent TABLE or MAP statements.

Example 1 `COLMATCH NAMES CUSTOMER_CODE = CUST_CODE`

Example 2 `COLMATCH PREFIX P_`

Example 3 `COLMATCH SUFFIX _K`

Example 4 `COLMATCH RESET`

COMMENT | --

Valid for Manager, Extract, Replicat

Use the COMMENT parameter or double hyphens (--) to indicate comments within a parameter file. Anything on the same line after COMMENT or double hyphens is ignored during processing.

COMMENT or double hyphens can be used to indicate a comment anywhere in the parameter file. Comments that continue to the next line must be preceded by another COMMENT keyword or double hyphens.

NOTE If any columns in the tables being synchronized contain the word "comment," there may be conflicts with the COMMENT parameter. Use double hyphens instead.

Default None

Syntax

```
{COMMENT <comment text>} | {-- <comment text>}
```

Example 1 `COMMENT GoldenGate param file for fin Extract group.`

Example 2 `-- GoldenGate param file for fin Extract group.`

COMPRESSDELETES | NOCOMPRESSDELETES

Valid for Extract

Use the COMPRESSDELETES and NOCOMPRESSDELETES parameters to control the way columns are written to the trail record for delete operations.

COMPRESSDELETES, the default, causes Extract to write only the primary key to the trail for delete operations. The key provides enough information to delete the correct target record, while restricting the amount of data that must be processed.

NOCOMPRESSDELETES sends all of the columns to the trail. This becomes the default when a table definition does not include a primary key or unique index. If a substitute key was defined with the KEYCOLS option of TABLE, then those columns are written to the trail, whether or not a real key was defined.

NOCOMPRESSDELETES also is required when using the Conflict Detection and Resolution (CDR) feature for a DB2 database on any of the platforms that are supported by Oracle GoldenGate. For more information about CDR, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

COMPRESSDELETES and NOCOMPRESSDELETES can be used globally for all TABLE statements in the parameter file, or they can be used as on-off switches for individual TABLE statements.

These parameters do not affect data pumps.

Default `COMPRESSDELETES`

Syntax `COMPRESSDELETES | NOCOMPRESSDELETES`

COMPRESSUPDATES | NOCOMPRESSUPDATES

Valid for Extract

Use the COMPRESSUPDATES and NOCOMPRESSUPDATES parameters for Extract to control the way columns are written to the trail record for update operations.

COMPRESSUPDATES, the default, causes Extract to write only the primary key and the changed columns of a row to the trail for update operations. This provides enough information to update the correct target record, while restricting the amount of data that must be processed.

NOCOMPRESSUPDATES sends all of the columns to the trail. This becomes the default when a table definition does not include a primary key or unique index. If a substitute key was defined for that table with the KEYCOLS option of the TABLE parameter, then those columns are written to the trail, whether or not a real key was defined.

NOCOMPRESSUPDATES also is required when using the Conflict Detection and Resolution (CDR) feature for a DB2 database on any of the platforms that are supported by Oracle GoldenGate. For more information about CDR, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

COMPRESSUPDATES and NOCOMPRESSUPDATES apply globally for all TABLE statements in a parameter file.

This parameter supports only the following databases:

- DB2 LUW
- DB2 z/OS
- Teradata version 12 or later
- SQL Server
- Sybase

COMPRESSUPDATES and NOCOMPRESSUPDATES do not affect data pumps.

Default `COMPRESSUPDATES`

Syntax

```
COMPRESSUPDATES | NOCOMPRESSUPDATES
```

CUSEREXIT

Valid for Extract and Replicat

Use the CUSEREXIT parameter to call a custom exit routine written in C programming code from a Windows DLL or UNIX shared object at a defined exit point within Oracle GoldenGate processing. Your user exit routine must be able to accept different events and information from the Extract and Replicat processes, process the information as desired, and return a response and information to the caller (the Oracle GoldenGate process that called it).

You can employ user exits as an alternative to, or in conjunction with, the data transformation functions that are available within the Oracle GoldenGate solution.

For help with creating and implementing user exits, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Default None

Syntax

```
CUSEREXIT <DLL or shared object name> <routine name>
[, PASSTHRU]
[, INCLUDEUPDATEBEFORES]
[, PARAMS "<startup string>”]
```

Argument	Description
`<DLL or shared object name>`	The name of the Windows DLL or UNIX shared object that contains the user exit function.
`<routine name>`	The name of the exit routine to be executed.

Argument	Description
`PASSTHRU`	Valid only for an Extract data pump. It assumes that no database is required, and that no output trail is allowed. It expects that the user exit will perform all of the processing and that Extract will skip the record. Extract will perform all of the required data mapping before passing the record to the user exit. Instead of a reply status of EXIT_OK_VAL, the reply will be EXIT_PROCESSED_REC_VAL. All process statistics are updated as if the records were processed by Oracle GoldenGate.
`INCLUDEUPDATEBEFORES`	Passes the before images of column values to a user exit. When using this parameter, you must explicitly request the before image by setting the requesting_before_after_ind flag to BEFORE_IMAGE_VAL within a callback function that supports this flag. Otherwise, only the after image is passed to the user exit. By default, Oracle GoldenGate only works with after images. When using INCLUDEUPDATEBEFORES for a user exit that is called from a data pump or from Replicat, always use the GETUPDATEBEFORES parameter for the primary Extract process, so that the before image is captured, written to the trail, and causes a process_record event in the user exit. In a case where the primary Extract also has a user exit, GETUPDATEBEFORES causes both the before image and the after image to be sent to the user exit as separate EXIT_CALL_PROCESS_RECORD events. If the user exit is called from a primary Extract (one that reads the transaction log), only INCLUDEUPDATEBEFORES is needed for that Extract. GETUPDATEBEFORES is not needed in this case, unless other Oracle GoldenGate processes downstream will need the before image to be written to the trail. INCLUDEUPDATEBEFORES does not cause before images to be written to the trail.
`PARAMS "<startup string>”`	Passes the specified string at startup. Can be used to pass a properties file, startup parameters, or other string. The string must be enclosed within double quote marks. Data in the string is passed to the user exit in the EXIT_CALL_START exit_params_def.function_param. If no quoted string is specified with PARAMS, the exit_params_def.function_param is NULL.

Example 1

```
CUSEREXIT userexit.dll MyUserExit
```

Example 2

```
CUSEREXIT userexit.dll MyUserExit, PARAMS "init.properties"
```

Example 3

```
CUSEREXIT userexit.dll MyUserExit, INCLUDEUPDATEBEFORES, & PARAMS
"init.properties”
```

Example 4

```
CUSEREXIT userexit.dll MyUserExit, INCLUDEUPDATEBEFORES, PASSTHRU, & PARAMS
"init.properties"
```

Example 5

```
CUSEREXIT cuserexit.dll MyUserExit, & PASSTHRU, & INCLUDEUPDATEBEFORES, &
PARAMS "Some text to start with during startup"
```

DBOPTIONS

Valid for Extract and Replicat

Use the DBOPTIONS parameter to specify database options. This is a global parameter, applying to all TABLE or MAP statements in the parameter file.

DBOPTIONS must be placed before the TARGETDB or SOURCEDB parameter statement and/or the USERID statement. Some DBOPTIONS options apply only to Extract or Replicat.

Default None

Syntax

```
DBOPTIONS
[ALLOWLOBDATATRUNCATE | NOALLOWLOBDATATRUNCATE]
[ALLOWUNUSEDCOLUMN]
[CATALOGCONNECT | NOCATALOGCONNECT]
[CONNECTIONPORT <port>]
[DECRYPTPASSWORD <shared secret> ENCRYPTKEY {DEFAULT | <key name>}]
[DEFERREFCONST]
[DISABLECOMMITNOWAIT]
[DISABLELOBCACHING]
[EMPTYLOBSTRING ‘<string>’]
[FETCHBATCHSIZE <num_recs>]
[FETCHLOBS | NOFETCHLOBS]
[HOST <host ID>]
[LIMITROWS | NOLIMITROWS]
[LOBBUFSIZE]
[LOBWRITESIZE <size>]
[SHOWINFOMESSAGES]
[SHOWWARNINGS]
[SPTHREAD | NOSPTHREAD]
[SUPPRESSTRIGGERS | NOSUPPRESSTRIGGERS]
[TDSPACKETSIZE <bytes>]
[TRUSTEDCONNECTION]
[USEODBC | USEREPLICATIONUSER]
[XMLBUFSIZE <buffer size>]
```

Argument	Description
ALLOWUNUSEDCOLUMN	Valid for Extract for Oracle. Prevents Extract from abending when it encounters a table with an unused column. Instead, it continues processing and generates a warning. When using this parameter, the same unused column must exist on the target, or else a source definitions file for the table must be specified to Replicat so that the correct metadata mapping can be performed. By default, Extract abends on unused columns.

Argument	Description
`ALLOWLOBDATATRUNCATE | NOALLOWLOBDATATRUNCATE`	Valid for Replicat for Sybase and MySQL. ALLOWLOBDATATRUNCATE prevents Replicat from abending when replicated LOB data is too large for a target CHAR, VARCHAR, BINARY or VARBINARY column. The LOB data is truncated to the maximum size of the target column *without any further error messages or warnings*. NOALLOWLOBDATATRUNCATE is the default and causes Replicat to abend with an error message if the replicated LOB is too large.
`CATALOGCONNECT | NOCATALOGCONNECT`	Valid for Extract and Replicat for ODBC databases. By default, Oracle GoldenGate creates a new connection for catalog queries, but you can use NOCATALOGCONNECT to prevent that. On DB2 for z/OS, NOCATALOGCONNECT prevents Oracle GoldenGate from attempting multiple connections when the MVS DB2 initialization parameter mvsattachtype is set to CAF. Because CAF mode does not support multiple connections, it is possible that Oracle GoldenGate may issue commit locks on the system catalog tablespaces until it receives the commit for its open connection. To prevent commit locks, Oracle GoldenGate recommends using RRSAF (mvsattachtype=RRSAF), which supports multiple connections.
`CONNECTIONPORT <port>`	Valid for Replicat for multi-daemon MySQL. Specifies the TCP/IP port of the instance to which Replicat must connect.
`DECRYPTPASSWORD <shared secret> <algorithm> ENCRYPTKEY {<keyname> | DEFAULT}`	Valid for Extract (Oracle) Specifies the *shared secret* (password) that decrypts the TDE key, which decrypts redo log data that was encrypted with Oracle Transparent Data Encryption (TDE). The TDE key is first encypted in the Oracle server by using the shared secret as a key, and then it is delivered to Extract, which decrypts it by using the same shared secret. The shared secret must be created in the Oracle Wallet or Hardware Security Module by the Oracle Server Security Officer. The only other person who should know the shared secret is the Oracle GoldenGate Administrator. To use the decryption options, you must first generate the encrypted shared secret with the ENCRYPT PASSWORD command in GGSCI and create an ENCKEYS file.

Argument	Description
	Parameter options: ◆ <shared secret> is the encrypted shared secret (password) that is copied from the ENCRYPT PASSWORD command results. ◆ <algorithm> specifies the encryption algorithm that was used to encrypt the password: AES128, AES192, AES256, or BLOWFISH. ◆ ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option. Requires an ENCKEYS file to be created on the local system. ◆ ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option. For more information about configuring Extract to support TDE, see the Oracle GoldenGate *Oracle Installation and Setup Guide*. For more information about Oracle GoldenGate encryption options, including ENCKEYS, see the security chapter of the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.
DEFERREFCONST	Valid for Replicat for Oracle. Sets constraints to DEFERRABLE to delay the checking and enforcement of cascade delete and cascade update referential integrity constraints by the Oracle target database until the Replicat transaction is committed. At that point, if there are constraint violations, an error is generated. You can use DEFERREFCONST instead of disabling the constraints on the target tables or setting them to DEFERRED. When used, DEFERREFCONST defers both DEFERABLE and NOT DEFERABLE constraints. DEFERREFCONST applies to every transaction that is processed by Replicat. If used with an Oracle version that does not support this functionality, DEFERREFCONST is ignored without returning a notification to the GoldenGate log. To handle errors on the commit operation, you can use REPERROR at the root level of the parameter file and specify the TRANSDISCARD or TRANSEXCEPTION option.
DISABLECOMMITNOWAIT	Valid for Replicat for Oracle. Disables the use of asynchronous COMMIT by Replicat. An asynchronous COMMIT statement includes the NOWAIT option. When DISABLECOMMITNOWAIT is used, Replicat issues a standard synchronous COMMIT (COMMIT with WAIT option).

Argument	Description
`DISABLELOBCACHING`	Valid for Replicat for Oracle. Disables Oracle's LOB caching mechanism. By default, Replicat enables Oracle's LOB caching mechanism.
`EMPTYLOBSTRING '<string>'`	Valid for Replicat. Substitutes a string value for empty (zero-length) LOB columns, such as Sybase IMAGE or TEXT values, that are replicated to the target. By default, Oracle GoldenGate sets empty columns to NULL on the target and will abend if the target database does not permit LOB columns to be NULL. This option prevents Replicat from abending. For '<string>' use any string that the column accepts, and enclose the string within single quotes. The default is NULL. Example: `DBOPTIONS EMPTYLOBSTRING 'empty'`
`FETCHBATCHSIZE <num_recs>`	Valid for Extract for Oracle, DB2, SQL/MX, Sybase, SQL Server, Sybase, and Teradata. Enables array fetches for initial loads to improve performance, rather than one row at a time. Valid values are 0 through 1000000 records per fetch. The default is 1000. Performance slows when batch size gets very small or very large. If the table contains LOB data, Extract reverts to single-row fetch mode, and then resumes batch fetch mode afterward.
`HOST <host id>`	Valid for Replicat for multi-daemon MySQL. Specifies the host's DNS name or IP address of the instance to which Replicat must connect.
`FETCHLOBS | NOFETCHLOBS`	Valid for Extract for DB2 for z/OS and DB2 for LUW. Suppresses the fetching of LOBs directly from the database table when the LOB options for the table are set to NOT LOGGED. With NOT LOGGED, the value for the column is not available in the transaction logs and can only be obtained from the table itself. By default, Oracle GoldenGate captures changes to LOBs from the transaction logs. The default is FETCHLOBS.

Argument	Description

`LIMITROWS | NOLIMITROWS`	Valid for Replicat for MySQL, Oracle, SQL Server, and Sybase. LIMITROWS prevents multiple rows from being updated or deleted by the same Replicat SQL statement when the target table does not have a primary or unique key.
	For MySQL, it uses a LIMIT 1 clause in the UPDATE or DELETE statement.
	For Oracle, it alters the WHERE clause used by Replicat by adding either of the following clauses, depending on whether or not a WHERE clause already exists:
	`WHERE ROWNUM = 1`
	or...
	`AND ROWNUM = 1`
	For SQL Server and Sybase, it uses a SET ROWCOUNT 1 clause before the UPDATE or DELETE statement.
	NOLIMITROWS permits multiple rows to be updated or deleted by the same Replicat SQL statement. This option does not work when using Oracle's OCI.
	LIMITROWS is the default. LIMITROWS and NOLIMITROWS apply globally to all MAP statements in a parameter file.

`LOBBUFSIZE <bytes>`	Valid for Extract for Oracle. Determines the memory buffer size to allocate for each embedded LOB attribute that is in an Oracle object type. Valid values are from 1024 and 10485760 bytes. The default is 1048576 bytes. If the length of embedded LOB exceeds the specified LOBBUFSIZE size, an error message similar to the following is generated: `GGS ERROR   ZZ-0L3  Buffer overflow, needed: 2048, allocated: 1024.`

Argument	Description
`LOBWRITESIZE <size>`	Valid for Replicat for Oracle. Specifies a fragment size for each LOB that Replicat writes to the target database. The LOB data is stored in a buffer until this size is reached. Because LOBs must be written to the database in fragments, writing in larger blocks prevents excessive I/O. The higher the value, the fewer I/O calls that are made by Replicat to the database server to write the whole LOB to the database. Specify a multiple of the Oracle LOB fragment size. A given value will be rounded up to a multiple of the Oracle LOB fragment size, if necessary. The default LOB write size is 32k. Valid values are from 2,048 bytes to 1,048,576 bytes (1MB). By default, Replicat enables Oracle's LOB caching mechanism. To disable Oracle's LOB caching, use the DISABLELOBCACHING option of DBOPTIONS.

`SHOWINFOMESSAGES`	Valid for Extract and Replicat for Sybase. Enables the following Sybase server messages to be printed to the error log.
	`0: /* General informational message */`
	`5701: /* Changed Database Context */`
	`5703: /* Changed language setting */`
	`5704: /* Changed client character set */`
	`7326: /* Non ANSI Escaping */`
	Normally, these messages are suppressed because they do not affect Oracle GoldenGate processing.

`SHOWWARNINGS`	Valid for Extract and Replicat for Sybase. Enables the logging of Sybase server messages with a severity level greater than 10. These messages may be useful for debugging when Sybase performs corrective action that causes data to change.
`SPTHREAD | NOSPTHREAD`	Valid for Extract and Replicat. Creates a separate database connection thread for stored procedures. The default is NOSPTHREAD.

Argument	Description
SUPPRESSTRIGGERS | NOSUPPRESSTRIGGERS	Valid for Replicat for Oracle. Controls whether or not triggers are fired during the Replicat session. SUPPRESSTRIGGERS prevents triggers from firing on target objects that are configured for replication with Oracle GoldenGate. Instead of manually disabling the triggers, you can use SUPPRESSTRIGGERS for these Oracle versions: ◆ Oracle 10.2.0.5 and later patches to 10.2.0.5 ◆ Oracle 11.2.0.2 and later 11*g*R2 versions SUPPRESSTRIGGERS is not available for 11*g*R1. To use SUPPRESSTRIGGERS, the Replicat user must be an Oracle Streams administrator, which can be granted by invoking the following: For Oracle 10.2.0.5 and patches, use: dbms_streams_auth.grant_admin_privilege For Oracle 11*g*R2 versions, use: dbms_goldengate_auth.grant_admin_privilege These procedures are part of the Oracle database installation. See the database documentation for more information. NOSUPPRESSTRIGGERS is the default and allows target triggers to fire. The SOURCEDB and USERID parameters must precede a DBOPTIONS statement that contains SUPPRESSTRIGGERS.

Argument	Description
`TDSPACKETSIZE <bytes>`	Valid for Extract and Replicat for Sybase. Sets the TDS packet size for replication to a Sybase target. Valid values: ◆ Sybase version 12.5.4 (Note: this version is de-supported as of Oracle GoldenGate 11.2.1): 512 to 65024 Default is 0 for Extract, 512 for Replicat ◆ Sybase15 or higher: 2048 to 65024 Default is 0 for Extract, 2048 for Replicat The value must be a mutiple of 512. The range of values that are set for the Sybase Adaptive Server max network packet size and additional network memory parameters must support the value that is set with TDSPACKETSIZE. Note: The higher the max network packet size value, the more memory (as set with additional network memory) the database server needs to allocate for the network data. For best performance, choose a server packet size that works efficiently with the underlying packet size on your network. The goals of this procedure are to: ◆ Reduce the number of server reads and writes to the network. ◆ Reduce unused space in the network packets to increase network throughput. For example, if your network packet size carries 1500 bytes of data, you can achieve better transfer performance by setting the packet size on the server to 1024 (512*2) than by setting it to 1536 (512*3). For optimal performance, start with the following configuration: DBOPTIONS TDSPACKETSIZE 8192 The DBOPTIONS parameter that contains TDSPACKETSIZE must be placed before the SOURCEDB or TARGETDB parameter in the parameter file.
`TRUSTEDCONNECTION`	Valid for Extract and Replicat for SQL Server. Causes Oracle GoldenGate to connect by using trusted connection = yes. Contact Oracle Support before using this option. For more information, go to http://support.oracle.com.

Argument	Description
`USEODBC`	Valid for Replicat for SQL Server. Configures Replicat to use ODBC to perform DML operations. The default is to use OLE DB. Not valid if USEREPLICATIONUSER is enabled; will cause Replicat to abend. **Note**: Replicat always uses ODBC to connect to the database catalog to obtain metadata.
`USEREPLICATIONUSER`	Valid for Replicat for SQL Server. Configures Replicat to perform target DML operations as the SQL Server replication user. The replication user is not a SQL Server user or account, but is a property of the database connection. USEREPLICATIONUSER enables the SQL Server NOT FOR REPLICATION flag. When the replication user is used, the following concerns must be addressed for their effect on data integrity: ◆ IDENTITY seeds on the target are not updated. A partitioning scheme is needed to avoid primary key violations unless the target is read-only. ◆ Foreign key constraints are not enforced. ◆ ON UPDATE CASCADE, ON DELETE CASCADE and triggers are disabled. This is beneficial to Replicat, since it prevents duplicate operations, but may not be appropriate for the target applications and might require modification to the code of the constraint or trigger to ensure data integrity. ◆ CHECK constraints are not enforced, so data integrity cannot be certain on the target. For more information about these considerations, see the Oracle GoldenGate *SQL Server Installation and Setup Guide*. By default, USEREPLICATIONUSER is disabled and the default is to use OLE DB. The use of USEREPLICATIONUSER is only advised if delivery performance must be increased. Not valid if USEODBC is enabled; will cause Replicat to abend.
`XMLBUFSIZE <bytes>`	Valid for Extract for Oracle. Sets the size of the memory buffer that stores XML data that was extracted from the sys.xmltype attribute of a SDO_GEORASTER object type. The default is 1048576 bytes (1MB). If the data exceeds the default buffer size, Extract will abend. If this occurs, increase the buffer size and start Extract again. The valid range of values is 1024 to 10485760 bytes.

Example 1 `DBOPTIONS HOST 127.0.0.1, CONNECTIONPORT 3307`

Example 2 `DBOPTIONS DECRYPTPASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH ENCRYPTKEY DEFAULT`

Example 3 `DBOPTIONS TDSPACKETSIZE 2048`

Example 4 `DBOPTIONS FETCHBATCHSIZE 2000`

Example 5 `DBOOPTION XMLBUFSIZE 2097152`

DDL

Valid for Extract and Replicat

Use the DDL parameter to:

- enable DDL support
- filter DDL operations
- configure a processing action based on a DDL record

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.

Valid for Extract and Replicat

Do not use DDL for:

- an Extract data pump
- a VAM-sort Extract (Teradata source databases)

These process types do not permit mapping or conversion of DDL and will propogate DDL records automatically in PASSTHRU mode (see page 287). DDL that is performed on a source table of a certain name (for example ALTER TABLE TableA...) will be applied by Replicat with the same table name (ALTER TABLE TableA). It cannot be mapped as ALTER TABLE TableB.

For detailed information about how to use Oracle GoldenGate DDL support, see the Oracle GoldenGate documentation for Oracle or Teradata, as applicable.

Syntax

```
DDL [
{INCLUDE | EXCLUDE}
   [, MAPPED | UNMAPPED | OTHER | ALL]
   [, OPTYPE <type>]
   [, OBJTYPE ‘<type>’]
   [, OBJNAME <name>]
   [, INSTR ‘<string>’]
   [, INSTRCOMMENTS ‘<comment_string>’]
   [, STAYMETADATA]
   [, EVENTACTIONS (<action specification>)
]
[...]
```

Table 35 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 OBJNAME “fin.ss”`
	An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.

Table 35 DDL inclusion and exclusion options (continued)

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. 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 35 DDL inclusion and exclusion options (continued)

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 derived objects, such as a trigger, the value for OBJNAME must be the name of the base object, not the name of the derived object. 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>’	(Valid for Oracle) 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 35 DDL inclusion and exclusion options (continued)

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>’`	(Valid for Oracle) 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.
`STAYMETADATA`	(Valid for Oracle). Prevents metadata from being captured by Extract or applied by Replicat. When Extract first encounters DML on a table, it retrieves the metadata for that table. When DDL is encountered on that table, the old metadata is invalidated. The next DML on that table is matched to the new metadata so that the target table structure always is up-to-date with that of the source. However, if you know that a particular DDL operation will not affect the table's metadata, you can use STAYMETADATA so that the current metadata is not retrieved or replicated. This is a performance improvement that has benefit for such operations as imports and exports, where such DDL as truncates and the disabling of constraints are often performed. These operations do not affect table structure, as it relates to the integrity of subsequent data replication, so they can be ignored in such cases. For example ALTER TABLE ADD FOREIGN KEY does not affect table metadata.

Table 35 DDL inclusion and exclusion options (continued)

Option	Description
	An example of how this can be applied selectively is as follows: `DDL INCLUDE ALL INCLUDE STAYMETADATA OBJNAME xyz` This example states that all DDL is to be included for replication, but only DDL that operates on object “xyz” will be subject to STAYMETADATA. STAYMETADATA also can be used the same way in an EXCLUDE clause. STAYMETADATA must be used the same way on the source and target to ensure metadata integrity. When STAYMETADATA is in use, a message is added to the report file. DDL reporting is controlled by the DDLOPTIONS parameter with the REPORT option. This same functionality can be applied globally to all DDL that occurs on the source by using the @ddl_staymetadata scripts: ◆ @ddl_staymetadata_on globally turns off metadata versioning. ◆ @ddl_staymetadata_off globally enables metadata versioning again. This option should be used with the assistance of Oracle GoldenGate technical support staff, because it might not always be apparent which DDL affects object metadata. If improperly used, it can break the integrity of the replication environment.
`EVENTACTIONS (<action specification>)`	Causes the Extract or Replicat process take a defined action based on a DDL record in the transaction log or trail, which is known as the *event record*. The DDL event is triggered if the DDL record is eligible to be written to the trail by Extract or a data pump, or to be executed by Replicat, as determined by the other filtering options of the DDL parameter. You can use this system to customize processing based on database events. For <action specification> see EVENTACTIONS under the MAP and TABLE parameters. Guidelines for using EVENTACTIONS on DDL records: ◆ CHECKPOINTBEFORE: Since each DDL record is autonomous, the DDL record is guaranteed to be the start of a transaction; therefore, the CHECKPOINT BEFORE event action is implied for a DDL record. ◆ IGNORE: This option is not valid for DDL records. Because DDL operations are autonomous, ignoring a record is equivalent to ignoring the entire transaction. EVENTACTIONS does not support the following DDL objects because they are derived objects: ◆ indexes ◆ triggers ◆ synonyms ◆ RENAME on a table and ALTER TABLE RENAME

Example 1 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.”

Example 2 The following example specifies an event action of REPORT for all DDL records.

```
DDL INCLUDE ALL EVENTACTIONS (REPORT)
```

Example 3 The following example shows how EVENTACTIONS can be used on a subset of the DDL. All DDL is to be replicated, but only the DDL that is executed on explicitly named objects qualifies to trigger the event actions of REPORT and LOG.

```
DDL INCLUDE ALL &
    INCLUDE OBJNAME sales.t* EVENTACTIONS (REPORT)
    INCLUDE OBJNAME fin.my_tab EVENTACTIONS (LOG)
```

Example 4 The following example shows the effect when the GLOBALS parameter USEANSISQLQUOTES is used. The string parameter that is passed to the Oracle GoldenGate function is enclosed within single quotes (instead of the default of double quotes). This example also shows compound functions that use both DDL functions and string functions.

```
DDL INCLUDE OBJNAME src.t* &
EVENTACTIONS (SHELL (“echo shell("echo extract $0 DDL &
optype-objtype-owner.name: $1", &
var $0=@GETENV('GGENVIRONMENT','GROUPNAME'), &
var $1 = @strcat(@ddl(optype), '-', @ddl(objname), '-', &
@ddl(objowner), '-', @ddl(objname))))
```

The result of this shell command for an “alter table t2 modify (col2 date)” might look something like this:

```
“extract E_CUST DDL optype-objtype-owner.name: ALTER-T2-SRC-T2”
```

DDLERROR

Valid for Extract and Replicat

Use the DDLERROR parameter to handle DDL errors on the source and target systems. Options are available for Extract and Replicat.

Extract DDLERROR options

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

Default Abend

Syntax

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

Argument	Description
`RESTARTSKIP <num skips>`	Causes Extract to skip and ignore a specific number of DDL operations on startup, to prevent Extract from abending on an error. By default, a DDL error causes Extract to abend so that no operations are skipped. Valid values for this parameter are 1 to 100000. To write information about skipped operations to the Extract report file, use the DDLOPTIONS parameter with the REPORT option.
`SKIPTRIGGERERROR <num errors>`	(Oracle) Causes Extract to skip and ignore a specific number of DDL errors that are caused by the DDL trigger on startup. Valid values for <num errors> are 1 through 100000. SKIPTRIGGERERROR is checked before the RESTARTSKIP option. If Extract skips a DDL operation because of a trigger error, that operation is not counted toward the RESTARTSKIP specification.

Replicat DDLERROR options

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.

Default Abend

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>}`	Identifies the beginning of an inclusion or exclusion clause that contols whether specific DDL is handled or not handled by the DDLERROR statement. For syntax and usage, see "DDL inclusion and exclusion options" on page 165.
`[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.

Example 1 DDLERROR basic example

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.

Example 2 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.

DDLOPTIONS

Valid for Extract and Replicat

Use the DDLOPTIONS parameter to configure aspects of DDL processing other than filtering and string substitution. You can use multiple DDLOPTIONS statements, but using one is recommended. If using multiple DDLOPTIONS statements, make each of them unique so that one does not override the other. Multiple DDLOPTIONS statements are executed in the order listed.

For more information about how to use DDLOPTIONS within the context of Oracle GoldenGate DDL support, see the Oracle GoldenGate *Windows and UNIX Administrator’s Guide.*

Default See the argument descriptions

Syntax

```
DDLOPTIONS
[, ADDTRANDATA [ABEND | RETRYOP <RETRYDELAY <seconds> MAXRETRIES <retries>]
[, DEFAULTUSERPASSWORDPASSWORD <password>
   [ENCRYPTKEY DEFAULT | ENCRYPTKEY <keyname>]]
[, GETAPPLOPS | IGNOREAPPLOPS]
[, GETREPLICATES | IGNOREREPLICATES]
[, IGNOREMAPPING]
[, MAPDERIVED | NOMAPDERIVED]
[, MAPSCHEMAS]
[, MAPSESSIONSCHEMA] <source_schema> TARGET <target_schema>
[, PASSWORD ENCRYPTKEY [DEFAULT | ENCRYPTKEY <keyname>]
[, REMOVECOMMENTS {BEFORE | AFTER}]
[, REPLICATEPASSWORD | NOREPLICATEPASSWORD]
[, REPORT | NOREPORT]
[, UPDATEMETADATA]
[, USEOWNERFORSESSION]
```

Argument	Description
`ADDTRANDATA [ABEND | RETRYOP <RETRYDELAY <seconds> MAXRETRIES <retries>]`	Valid for Extract (Oracle and Teradata) Use ADDTRANDATA to: ◆ enable Oracle supplemental logging automatically for new tables created with a CREATE TABLE. ◆ to update supplemental logging for tables affected by an ALTER TABLE to add or drop columns. ◆ update supplemental logging for tables that are renamed. ◆ update supplemental logging for tables where unique or primary keys are added or dropped. ABEND | RETRYOP <RETRYDELAY <seconds> MAXRETRIES <retries> Controls whether Extract abends or tries again when the ADD TRANDATA command fails because the table is locked. ◆ ABEND causes Extract to abend. ◆ RETRYOP causes Extract to try again based on RETRYDELAY and MAXRETRIES. Both are required when RETRYOP is used. ◆ RETRYDELAY sets the delay before Extract tries again. The maximum delay is 10,000 seconds. ◆ MAXRETRIES sets the number of retries that Extract can make before abending. The maximum is 10,000 retries. The default is: `DDLOPTIONS ADDTRANDATA RETRYOP RETRYDELAY 10 MAXRETRIES 10`

Argument	Description

	To use this functionality, Oracle GoldenGate, the database, and the appropriate tables must be configured for DDL capture, according to the instructions in the Oracle GoldenGate *Windows and UNIX Reference Guide*. For Oracle, the Oracle GoldenGate DDL objects must be installed and configured according to the instructions in the Oracle GoldenGate *Oracle Installation and Setup Guide*.
	For new tables created with CREATE TABLE, ADDTRANDATA produces the same results as the default ADD TRANDATA command in GGSCI by issuing the Oracle ALTER TABLE command with the ADD SUPPLEMENTAL LOG GROUP option. Oracle GoldenGate executes this command when the CREATE TABLE or ALTER TABLE is captured on the source. If you have special requirements for the supplemental logging, use the ADD TRANDATA command, not DDLOPTIONS ADDTRANDATA.
	By default, the ALTER TABLE that adds the supplemental logging is not replicated to the target unless the GETREPLICATES option is in use.
	For renamed tables, ADDTRANDATA deletes the supplemental log group for the old table and creates it for the new one. If you do not use ADDTRANDATA and tables will be renamed, do the following to create the log group before doing the rename:
	1. Drop the supplemental log group using the database interface or DELETE TRANDATA <table> in GGSCI.
	2. Rename the table.
	3. Create the new supplemental log group using the database interface or ADD TRANDATA <table> in GGSCI.
	There might be a lag between the time when an original DDL operation occurs and when the ADD TRANDATA takes effect. During this time, do not allow DML operations (insert, update, delete) on the affected table if the data is to be replicated; otherwise, it will not be captured. To determine when DML can be resumed after ADDTRANDATA:
	1. Edit the Extract parameter file in GGSCI.
	Warning: Do not use the VIEW PARAMS or EDIT PARAMS command to view or edit a parameter file that was created in a character set other than that of the local operating system. View the file from outside GGSCI; otherwise, the contents may become corrupted.
	2. Add the REPORT option to DDLOPTIONS, then save and close the file.
	`DDLOPTIONS [, other DDLOPTIONS options], REPORT`
	3. Stop and start Extract to activate the parameter changes.
	`STOP EXTRACT <group>`
	`START EXTRACT <group>`

Argument	Description

	4. View the Extract process report.
	`VIEW REPORT <group name>`
	5. Look for the ALTER TABLE that added the log group to the table, and make a note of the time that the command took effect. The entry looks similar to the following:
	`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 */ ].`
	6. Permit DML operations on the new table.
	See also ADD SCHEMATRANDATA on page 84.

`DEFAULTUSERPASSWORD <password> [<algorithm> ENCRYPTKEY {<keyname> | DEFAULT}]`	Valid for Replicat. (Oracle only) Specifies a different password for a replicated {CREATE | ALTER} USER <name> IDENTIFIED BY <password> statement from the one used in the source statement. Replicat will replace the placeholder that Extract writes to the trail with the specified password. When using DEFAULTUSERPASSWORD, use the NOREPLICATEPASSWORD option of DDLOPTIONS for Extract. DEFAULTUSERPASSWORD <password> without options specifies a clear-text password. If the password is case-sensitive, type it that way. Use the following options if the password was encrypted with the ENCRYPT PASSWORD command in GGSCI: ◆ <algorithm> specifies the encryption algorithm that was used to encrypt the password with ENCRYPT PASSWORD: AES128, AES192, AES256, or BLOWFISH. ◆ ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option, and specify the same key name. ◆ ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option. For more information about Oracle GoldenGate encryption options, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Argument	Description
`GETAPPLOPS | IGNOREAPPLOPS`	Valid for Extract. (Oracle only) Controls whether or not DDL operations produced by business applications *except* Replicat are included in the content that Extract writes to a trail or file. GETAPPLOPS and IGNOREAPPLOPS can be used together with the GETREPLICATES and IGNOREREPLICATES options to control which DDL is propagated in a bidirectional or cascading configuration. ◆ For a bidirectional configuration, use GETAPPLOPS with GETREPLICATES. You also must use the UPDATEMETADATA option. ◆ For a cascading configuration, use IGNOREAPPLOPS with GETREPLICATES on the systems that will be cascading the DDL operations to the target. The default is GETAPPLOPS.

`GETREPLICATES |`
`IGNOREREPLICATES`
	Valid for Extract (Oracle only). Controls whether or not DDL operations produced by Replicat are included in the content that Extract writes to a trail or file. The default is IGNOREREPLICATES. For more information, see the GETAPPLOPS | IGNOREAPPLOPS options of DDLOPTIONS.

`IGNOREMAPPING`	Valid for Replicat. Disables the evaluation of name mapping that determines whether DDL is of MAPPED or UNMAPPED scope. This option improves performance in like-to-like DDL replication configurations, where source and target schema names and object names match, and where mapping functions are therefore unnecessary. With IGNOREMAPPING enabled, MAPPED or UNMAPPED scope cannot be determined, so all DDL statements are treated as OTHER scope. Do not use this parameter when source schemas and object names are mapped to different schema and object names on the target.
`MAPDERIVED | NOMAPDERIVED`	Valid for Replicat (Oracle and Teradata). Controls how derived object names are mapped. ◆ MAPDERIVED: If a MAP statement exists for the derived object, the name is mapped to the name specified in that TARGET clause. Otherwise, the name is mapped to the name specified in the TARGET clause of the MAP statement that contains the base object. MAPDERIVED is the default. ◆ NOMAPDERIVED: Prevents name mapping. NOMAPDERIVED overrides any explicit MAP statements that contain the name of the derived object. For more information about how derived objects are handled during DDL replication, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Argument	Description

MAPSCHEMAS	Valid for Replicat (Oracle and Teradata). Use only when MAPSESSIONSCHEMA is used.
	◆ MAPSESSIONSCHEMA establishes a source-target mapping for session schemas and is used for objects whose schemas are not qualified in the DDL.
	◆ MAPSCHEMAS maps objects that do have qualified schemas in the source DDL, but which do not qualify for mapping with MAP, to the same session-schema mapping as in MAPSESSIONSCHEMA. Examples of such objects are the Oracle CREATE TABLE AS SELECT statement, which contains a derived object in the AS SELECT clause, or the Teradata CREATE REPLICATION RULESET statement.
	This mapping takes place after the mapping that is specified in the MAP statement.
	As an example, suppose the following DDL statement is issued on a source Oracle database:
	`create table a.t as select from b.t;`
	Suppose the MAP statement on the target is as follows:
	`MAP a.*, TARGET c.*;`
	`DDLOPTIONS MAPSESSIONSCHEMA b, TARGET b1, MAPSCHEMAS`
	As a result of this mapping, Replicat issues the following DDL statement on the target:
	`create table c.t as select from b1.t;`
	◆ The base table gets mapped according to the TARGET clause (to schema “c”).
	◆ The qualified derived object (table “t” in “SELECT FROM”) gets mapped according to MAPSESSIONSCHEMA (to schema “b1”) because MAPSCHEMAS is present.
	Without MAPSCHEMAS, the derived object would get mapped to schema “c” (as specified in the TARGET clause), because MAPSESSIONSCHEMA alone only maps unqualified objects.

MAPSESSIONSCHEMA <source_schema> TARGET <target_schema>	Valid for Replicat (Oracle only). Enables a source session schema to be mapped to (transformed to) a different session schema on the target. ◆ <source_schema> is the session schema that is set with ALTER SESSION set CURRENT_SCHEMA on the source. ◆ <target_schema> is the session schema that is set with ALTER SESSION set CURRENT_SCHEMA on the target. Wildcards are not supported. You can use multiple MAPSESSIONSCHEMA parameters to map different schemas. MAPSESSIONSCHEMA overrides any mapping of schema names that is based on master or derived object names See the example at the end of this topic for usage. See also MAPSCHEMAS.

Argument	Description
`PASSWORD <algorithm> ENCRYPTKEY {<keyname> | DEFAULT}`	Valid for Extract (Oracle only) Directs Extract to encrypt all passwords in source DDL before writing the DDL to the trail. ◆ <algorithm> specifies the encryption algorithm to be used to encrypt the password: AES128, AES192, AES256, or BLOWFISH. ◆ ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in an ENCKEYS lookup file. ◆ ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key.
`REMOVECOMMENTS {BEFORE | AFTER}`	Valid for Extract and Replicat (Oracle only). Controls whether or not comments are removed from the DDL operation. By default, comments are not removed, so that they can be used for string substitution with the DDLSUBST parameter (see page 181). ◆ REMOVECOMMENTS BEFORE removes comments before the DDL operation is processed by Extract or Replicat. They will not be available for string substitution. ◆ REMOVECOMMENTS AFTER removes comments after they are used for string substitution.
`REPLICATEPASSWORD | NOREPLICATEPASSWORD`	Valid for Extract (Oracle only). Applies to the password in a {CREATE | ALTER} USER <user> IDENTIFIED BY <password> command. ◆ By default (REPLICATEPASSWORD), Oracle GoldenGate uses the source password in the target CREATE or ALTER statement. ◆ To prevent the source password from being sent to the target, use NOREPLICATEPASSWORD. When using NOREPLICATEPASSWORD, specify a password for the target DDL statement by using a DDLOPTIONS statement with the DEFAULTUSERPASSWORD option in the Replicat parameter file.
`REPORT | NOREPORT`	Valid for Extract and Replicat (Oracle and Teradata). Controls whether or not expanded DDL processing information is written to the report file. The default of NOREPORT reports basic DDL statistics. REPORT adds the parameters being used and a step-by-step history of the operations that were processed.

Argument	Description
UPDATEMETADATA	Valid for Replicat (Oracle only). Use in an active-active bi-directional configuration. This parameter notifies Replicat on the system where DDL originated that this DDL was propagated to the other system, and that Replicat should now update its object metadata cache to match the new metadata. This keeps Replicat's metadata cache synchronized with the current metadata of the local database.
USEOWNERFORSESSION	Valid for Replicat (Oracle only). Forces the schema of an unqualified object in the Replicat DDL statement to be that of the Replicat session schema, instead of the schema in an ALTER SESSION SET CURRENT_SCHEMA statement, which is the default behavior.

Example 1 The following shows how MAPSESSIONSCHEMA works to allow mapping of a source session schema to another schema on the target.

Assume the following DDL capture and mapping configurations in Extract and Replicat:

Extract

```
DDL INCLUDE OBJNAME SRC.* &
INCLUDE OBJNAME SRC1.*
TABLE SRC.*;
TABLE SRC1.*;
```

Replicat

```
DDLOPTIONS MAPSESSIONSCHEMA SRC TARGET DST
DDLOPTIONS MAPSESSIONSCHEMA SRC1 TARGET DST1
MAP SRC.*, TARGET DST.*;
MAP SRC1.*, TARGET DST1.*;
DDL INCLUDE OBJNAME DST.* &
INCLUDE OBJNAME DST1.*
```

Assume that the following DDL statements are issued by the logged-in user OTH on the source:

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

Replicat will perform the DDL as follows (explanations precede each code segment):

```
-- Set session to DST, because SRC.* is mapped to DST.* in MAP statement.
ALTER SESION SET CURRENT_SCHEMA=DST;
-- Create the first TAB table in the DST schema, using the DST session schema.
CREATE TABLE DST.tab (X NUMBER);
-- Restore Replicat schema.
ALTER SESSION SET CURRENT_SCHEMA=REPUSER
-- Set session schema to DST, per MAPSESSIONSCHEMA, so that AS SELECT succeeds.
ALTER SESION SET CURRENT_SCHEMA=DST;
-- Create the DST1.TAB table AS SELECT * FROM the first table (DST.TAB).
CREATE TABLE DST1.tab (X NUMBER) AS SELECT * FROM tab;
-- Restore Replicat schema.
ALTER SESSION SET CURRENT_SCHEMA=REPUSER
```

Without MAPSESSIONSCHEMA, the SELECT * FROM TAB would attempt to select from a non-existent SRC.TAB table and fail. The default is to apply the source schema to unqualified objects in a target DDL statement. The DDL statement in that case would look as follows and would fail:

```
-- Set session to DST, because SRC.* is mapped to DST.* in MAP statement.
ALTER SESION SET CURRENT_SCHEMA=DST;
-- Create the first TAB table in the DST schema, using the DST session schema.
CREATE TABLE DST.tab (X NUMBER);
-- Restore Replicat schema.
ALTER SESSION SET CURRENT_SCHEMA=REPUSER
-- Set session schema to SRC, because TAB in the AS SELECT is unqualified and
SRC is the source session schema.
ALTER SESION SET CURRENT_SCHEMA=SRC;
-- Create DST1.TAB AS SELECT * from SRC.TAB (SRC=current session schema).
CREATE TABLE DST1.tab (X NUMBER) AS SELECT * FROM tab;
-- SRC.TAB does not exist.
-- Abend with an error unless the error is handled by a DDLERROR statement.
```

Example 2 The following shows different ways to use DEFAULTUSERPASSWORD to specify a different password for a replicated {CREATE | ALTER} USER <name> IDENTIFIED BY <password> statement from the one used in the source statement.

```
DDLOPTIONS DEFAULTUSERPASSWORD ocean

DDLOPTIONS DEFAULTUSERPASSWORD &
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
AES 256 ENCRYPTKEY mykey

DDLOPTIONS DEFAULTUSERPASSWORD &
AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
BLOWFISH ENCRYPTKEY DEFAULT
```

DDLSUBST

Valid for Extract and Replicat

Use the DDLSUBST parameter to substitute strings in a DDL operation. For example, you could substitute one table name for another or substitute a string within comments. The search is not case-sensitive. To represent a quotation mark in a string, use a double quote mark.

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.

Default No substitution

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>`
	Specifies one or more INCLUDE and EXCLUDE statements to filter the DDL operations for which the string substitution rules are applied. For syntax and usage, see “DDL inclusion and exclusion options” on page 165.

Example 1 The following replaces the string ‘cust’ with the string ‘customers’ for tables in the “fin” schema.

```
DDLSUBST ‘cust’ WITH ‘customers’
INCLUDE ALL OBJTYPE ‘table’ OBJNAME fin.*
```

Example The following substitutes a new directory 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 2 The following uses multiple DDLSUBST statements, which execute in the order shown.

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

The net effect of the preceding substitutes all “a” and “b” strings with “c.”

DDLTABLE

Valid for GLOBALS

Use the DDLTABLE parameter to specify the name of the DDL history table that supports Oracle DDL synchronization, if other than the default of GGS_DDL_HIST. The DDL history table stores a history of DDL operations processed by Oracle GoldenGate.

The name of the history table must also be specified with the ddl_hist_table parameter in the params.sql script. This script resides in the root Oracle GoldenGate installation directory.

This parameter is valid only for Oracle. For more information about the DDL history table and params.sql, see Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Default GGS_DDL_HIST

Syntax `DDLTABLE <table_name>`

Argument	Description
`<table_name>`	The name of the DDL history table.

Example `DDLTABLE GG_DDL_HISTORY`

DECRYPTTRAIL

Valid for Extract and Replicat

Use the DECRYPTTRAIL parameter to decrypt data in a trail or extract file when ENCRYPTTRAIL is used for the associated Extract process to encrypt the trail or file.

Default None

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

Argument	Description
`DECRYPTTRAIL`	If the trail or file was encrypted with ENCRYPTTRAIL without options (256-key byte substitution), use DECRYPTTRAIL without options.

Argument	Description

`{AES128 | AES192 | AES256}`
`KEYNAME <keyname>`
	If an AES cipher was used in ENCRYPTTRAIL, specify the matching cipher. For keyname, specify the logical key name that is used in the ENCRYPTTRAIL statement. This key must exist in a local ENCKEYS lookup file. For more information on using trail or file encryption, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Example

```
DECRYPTTRAIL AES192, KEYNAME mykey1
```

DEFERAPPLYINTERVAL

Valid for Replicat

Use the DEFERAPPLYINTERVAL parameter to set an amount of time that Replicat waits before applying captured transactions to the target database. To determine when to apply the transaction, Replicat adds the delay value to the commit timestamp of the source transaction, as recorded in the local GMT time of the source system.

You can use DEFERAPPLYINTERVAL for such purposes as to prevent the propagation of erroneous changes made to the source data, to control data arrival across different time zones, and to allow time for other planned events to occur before the data is applied to the target. Note that by using DEFERAPPLYINTERVAL, you are purposely building latency into the target data, and it should be used with caution if the target applications are time-sensitive.

To find out if Replicat is deferring operations, use the SEND REPLICAT command with the STATUS option and look for a status of Waiting on deferred apply.

NOTE If the TCPSOURCETIMER parameter is in use, it is possible that the timestamps of the source and target transactions could vary by a few seconds, causing Replicat to hold its transaction (and hence row locks) open for a few seconds. This small variance should not have a noticeable affect on performance.

Default 0 (no delay)

Syntax

```
DEFERAPPLYINTERVAL <n><unit>
```

Argument	Description
`<n>`	A numeric value for the amount of time to delay. The minimum delay time is the value that is set for the EOFDELAY parameter. The maximum delay time is seven days.
`<unit>`	The unit of time for the delay. Can be: S | SEC | SECS | SECOND | SECONDS | MIN | MINS | MINUTE | MINUTES | HOUR | HOURS | DAY | DAYS

Example This example directs Replicat to wait ten hours before posting its transactions.

```
DEFERAPPLYINTERVAL 10
```

If a transaction completes at 08:00:00 source GMT time, and the delay time is 10 hours, the transaction will be applied to the target at 18:00:00 target GMT time the same day.

DEFSFILE

Valid for DEFGEN

Use the DEFSFILE parameter to identify the name of the file to which DEFGEN will write data definitions. By default, the data definitions file is written in the character set of the local operating system. You can change the character set with the CHARSET option.

Default None

Syntax

```
DEFSFILE <filename> [APPEND | PURGE] [CHARSET <character set>]
```

Argument	Description
`<filename>`	The relative or fully qualified file name. The file is created when you run DEFGEN.
`APPEND`	Directs DEFGEN to write new content (from the current run) at the end of any existing content, if the specified file already exists. When APPEND is used and the definitions file already exists, DEFGEN appends the data definitions in following manner: ◆ If the existing data definitions file is in a format older than Oracle GoldenGate 11.2.1, DEFGEN appends the table definitions in the old format, where table and column names with multi-byte and special characters are not supported. ◆ If the existing data definitions file is in the newer format introduced in version 11.2.1, DEFGEN appends the table definitions in the existing character set of the file.
`PURGE`	Directs DEFGEN to purge the specified file before writing new content from the current run. This is the default.
`CHARSET <character set>`	Generates the definitions file in the specified character set. Without CHARSET, the default character set of the operating system is used. If APPEND mode is specified for a definitions file that is version 11.2.1 or later, CHARSET is ignored, and the character set of the existing definitions file is used.

Example

```
DEFSFILE ./dirdef/orcldef CHARSET ISO-8859-11
```

DISCARDFILE

Valid for Extract and Replicat

Use the DISCARDFILE parameter to generate a discard file to which Oracle GoldenGate can log records that it cannot process. Records can be discarded for several reasons. For example, a record is discarded if the underlying table structure changed since the record was written to the trail. You can use the discard file to help you identify the cause of processing errors.

Each entry in the discard file contains the discarded record buffer and an error code indicating the reason. Oracle GoldenGate creates the specified discard file in the dirrpt subdirectory of the Oracle GoldenGate installation directory. You can view it with a text editor or by using the following command in GGSCI.

```
VIEW REPORT <file name>
```

Where: <file name> is the name of the discard file.

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 and will terminate.

To set an upper limit for the size of the file, use either the MAXBYTES or MEGABYTES option. If the specified size is exceeded, the process will abend.

Default By default, Oracle GoldenGate does not generate a discard file.

Syntax

```
DISCARDFILE <file name>
[, APPEND | PURGE]
[, MAXBYTES <n> | MEGABYTES <n>]
```

Argument	Description
`<file name>`	The relative or fully qualified name of the discard file. If the file is in the Oracle GoldenGate directory, a relative path name is sufficient, because Oracle GoldenGate qualifies the name with the Oracle GoldenGate installation directory. Use the same name as that of the process group whenever possible.
`APPEND`	Adds new content to existing content if the file already exists.
`PURGE`	Purges the file before writing new content.
`MAXBYTES <n>`	Sets the maximum size of the file in bytes. The valid range is from 1 to 2147483646. The default is 1000000.
`MEGABYTES <n>`	Sets the maximum size of the file in megabytes. The valid range is from 1 to 2147. The default is 1 MB.

Example

```
DISCARDFILE discard.dsc, PURGE, MEGABYTES 2
```

DISCARDROLLOVER

Valid for Extract and Replicat

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.

When the DISCARDROLLOVER age point is reached, a new discard file is created, and old files are renamed in the format of <group name><n>.<extension>, where:

- <group name> is the name of the Extract or Replicat group
- <n> is a number that gets incremented by one each time a new file is created, for example: myext0.dsc, myext1.dsc, myext2.dsc, and so forth.

You can specify a time of day, a day of the week, or both. Specifying just a time of day (AT option) without a day of the week (ON option) generates a discard file at the specified time every day.

Default Disabled. No rules specified.

Syntax

```
DISCARDROLLOVER
{AT <hh:mi> |
ON <day of week> |
AT <hh:mi> ON <day of week>}
```

Argument	Description
`AT <hh:mi>`	The time of day to age the file based on a 24-hour clock. Valid values: ◆ <hh> is an hour of the day from 1 through 23. ◆ <mi> is minutes from 00 through 59.

`ON <day of week>`	The day of the week to age the file.
	Valid values:
	`SUNDAY`
	`MONDAY`
	`TUESDAY`
	`WEDNESDAY`
	`THURSDAY`
	`FRIDAY`
	`SATURDAY`
	They are not case-sensitive.

Example 1 `DISCARDROLLOVER AT 05:30`

Example 2 `DISCARDROLLOVER ON friday`

Example 3 `DISCARDROLLOVER AT 05:30 ON friday`

DOWNCRITICAL

Valid for Manager

Use the DOWNCRITICAL parameter to include processes that either abended or stopped normally in the report that is generated by the DOWNREPORT parameter.

Default None

Syntax `DOWNCRITICAL`

DOWNREPORT

Valid for Manager

Use the DOWNREPORTMINUTES or DOWNREPORTHOURS parameter to specify the frequency with which Manager reports Extract and Replicat processes that are not running. Whenever a process starts or stops, events are generated to the error log, and those messages can easily be overlooked if the log is large. DOWNREPORTMINUTES and DOWNREPORTHOURS report on a periodic basis to ensure that you are aware of stopped processes.

To report on running processes, use the UPREPORT parameter.

Default Do not report down processes.

Syntax

```
DOWNREPORTMINUTES <minutes> | DOWNREPORTHOURS <hours>
```

Argument	Description
<minutes>	The frequency, in minutes, to report processes that are not running.
<hours>	The frequency, in hours, to report processes that are not running.

Example The following generates a report every 30 minutes.

```
DOWNREPORTMINUTES 30
```

DSOPTIONS

Valid for Extract

Use the DSOPTIONS parameter to specify Extract processing options for extraction that uses a Teradata Access Module (TAM). For more information about configuring Oracle GoldenGate for Teradata extraction, see the Oracle GoldenGate *Teradata Installation and Setup Guide*.

Default None

Syntax

```
DSOPTIONS
[COMMITTEDTRANLOG]
[CREATETRANLOG]
[SORTTRANLOG]
[RESTARTAPPEND]
```

Argument	Description
COMMITTEDTRANLOG	(Maximum performance mode) Specifies that transaction data will not be persisted to disk. The TAM will send transaction data changes to an Extract process that will save it to a regular Oracle GoldenGate trail in transaction commit order. The trail can be read by Replicat or by a data pump Extract group.
CREATETRANLOG	(Maximum protection mode) Causes Extract to create a VAM trail, a local trail to which transaction data changes will be persisted for further processing by a VAM-sort Extract. Data will be written to the VAM trail in log-style format, which interleaves change records from different transactions. Use this option for the primary Extract process that interfaces with a Teradata Access Module (TAM) and writes to the VAM trail. Specify the VAM trail with the ADD EXTTRAIL command in GGSCI.

Argument	Description
`SORTTRANLOG`	(Maximum protection mode) Indicates that Extract needs to perform transaction sorting functions. Use this option for a VAM-sort Extract group that reads a VAM trail that is populated by a primary Extract process. The VAM-sort Extract sorts the interleaved operations into the correct prepare/commit/rollback transactional units before further processing by Oracle GoldenGate.
`RESTARTAPPEND`	(Maximum performance mode) Directs Extract to append data to the end of the Oracle GoldenGate trail upon restart, rather than rewriting data that was written in a previous run. Use this option with the COMMITTEDTRANLOG argument.

DYNAMICPORTLIST

Valid for Manager

Use the DYNAMICPORTLIST parameter to specify a list of available ports to which the following local Oracle GoldenGate processes can bind for communication with a remote Oracle GoldenGate process:

- Collector: to communicate with a remote Extract to receive incoming data.
- Replicat: to communicate with a remote Extract to receive data during an initial load task.
- Passive Extract: to communicate with a remote Collector
- GGSCI: to issue remote commands

Specify enough ports to accommodate expansion of the number of processes without having to stop and restart Manager to add them to the list. You can specify an individual port, a range of ports, or both.

Default None

Syntax `DYNAMICPORTLIST {<port> | <port>-<port>} [ , ...]`

Argument	Description

`<port>`	A port number that can be allocated. The maximum number of port entries is 5000.
	◆ To specify multiple ports, use a comma-delimited list. Example:
	`7830, 7833`
	◆ To specify a range of ports, use a dash (-) to separate the first and last port in the range. Do not put any spaces before or after the dash. Example:
	`7830-7835`
	◆ To specify a range of ports plus an individual port, place a comma between the range and the individual port number. Example:
	`7830-7835, 7839`

Example `DYNAMICPORTLIST 7820-7830, 7833, 7835`

DYNAMICRESOLUTION | NODYNAMICRESOLUTION

Valid for Extract and Replicat

Use the DYNAMICRESOLUTION and NODYNAMICRESOLUTION parameters to control how table names are resolved.

DYNAMICRESOLUTION, the default, enables fast process startup when there are numerous tables specified in TABLE or MAP statements. To get metadata for transaction records that it needs to process, Oracle GoldenGate queries the database and then builds a record of the tables that are involved. DYNAMICRESOLUTION causes the record to be built one table at a time, instead of all at once. The metadata of any given table is added when Extract first encounters the object ID in the transaction log, while record-building for other tables is deferred until their object IDs are encountered. DYNAMICRESOLUTION is the same as WILDCARDRESOLVE DYNAMIC.

NODYNAMICRESOLUTION causes the entire object record (for all tables) to be built at startup, which can be time-consuming if the database is large. This option is not supported for Teradata. NODYNAMICRESOLUTION is the same as WILDCARDRESOLVE IMMEDIATE.

For more information about WILDCARDRESOLVE, see page 423.

Default `DYNAMICRESOLUTION`

Syntax `DYNAMICRESOLUTION | NODYNAMICRESOLUTION`

DYNSQL | NODYNSQL

Valid for Replicat

Use the DYNSQL and NODYNSQL parameters to control the way that SQL statements are formed. With NODYNSQL, Replicat uses literal SQL statements with the bind variables resolved. With DYNSQL, the default, Replicat uses dynamic SQL to compile a statement once, and then execute it many times with different bind variables.

- Statement with DYNSQL:
  ```
  UPDATE <table> ... WHERE ID = :B
  ```
- Statement with NODYNSQL:
  ```
  UPDATE <table> ... WHERE ID = ‘1234’
  ```

In most environments, using DYNSQL yields the best efficiency and most throughput. However, in isolated instances, using NODYNSQL has proven faster and more efficient. Try NODYNSQL only if Replicat throughput appears unsatisfactory.

Do not use DYNSQL when replicating to target databases that do not support dynamic SQL.

When using NODYNSQL, you must also use the NOBINARYCHARS parameter. Contact Oracle Support before using either of these parameters. For more information, go to http://support.oracle.com.

Oracle GoldenGate for MySQL does not support LOB replication in NODYNSQL mode.

Default `DYNSQL`

Syntax `DYNSQL | NODYNSQL`

EBCDICTOASCII

Valid for Extract data pump and Replicat

Use the EBCDICTOASCII parameter to convert character data in the input trail from EBCDIC to ASCII format when sending it to a DB2 target database on a z/OS system. This parameter can be specified to request conversion of all EBCDIC columns and user token data to ASCII. This parameter must precede the SOURCEDB parameter. This parameter is only needed if the input trail file was created by an Extract version prior to v10.0. It is ignored for all other cases, because the conversion is done automatically.

As of version 11.2.1, conversion is not allowed by a data pump.

Default None

Syntax `EBCDICTOASCII`

ENABLEMONITORING

Valid for GLOBALS

Use the ENABLEMONITORING parameter to enable the monitoring of Oracle GoldenGate instances from Oracle GoldenGate Monitor. It directs Manager to publish the monitoring points that provide status and other information to the Oracle GoldenGate Monitor clients. To control the Java agent that sends the information from the monitoring points to the Oracle GoldenGate Monitor Server, use the following commands in GGSCI:

CREATE DATASTORE

REPAIR DATASTORE

START JAGENT

STOP JAGENT

Before enabling monitoring on any given platform, consult the *Oracle GoldenGate Monitor Administrator's Guide* to make certain that the operating system is supported. That guide also contains instructions for installing and configuring Oracle GoldenGate Monitor. For help with using Oracle GoldenGate Monitor, see the online help.

This parameter is not valid for NonStop SQL/MX.

Default Disabled

Syntax `ENABLEMONITORING`

ENCRYPTTRAIL | NOENCRYPTTRAIL

Valid for Extract

Use the ENCRYPTTRAIL and NOENCRYPTTRAIL parameters to encrypt data that is written to a trail or extract file. ENCRYPTTRAIL encrypts all records across data links and within the files themselves. NOENCRYPTTRAIL implies no encryption and is the default.

You can use encryption for trails or extract files that are specified with the following parameters in the Extract parameter file.

EXTTRAIL and RMTTRAIL

EXTFILE and RMTFILE

ENCRYPTTRAIL and NOENCRYPTTRAIL are trail or file-specific. One affects all subsequent trail or extract file specifications in the parameter file until the other parameter is encountered. The parameter must be placed before the parameter entry for the trail that it will affect.

To decrypt the data, use the DECRYPTTRAIL parameter (see page 183) in the parameter files of the following processes:

- Data pumps that read the encrypted files and must perform column mapping, filtering, transformation, or other operations. You can include ENCRYPTTRAIL in the parameter file of the data pump to encrypt any or all of the data again before it is written to downstream trails or files, or you can omit ENCRYPTTRAIL to send the data in unencrypted format.
- Replicat processes that read encrypted trails or file.

ENCRYPTTRAIL and NOENCRYPTTRAIL 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 Oracle GoldenGate canonical format.

ENCRYPTTRAIL encrypts only the data blocks. User tokens are not encrypted.

ENCRYPTTRAIL is not compatible with ETOLDFORMAT.

Default `NOENCRYPTTRAIL`

Syntax

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

Argument	Description
`ENCRYPTTRAIL`	ENCRYPTTRAIL without options specifies 256-key byte substitution. This format is not secure and should not be used in a production environment. Use only for backward compatibility with earlier Oracle GoldenGate versions.
`{AES128 | AES192 | AES256} KEYNAME <keyname>`	Specifies Advanced Encryption Standard (AES) encryption. This is a symmetric-key encryption standard that is used by governments and other organizations that require a high degree of data security. Specify one of the AES ciphers to encrypt the file(s): ◆ AES128 has a 128-bit block size with a key size of 128 bits. ◆ AES192 has a 128-bit block size with a key size of 192 bits. ◆ AES256 has a 128-bit block size with a key size of 256 bits. For <keyname>, specify the logical name of a user-defined encryption key. Oracle GoldenGate uses the key name to look up the actual key in an ENCKEYS lookup file. To use AES, you must: ◆ Generate the encryption key. ◆ Store it in an ENCKEYS lookup file. ◆ Copy ENCKEYS to every system where encryption or decryption (or both) are performed.

Argument	Description
	To use AES encryption for any database other than Oracle, the path of the lib sub-directory of the Oracle GoldenGate installation directory must be specified as an environment variable before starting any processes: ◆ UNIX: Specify the path as an entry to the LD_LIBRARY_PATH or SHLIB_PATH variable. For example: `setenv LD_LIBRARY_PATH ./lib:$LD_LIBRARY_PATH` ◆ Windows: Add the path to the PATH variable. You can use the SETENV parameter to set it as a session variable for the process. For more information about using encryption, see the security guidelines in the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Example In the following example, the Extract process writes to two trails. The data for the emp table is written to trail "em," which is encrypted with the AES-192 cipher. The data for the stores table is written to trail "st," which is not encrypted.

```
ENCRYPTTRAIL AES192, KEYNAME mykey1
RMTTRAIL /home/ggsora/dirdat/em
TABLE hr.emp;
NOENCRYPTTRAIL
RMTTRAIL /home/ggsora/dirdat/st
TABLE ops.stores;
```

Example As an alternative to the preceding example, you can omit NOENCRYPTTRAIL if you list all non-encrypted trails before the ENCRYPTTRAIL parameter.

```
RMTTRAIL /home/ggsora/dirdat/st
TABLE ops.stores;
ENCRYPTTRAIL AES192, KEYNAME mykey1
RMTTRAIL /home/ggsora/dirdat/em
TABLE hr.emp;
```

For additional examples, see the security guidelines in the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

END

Valid for Replicat

Use the END parameter to terminate Replicat when it encounters the first record in the data source whose timestamp is the specified point in time.

Without END, the process runs continuously until:

- the end of the transaction log or trail is reached, at which point it will stop gracefully.
- manually terminated from the command shell.

Use END with the SPECIALRUN parameter to post data as a point-in-time snapshot, rather than continuously updating the target tables.

Default Continuous processing

Syntax
```
END {<date> [<time>] | RUNTIME}
```

Argument	Description
`<date> [<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.

Example 1
```
SPECIALRUN
END 2010-12-31 17:00:00
```

Example 2
```
SPECIALRUN
END RUNTIME
```

EOFDELAY | EOFDELAYCSECS

Valid for Extract and Replicat

Use the EOFDELAY or EOFDELAYCSECS parameter to control how often Extract, a data pump, or Replicat checks for new data after it has reached the end of the current data in its data source. You can reduce the system I/O overhead of these reads by increasing the value of this parameter.

NOTE Large increases can increase the latency of the target data, especially when the activity on the source database is low

This parameter is not valid when SOURCEISTABLE is used.

Default 1 second

Syntax
```
EOFDELAY <seconds> | EOFDELAYCSECS <centiseconds>
```

Argument	Description
`<seconds>`	The delay, in seconds, before searching for data to process.

Argument	Description
`<centiseconds>`	The delay, in centiseconds, before searching for data to process.

Example `EOFDELAY 3`

ETOLDFORMAT

Valid for Extract

Use the ETOLDFORMAT parameter to generate trails in a format that is compatible with Replicat versions prior to Oracle GoldenGate version 6.0.

From Oracle GoldenGate version 10 and later, ETOLDFORMAT implies a compatibility level of 0. ETOLDFORMAT applies globally to all output files. If any compatibility level is specified with EXTFILE, EXTTRAIL, RMTFILE, or RMTTRAIL, in addition to ETOLDFORMAT, then the Extract process will abend. For more information, see the documentation for those parameters.

ETOLDFORMAT is not compatible with ENCRYPTTRAIL.

Default None

Syntax `ETOLDFORMAT`

EXTFILE

Valid for Extract and Replicat

Use the EXTFILE parameter to specify an extract file, which is a local file that will be read by a data pump Extract group on the local system, or to specify a local extract file that Replicat reads when SPECIALRUN is used. The size of an extract file cannot exceed 2GB .

EXTFILE must precede all associated TABLE or MAP statements. Multiple EXTFILE statements can be used to define different files.

You can encrypt the data in this file by using the ENCRYPTTRAIL parameter (page 191).

Default None

Syntax

```
EXTFILE <file name>
[, MAXFILES <number>]
[, MEGABYTES <megabytes>]
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<file name>`	Valid for Extract and Replicat. Specifies the relative or fully qualified name of the extract file.

Argument	Description
`MAXFILES <number>`	Valid for Extract. Forces a sequence of files to be created, rather than a single file. Use when you expect the size of a file to exceed the limit permitted by the operating system. MAXFILES permits as many files to be created as needed. Aged files are appended with a six-digit sequence number, for example datafile000002. Checkpoints are not maintained in these files. When using MAXFILES, also use MEGABYTES to set the maximum size of each file in the sequence.
`MEGABYTES <megabytes>`	Valid for Extract. Defines the maximum size of the file (or of each file created when MAXFILES is used). The size of an extract file cannot exceed 2GB .

`FORMAT`
`RELEASE <major>.<minor>`
	Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones.
	◆ FORMAT is a required keyword.
	◆ RELEASE specifies an Oracle GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current Oracle GoldenGate version number. (If you use an Oracle GoldenGate version that is earlier than 9.0, specify either 9.0 or 9.5.) The release version is programatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail.
	There is a dependency between FORMAT and the RECOVERYOPTIONS parameter. When RECOVERYOPTIONS is set to APPENDMODE, FORMAT must be set to RELEASE 10.0 or greater. When RECOVERYOPTIONS is set to OVERWRITEMODE, FORMAT must be set to RELEASE 9.5 or less.

Example 1 `EXTFILE dirdat/datafile`

Example 2 `EXTFILE dirdat/extdat, MAXFILES 3, MEGABYTES 2`

Example 3 `EXTFILE /ggs/dirdat/extdat, FORMAT RELEASE 10.4`

EXTRACT

Valid for Extract

Use the EXTRACT parameter to specify an Extract group for online change synchronization. This parameter links the current run with previous runs, so that data changes are continually processed to maintain synchronization between source and target tables.

Extract will run continuously and maintain checkpoints in the data source and trail to ensure data integrity and fault tolerance throughout planned or unplanned process termination, system outages, or network failure. EXTRACT must be the first entry in the parameter file.

For more information about implementing change synchronization, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Default None

Syntax
```
EXTRACT <group name>
```

Argument	Description
`<group name>`	The group name as defined with the ADD EXTRACT command.

Example The following specifies an Extract group named “finance.”

```
EXTRACT finance
```

EXTTRAIL

Valid for Extract and Replicat

Use the EXTTRAIL parameter to specify a trail on the local system that was created with the ADD EXTTRAIL command. The trail is read by a data pump Extract group or by a Replicat group on the local system.

EXTTRAIL must precede all associated TABLE statements. Multiple EXTTRAIL statements can be used to define different trails.

Do not use EXTTRAIL for an Extract that is configured in PASSIVE mode. See “ADD EXTRACT” on page 17 for more information.

You can encrypt the data in this trail by using the ENCRYPTTRAIL parameter (page 191).

Default None

Syntax
```
EXTTRAIL <file name>
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<file name>`	The relative or fully qualified name of the trail. Use a maximum of two characters for the name. As trail files are aged, a six-character sequence number will be added to this name, for example /ggs/dirdat/rt000001.

Argument	Description
`FORMAT RELEASE <major>.<minor>`	Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones. ◆ FORMAT is a required keyword. ◆ RELEASE specifies an Oracle GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current Oracle GoldenGate version number. (If you use an Oracle GoldenGate version that is earlier than 9.0, specify either 9.0 or 9.5.) The release version is programatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail. There is a dependency between FORMAT and the RECOVERYOPTIONS parameter. When RECOVERYOPTIONS is set to APPENDMODE, FORMAT must be set to RELEASE 10.0 or greater. When RECOVERYOPTIONS is set to OVERWRITEMODE, FORMAT must be set to RELEASE 9.5 or less. See Appendix 2 on page 562 for additional information about Oracle GoldenGate trail file versioning and recovery modes.

Example 1 `EXTTRAIL dirdat/ny`

Example 2 `EXTTRAIL /ggs/dirdat/ex, FORMAT RELEASE 10.4`

FETCHOPTIONS

Valid for Extract

Use the FETCHOPTIONS parameter to control certain aspects of the way that Oracle GoldenGate fetches data in the following circumstances:

- When the transaction record does not contain enough information for Extract to reconstruct an update operation.
- When Oracle GoldenGate must fetch a column value as the result of a FETCHCOLS clause of a TABLE statement.

FETCHOPTIONS is table-specific. One FETCHOPTIONS statement applies for all subsequent TABLE statements until a different FETCHOPTIONS statement is encountered.

Default fetch properties are adequate for most installations.

Default Ignore missing rows and continue processing

Syntax

```
FETCHOPTIONS
[, FETCHPKUPDATECOLS]
[, MISSINGROW <action>]
[, NOFETCH]
[, USEKEY | NOUSEKEY]
[, USELATESTVERSION | NOUSELATESTVERSION]
[, USESNAPSHOT | NOUSESNAPSHOT]
[, USEROWID | NOUSEROWID]
```

Argument	Description
`FETCHPKUPDATECOLS`	Fetches all unavailable columns when a primary key is updated. This option is off by default. When off, column fetching is performed according to other FETCHOPTIONS options that are enabled. When on, it only takes effect during an update to a primary key column. The results are the same as using FETCHCOLS (*) in the TABLE statement. LOB columns are included in the fetch. Use this parameter when using HANDLECOLLISIONS. When Replicat detects a missing update, all of the columns will be available to turn the update into an insert.

`MISSINGROW <action>`	Provides a response when Oracle GoldenGate cannot locate a row to be fetched, causing only part of the row (the changed values) to be available for processing. Typically a row cannot be located because it was deleted between the time the change record was created and when the fetch was triggered, or because the row image required was older than the undo retention specification.
	<action> can be one of the following:
	`IGNORE`
	Ignore the condition and continue processing. This is the default.
	`REPORT`
	Report the condition and contents of the row to the discard file, but continue processing the partial row. A discard file must be specified with the DISCARDFILE parameter.
	`DISCARD`
	Discard the data and do not process the partial row. A discard file must be specified with the DISCARDFILE parameter.
	`ABEND`
	Discard the data and quit processing. A discard file must be specified with the DISCARDFILE parameter.

`NOFETCH`	Prevents Extract from fetching the column from the database. Extract writes the record to the trail, but inserts a token indicating that the column is missing.

Argument	Description

`USEKEY |`
`NOUSEKEY`
	Valid for Oracle. Determines whether or not Oracle GoldenGate uses the primary key to locate the row to be fetched.
	If both USEKEY and USEROWID are specified, ROWID takes priority for faster access to the record. USEROWID is the default.

`USELATESTVERSION |`
`NOUSELATESTVERSION`
	Valid for Oracle. Use with USESNAPSHOT. The default, USELATESTVERSION, directs Extract to fetch data from the source table if it cannot fetch from the undo tablespace. NOUSELATESTVERSION directs Extract to ignore the condition if the snapshot fetch fails, and continue processing.
	To provide an alternate action if a snapshot fetch does not succeed, use the MISSINGROW option.

`USESNAPSHOT |`
`NOUSESNAPSHOT`
	Valid for Oracle. The default, USESNAPSHOT, causes Extract to use the Oracle Flashback mechanism to fetch the correct snapshot of data that is needed to reconstruct certain operations that cannot be fully captured from the redo record. NOUSESNAPSHOT causes Extract to fetch the needed data from the source table instead of the flashback logs. For more information about how Oracle GoldenGate fetches data from Oracle, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

`USEROWID |`
`NOUSEROWID`
	Valid for Oracle. Determines whether or not Oracle GoldenGate uses the row ID to locate the row to be fetched.
	If both USEKEY and USEROWID are specified, ROWID takes priority for faster access to the record. USEROWID is the default.

Example 1 The following directs Extract to fetch data by using Flashback Query and to ignore the condition and continue processing the record if the fetch fails.

```
FETCHOPTIONS USESNAPSHOT, NOUSELATESTVERSION
```

Example 2 The following directs Extract to fetch data by using Flashback Query and causes Extract to abend if the data is not available.

```
FETCHOPTIONS USESNAPSHOT, NOUSELATESTVERSION, MISSINGROW ABEND
```

FILTERDUPS | NOFILTERDUPS

Valid for Replicat

Use the FILTERDUPS and NOFILTERDUPS parameters to handle anomalies that can occur on a NonStop system when an application performs multiple operations on the same record within the same transaction. This type of transaction can cause out-of-order records in the TMF audit trail and will cause Replicat to abend. For example:

- An insert can occur in the audit trail before a delete on the same primary key, even though the source application performed the delete first, followed by the insert (resulting in a duplicate-record error when the insert is performed by Replicat).

- An update can occur in the audit trail before an insert on the same primary key (resulting in a missing-record error when the update is performed by Replicat).

FILTERDUPS prevents Replicat from abending by resolving the conditions as follows:

- In the event of a duplicate insert, Replicat saves the duplicated insert until the end of the transaction. If a delete with the same primary key is subsequently encountered, Replicat performs the delete, then the insert.
- In the event of a missing update, Replicat saves the missing update until the end of the transaction. If an insert with the same primary key is subsequently encountered, Replicat performs the insert, then the update.

IDX hospital applications and some BASE24 bank applications are the typical, but not the only, sources of this anomaly. Use FILTERDUPS only if Replicat is abending on duplicate or missing records and you know they were caused by out-of-order transactions originating on a NonStop system. The Logdump utility can be used to diagnose this condition. See the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

FILTERDUPS and NOFILTERDUPS can be used as on-off switches for different groups of MAP statements to enable or disable the exception processing as needed.

Default NOFILTERDUPS

Syntax FILTERDUPS | NOFILTERDUPS

Example This example turns on FILTERDUPS for Orders but disables it for any MAP statements that are defined later in the same parameter file.

```
FILTERDUPS
MAP $DATA1.SQLDAT.ORDERS, TARGET MASTER.ORDERS;
NOFILTERDUPS
```

FLUSHSECS | FLUSHCSECS

Valid for Extract

Use the FLUSHSECS and FLUSHCSECS parameters to control when Oracle GoldenGate flushes the Extract memory buffer. When sending data to remote systems, Extract buffers data to optimize network performance. The buffer is flushed to the target system when it is full or after the amount of time specified with FLUSHSECS or FLUSHCSECS. Data changes are not available to the target users until the buffer is flushed and the data is posted. To control the size of the buffer, use the TCPBUFSIZE option of RMTHOST (see page 313).

Increasing the value of FLUSHSECS or FLUSHCSECS could result in slightly more efficient use of the network, but it could increase the latency of the target data if activity on the source system is low and the buffer does not fill up. When source tables remain busy, FLUSHSECS and FLUSHCSECS have little effect.

Default 1 second

Syntax FLUSHSECS <seconds> | FLUSHCSECS <centiseconds>

Argument	Description
`<seconds>`	The delay, in seconds, before flushing the buffer.
`<centiseconds>`	The delay, in centiseconds, before flushing the buffer.

Example `FLUSHSECS 80`

FORMATASCII

Valid for Extract

Use the FORMATASCII parameter to output data in external ASCII format instead of the default Oracle GoldenGate canonical format. Using FORMATASCII, you can format output that is compatible with most database load utilities and other programs that require ASCII input. This parameter is required by the *file-to-database-utility* initial load method.

A FORMATASCII statement affects all extract files or trails that are listed after it in the parameter file. The relative order of the statements in the parameter file is important. If listed after a file or trail specification, FORMATASCII will not take effect.

Limitations

- Do not use FORMATASCII if the data will be processed by the Replicat process. Replicat expects the default canonical format.
- Do not use FORMATASCII if FORMATSQL or FORMATXML is being used.
- Do not use FORMATASCII if the data contains LOBs.
- Do not use FORMATASCII if Extract is connected to a multi-byte DB2 subsystem.

Default output

Database object names, such as table and column names, and CHAR and VARCHAR data are written in the default character set of the operating system.

Without specifying any parameter options, FORMATASCII generates records in the following format.

Line 1, the following tab-delimited list:

- The operation-type indicator: I, D, U, V (insert, delete, update, compressed update).
- A before or after image indicator: B or A.
- The table name in the character set of the operating system.
- A column name, column value, column name, column value, and so forth.
- A newline character (starts a new line).

Line 2, the following tab-delimited begin-transaction record:

- The begin transaction indicator, B.
- The timestamp at which the transaction committed.
- The sequence number of the transaction log in which the commit was found.
- The relative byte address (RBA) of the commit record within the transaction log.

Line 3, the following tab-delimited commit record:

- The commit character C.
- A newline character.

Every record in a source transaction is contained between the begin and commit indicators. Each combination of commit timestamp and RBA is unique.

Custom output

You can customize the output format with optional arguments. See the syntax descriptions.

Default See "Default output".

Syntax `FORMATASCII [, <option>] [, ...]`

Option	Description
`BCP`	Formats the output for compatibility with SQL Server's BCP, DTS, or SQL Server Integration Services (SSIS) bulk-load utility.
`DATE | TIME | TS`	Outputs one of the following: ◆ DATE outputs the date (year to day). ◆ TIME outputs the time (year to second). ◆ TS outputs the transaction timestamp (year to fraction).
`CHARSET <set>`	(Oracle SQL*Loader) Specifies the encoding of ASCII characters in Oracle NCHAR columns. Valid value is UTF8. CHARSET allows the load to include character-length semantics when the source table contains NCHAR data and variable-length characters set to UTF-8. Note: If both NCHAR and CHAR columns contain 8-bit ASCII characters, the generated file will contain a mix of operating system-native 8-bit ASCII character coding and UTF-8 coding, and the load will not succeed.
`DELIMITER <delimiter>`	An alternative delimiter character (the default is tab). Valid values: ◆ `TAB` (delimit with tabs). ◆ A character enclosed within single quotes, for example, `'/'`.
`EXTRACOLS <number>`	Includes placeholders for additional columns at the end of each record. Use this option when a target table has more columns than the source table.
`NAMES | NONAMES`	Includes or excludes column names as part of the output. For compressed updates, column names are included unless you also specify the PLACEHOLDERS option.

Option	Description
`NOHDRFIELDS [IND], [OP]`	Suppresses output as follows: ◆ NOHDRFIELDS without options suppresses everything except the data values themselves. ◆ IND suppresses everything except the before or after indicator (B or A) and the data values. ◆ OP suppresses everything except the operation-type indicator (I, D, U, V) and the data values.
`NOQUOTE`	Excludes quotation marks from character data. Without NOQUOTE, characters are enclosed within single-quotes.
`NOTRANSTMTS`	Excludes transaction information.
`NULLISSPACE`	Outputs null columns as empty columns. Without NULLISSPACE, null columns are output as the word "NULL."
`PLACEHOLDERS`	Outputs a placeholder for missing columns. For example, if the second and fourth columns are missing in a four-column table, the data might look like: `‘ABC’,,123,,`
`SQLLOADER`	Produces a fixed-length, ASCII-formatted file that is compatible with the Oracle SQL*Loader utility or the IBM Load Utility program.

Example 1 The following examples are based on a source table named test.customer and a sample transaction. The examples show how various FORMATASCII options configure the output.

Table test.customer:

```
CUSTNAME     CHAR(10)    Primary key
LOCATION     CHAR(10)
BALANCE      INTEGER
```

Transaction:

```
INSERT INTO CUSTOMER VALUES ("Eric", "San Fran", 550);
UPDATE CUSTOMER SET BALANCE = 100 WHERE CUSTNAME = "Eric";
COMMIT;
```

Example 1 FORMATASCII without options produces the following:

```
B,2011-01-21:14:09:46.421335,8,1873474,
I,A,TEST.CUSTOMER,CUSTNAME,'Eric',LOCATION,
'San Fran',BALANCE,550,
V,A,TEST.CUSTOMER,CUSTNAME,'Eric',BALANCE,100,
C,
```

Example 2 FORMATASCII, NONAMES, DELIMITER '|' produces the following:

```
B|2011-01-21:14:09:46.421335|8|1873474|
I|A|CUSTOMER|'Eric'|'San Fran'|550|
V|A|CUSTOMER|CUSTNAME|'Eric'|BALANCE|100|
C|
```

The last record returns column names for the CUSTNAME and BALANCE columns because the record is a compressed update and PLACEHOLDERS was not used.

Example 3 FORMATASCII, NOHDRFIELDS, OP, TS, NONAMES, NOQUOTE produces the following:

```
I,CUSTOMER,2011-01-21:14:09:46.421335,Eric,San Fran,550,
V,CUSTOMER,2011-01-21:14:09:46.421335,Eric,,100,
```

The absence of a value for the second column in the compressed update record is indicated by two consecutive commas.

FORMATSQL

Valid for Extract

Use the FORMATSQL parameter to output data in external SQL format, instead of the default Oracle GoldenGate canonical format. FORMATSQL generates SQL statements (INSERT, UPDATE, DELETE) that can be applied to both SQL and Enscribe tables by utilities other than Oracle GoldenGate Replicat.

NOTE Do not use FORMATSQL if the data will be processed by the Replicat process. Replicat expects the default canonical format. Do not use FORMATSQL if FORMATASCII or FORMATXML is being used.

A FORMATSQL statement affects all extract files or trails defined after it.

Do not use FORMATSQL if Extract is connected to a multi-byte DB2 subsystem.

Default output

Database object names, such as table and column names, and CHAR and VARCHAR data are written in the default character set of the operating system.

Without options, FORMATSQL transactions are output as follows, in comma-delimited format:

- The begin-transaction indicator, B.
- The timestamp at which the transaction was committed.
- The sequence number of the transaction log in which the commit was found.
- The relative byte address (RBA) of the commit record within the transaction log.
- The SQL statements.
- The commit indicator, C.
- A newline indicator.

Every record in a transaction is contained between the begin and commit indicators. Each combination of commit timestamp and RBA is unique.

Customized output

You can customize the output format with optional arguments. See the syntax descriptions.

Default See "Default output"

Syntax `FORMATSQL [<option>] [, ...]`

Option	Description
`NONAMES`	Omits column names for insert operations, because inserts contain all column names. This option conserves file size.
`NOPKUPDATES`	Converts UPDATE operations that affect columns in the target primary key to a DELETE followed by an INSERT. By default (without NOPKUPDATES), the output is a standard UPDATE operation.
`ORACLE`	Formats records for compatibility with Oracle databases by converting date and time columns to a format accepted by SQL*Plus (for example: `TO_DATE('2011-01-01','YYYY-MM-DD')`

Example `FORMATSQL ORACLE, NONAMES`

FORMATXML

Valid for Extract

Use the FORMATXML parameter to output data in XML format, instead of the default Oracle GoldenGate canonical format. A FORMATXML statement affects all extract files or trails that are defined after it.

Database object names, such as table and column names, and CHAR and VARCHAR data are written in the default character set of the operating system.

When using FORMATXML, use the NOBINARYCHARS parameter. NOBINARYCHARS is an undocumented parameter that causes Oracle GoldenGate to treat binary data as a null-terminated string. Contact Oracle Support before using NOBINARYCHARS. For more information, go to http://support.oracle.com.

Limitations

- Do not use FORMATXML if the data will be processed by the Replicat process. Replicat expects the default canonical format. Do not use FORMATXML if FORMATASCII or FORMATSQL is being used.
- Do not use FORMATXML if Extract is connected to a multi-byte DB2 subsystem.

Default None

Syntax `FORMATXML [<option>] [, ...]`

Options	Description
`INLINEPROPERTIES | NOINLINEPROPERTIES`	Controls whether or not properties are included within the XML tag or written separately. INLINEPROPERTIES is the default.
`TRANS | NOTRANS`	Controls whether or not transaction boundaries and commit timestamps should be included in the XML output. TRANS is the default.

Example `FORMATXML NOINLINEPROPERTIES, NOTRANS`

FUNCTIONSTACKSIZE

Valid for Extract and Replicat

Use the FUNCTIONSTACKSIZE parameter to control the size of the memory stack that is used for processing Oracle GoldenGate column-conversion functions. The memory stack holds arguments supplied to and from an Oracle GoldenGate function. You should not need to use this parameter unless Oracle GoldenGate returns a message indicating that the size of the stack should be increased. The message is similar to:

```
Not enough stack space. Specify FUNCTIONSTACKSIZE greater than
{0,number,0}
```

This could happen when you are using a very large number of functions or arguments.

The default without FUNCTIONSTACKSIZE is 200 arguments, which optimizes the performance of Oracle GoldenGate and its usage of system memory. Increasing this parameter can adversely affect performance and the use of system memory.

FUNCTIONSTACKSIZE must appear in the parameter file before any parameters that include functions are listed. FUNCTIONSTACKSIZE is a global parameter. It affects all clauses in a parameter file.

Default 200 arguments

Syntax `FUNCTIONSTACKSIZE <stack size>`

Argument	Description
`<stack size>`	A value between 0 and 5000 that denotes the number of function arguments to allow in a parameter clause.

Example `FUNCTIONSTACKSIZE 300`

GENLOADFILES

Valid for Replicat

Use the GENLOADFILES parameter when using the *file-to-database-utility* initial load method to generate run and control files that are compatible with:

- Oracle's SQL*Loader utility
- Microsoft's BCP, DTS, or SQL Server Integration Services (SSIS) utility
- IBM's Load Utility (LOADUTIL).

A run file and a control file are generated for each MAP statement in the Replicat parameter file. Replicat stops after generating the control and run files and does not process data.

Use the run and control files with a data file that contains the data to be loaded into the target. To generate the data file, use the FORMATASCII parameter in the Extract parameter file. Use the SQLLOADER option of FORMATASCII for the Oracle and DB2 for z/OS utilities and use the BCP option for the Microsoft utility.

FORMATASCII outputs the table data to an Oracle GoldenGate trail or file in external ASCII format, which is compatible with the load utility. You can generate multiple data files by specifying multiple files. For step-by-step instructions on configuring Oracle GoldenGate to output the load files and performing the initial load, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

NOTE For IBM's Load Utility, you will need to specify the -E and -d <defs file> Collector parameters with the PARAMS option of the RMTHOST parameter. These parameters are necessary to convert ASCII to EBCDIC and to specify the source-definitions file.

By default, GENLOADFILES creates the following file names:

- The SQL*Loader run file is named <source table>.run, and the control file is named <source table>.ctl, where <source table> is the name of a source table specified in the MAP statement.
- The BCP/DTS/SSIS run file is named <target table>.bat, and the control file is named <target table>.fmt, where <target table> is the name of a target table specified in the MAP statement.
- The Load Utility run file is named <target table>.run, and the control file is named <target table>.ctl, where <target table> is the name of a target table specified in the MAP statement.

Control files

The control file contains load parameters that are generated based on a template. Oracle GoldenGate provides default templates for SQL*Loader, BCP/DTS/SSIS, and Load Utility. You can modify the templates as needed to change the load rules, or you can create new templates.

The following are examples of the Oracle GoldenGate templates, which contain placeholders for the target tables, the data file(s) produced by FORMATASCII, and other run parameters. Oracle GoldenGate replaces the placeholders with values based on parameters specified in the Replicat parameter file.

Figure 15 SQL*Loader template sqlldr.tpl

```
# File Names
controlfile ?target.ctl
runfile     ?target.run
#
# Run File Template
sqlldr userid=?pw control=?target log=?target direct=true
#
# Control File Template
unrecoverable
load data
infile ?source.dat
truncate
into table ?target
```

Figure 16 BCP/DTS/SSIS template bcpfmt.tpl

```
# Run File Template
# Substitute your database name for <db>
bcp <db>..?target in ?source.dat -U ?user -P ?pw -f ?target.fmt -e
?target.err
#
# Control File Template
# The value below must specify the BCP version, not the Sybase Adaptive
# Server or Microsoft SQL Server version. "bcp -v" can be used to
# determine the correct version number.
12.0
```

Figure 17 Load Utility template db2cntl.tpl

```
# File Names
controlfile ?target.ctl
runfile     ?target.run
#
# Run File Template
odb2 load
#
# Control File Template
LOAD REPLACE INTO TABLE ?target
```

Run files

The run files contain the input parameters for starting the load. To execute the files, issue one of the following commands.

- Execute the SQL*Loader run file from the UNIX command shell.

  ```
  % <table>.run
  ```

- Execute the BCP run file from the DOS shell.

  ```
  > <table>.bat
  ```

- Execute the Load Utility run file with a JCL script to load the data to the DB2 for z/OS table. Add other environment-related parameters to the job script as needed.

NOTE A setting of DYNAMIC for the WILDCARDRESOLVE parameter is not compatible with the GENLOADFILES parameter. Oracle GoldenGate defaults to IMMEDIATE when GENLOADFILES is specified.

Note that Oracle GoldenGate does not support multi-byte characters when the operating system and database character set are different, or when fixed length output format is used.

Default None

Syntax

```
GENLOADFILES [<template file>]
[CHARSET <value>]
```

Argument	Description
`<template file>`	The fully qualified name of the template file. The default template file is sqlldr.tpl for SQL*Loader, bcpfmt.tpl for BCP, DTS, or SSIS, and db2cntl.tpl for DB2 on z/OS, all located in the Oracle GoldenGate home directory.
`CHARSET <set>`	(Oracle SQL*Loader) Specifies the encoding of ASCII characters in Oracle NCHAR columns. Valid value is UTF8. Required if using CHARSET option of FORMATASCII. CHARSET allows the load to include character-length semantics when the source table contains NCHAR data and variable-length characters set to UTF-8. Currently, Oracle SQL*Loader uses byte-length semantics and is not compatible with character-length semantics. Note: If both NCHAR and CHAR columns contain 8-bit ASCII characters, the generated file will contain a mix of operating system-native 8-bit ASCII character coding and UTF-8 coding, and the load will not succeed.

Example

```
GENLOADFILES sqlldr.tpl
```

GETAPPLOPS | IGNOREAPPLOPS

Valid for Extract

Use the GETAPPLOPS or IGNOREAPPLOPS parameter to capture or ignore DML operations produced by any application except the local Replicat. By default, application data is captured.

These parameters are useful in conjunction with the GETREPLICATES and IGNOREREPLICATES parameters for the following:

- To separate data operations performed by a local Replicat from those performed by the business applications configured for Oracle GoldenGate extraction. Use IGNOREAPPLOPS and GETREPLICATES for one trail or file to contain just the Replicat operations, and use GETAPPLOPS and IGNOREREPLICATES for another trail or file to contain just the operations of the business applications.

- As part of a cascading configuration, where changes applied by Replicat locally must be captured by a local Extract to be propagated to another system. In this case, IGNOREAPPLOPS and GETREPLICATES would be used.
- As part of a loop detection scheme when using bidirectional replication. The default combination of GETAPPLOPS and IGNOREREPLICATES causes Extract to capture application data while ignoring Replicat operations posted to the same database objects. In addition to using these parameters, Extract must be configured to identify Replicat transactions.

For more information about IGNOREREPLICATES, see page 212.

For more information about configuring bidirectional replication, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Using GETAPPLOPS for Oracle sequences

GETAPPLOPS must be enabled to capture sequences that are replicated by Replicat. Replicat issues sequence updates in an autonomous transaction, so they are not reflected in the trace table. The sequence update appears as if it is an application operation.

Using GETAPPLEOPS for DDL operations

To use GETAPPLOPS or IGNOREAPPLOPS functionality for DDL operations, see the DDLOPTIONS parameter on page 173.

Default `GETAPPLOPS`

Syntax `GETAPPLOPS | IGNOREAPPLOPS`

GETDELETES | IGNOREDELETES

Valid for Extract and Replicat

Use the GETDELETES and IGNOREDELETES parameters to control whether or not Oracle GoldenGate processes delete operations. These parameters are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements, until the other parameter is encountered.

Default `GETDELETES`

Syntax `GETDELETES | IGNOREDELETES`

GETENV

Valid for Extract and Replicat

Use the GETENV parameter to view environment variables that were set with the SETENV parameter. The results are printed to screen and the report file. Use one GETENV statement per variable to be retrieved.

Default None

Syntax `GETENV (<environment variable>)`

Options	Description
`<environment variable>`	The name of the environment variable.

Example The following shows GETENV statements and sample return values.

```
GETENV (ORACLE_HOME)
ORACLE_HOME = /home/oracle/ora9/product

GETENV (ORACLE_SID)
ORACLE_SID = ora9
```

GETINSERTS | IGNOREINSERTS

Valid for Extract and Replicat

Use the GETINSERTS and IGNOREINSERTS parameters to control whether or not insert operations are processed by Oracle GoldenGate. The parameters are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements, until the other parameter is encountered.

Default `GETINSERTS`

Syntax `GETINSERTS | IGNOREINSERTS`

GETREPLICATES | IGNOREREPLICATES

Valid for Extract

Use the GETREPLICATES and IGNOREREPLICATES parameters to control whether or not DML transactions issued by Replicat are captured or ignored by an Extract process that is processing the same tables on the same system.

These parameters are not valid for Teradata.

Ignoring Replicat transactions

By default, Extract uses a combination of IGNOREREPLICATES and GETAPPLOPS. In this configuration, Extract captures all application data that is configured for synchronization by Oracle GoldenGate, and it ignores all Replicat operations. In a bi-directional configuration, this prevents the data that Replicat applies from looping back to the original system, which would cause duplicate-record errors.

Capturing Replicat transactions

Use GETREPLICATES with IGNOREAPPLOPS in a cascading configuration to enable replicated data to be captured again by Extract on an intermediary system so that it can be replicated to the final target. For example, if database A replicates to database B, and database B replicates to database C, you would use GETREPLICATES for the Extract on database B.

NOTE Even with GETREPLICATES in effect, however, you still can exclude specific replicated data from being captured by using a WHERE or FILTER clause in a TABLE or MAP statement.

Identifying Replicat transactions

For some databases, if you want Extract to ignore Replicat transactions, you must identify those transactions to Extract, in addition to using IGNOREREPLICATES or GETREPLICATES. For more information about identifying Replicat transactions, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

Additional information about these parameters

- For more information about creating a checkpoint table, see the ADD CHECKPOINTTABLE command on page 92.
- For more information about creating and using a trace table, see the TRACETABLE parameter on page 382 and ADD TRACETABLE on page 95.
- For more information about TRANLOGOPTIONS, see page 385.
- For more information about SQLEXEC, see page 332.
- For more information about using a cascading or bidirectional configuration, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.
- For more information about GETAPPLOPS and IGNOREAPPLOPS, see page 210.
- To use GETAPPLOPS or IGNOREAPPLOPS functionality for DDL operations, see the DDLOPTIONS parameter on page 173.

Default `IGNOREREPLICATES`

Syntax `GETREPLICATES | IGNOREREPLICATES`

GETTRUNCATES | IGNORETRUNCATES

Valid for Extract and Replicat

Use the GETTRUNCATES and IGNORETRUNCATES parameters to control whether or not Oracle GoldenGate processes table truncate operations. By default, truncate operations are not captured from the source or replicated to the target.

GETTRUNCATES and IGNORETRUNCATES are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements, until the other parameter is encountered.

Supported databases

- GETTRUNCATES and IGNORETRUNCATES are not supported for Teradata.
- GETTRUNCATES and IGNORETRUNCATES are supported by Extract for Oracle, SQL Server 2005 upgraded to CU6 and later versions, and Sybase.
- GETTRUNCATES and IGNORETRUNCATES are supported by Replicat for Oracle, SQL Server 2005 upgraded to CU6 and later versions, Sybase, DB2 LUW, DB2 z/OS, MySQL, and other ODBC targets that support the TRUNCATE command.

NOTE It is not possible to ignore TRUNCATEs during capture from a DB2 on z/OS database. By default, TRUNCATEs are always captured from a DB2 on z/OS source, but they can be ignored by Replicat if IGNORETRUNCATES is used in the Replicat parameter file.

DB2 LUW Limitations

- DB2 LUW does not support a TRUNCATE command, so Replicat replicates a truncate operation by performing an IMPORT REPLACE from a NULL (blank) file.

Oracle Limitations

- Oracle GoldenGate supports the Oracle TRUNCATE TABLE command, but not TRUNCATE PARTITION. You can replicate TRUNCATE PARTITION as part of the full Oracle GoldenGate DDL replication support.
- The database does not log truncates against an empty table, so those operations are not captured by Oracle GoldenGate. The DDL support of Oracle GoldenGate can be used for this purpose.
- The database does not log truncates for empty partitions, so Oracle GoldenGate cannot reliably process TRUNCATE TABLE when the table contains any empty partitions. Do not use GETTRUNCATES on any partitioned table. Oracle GoldenGate DDL support can be used to capture truncates on tables that might include empty partitions.

Sybase Limitations

For Oracle GoldenGate to support TRUNCATE TABLE for Sybase, all table names must be unique across all schemas within a given database.

Default `IGNORETRUNCATES`

Syntax `GETTRUNCATES | IGNORETRUNCATES`

GETUPDATEAFTERS | IGNOREUPDATEAFTERS

Valid for Extract and Replicat

Use the GETUPDATEAFTERS and IGNOREUPDATEAFTERS parameters to control whether or not the after images of updated records are included in the records processed by Oracle GoldenGate. After images contain the results of the update.

The parameters are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements, until the other parameter is encountered.

Default `GETUPDATEAFTERS`

Syntax `GETUPDATEAFTERS | IGNOREUPDATEAFTERS`

GETUPDATEBEFORES | IGNOREUPDATEBEFORES

Valid for Extract and Replicat

Use the GETUPDATEBEFORES and IGNOREUPDATEBEFORES parameters to control whether or not the before images of updated columns are included in the records that are processed by Oracle GoldenGate. Before images contain column details that existed before a record was updated. Use the GETUPDATEBEFORES parameter as follows:

- in the Extract parameter file to extract before images from the data source.
- in the Replicat parameter file to include before images in a Replicat operation.

You can compare before images with after images to identify the net results of a transaction or perform other delta calculations. For example, if a BALANCE field is $100 before an update and $120 afterward, a comparison would show the difference of $20. You can use the column-conversion functions of Oracle GoldenGate to perform the comparisons and calculations.

You also can use GETUPDATEBEFORES to maintain a transaction-history table. For more information about performing delta calculations and using transaction history, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

- To reference before images in the parameter file, use the "before.<column>" syntax, for example:

```
COLMAP (previous = before.balance, [...])
```

GETUPDATEBEFORES is required when using the Conflict Detection and Resolution (CDR) feature for a DB2 database on any of the platforms that are supported by Oracle GoldenGate. For more information about CDR, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

The GETUPDATEBEFORES and IGNOREUPDATEBEFORES parameters are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements, until the other parameter is encountered.

Default IGNOREUPDATEBEFORES

Syntax `GETUPDATEBEFORES | IGNOREUPDATEBEFORES`

GETUPDATES | IGNOREUPDATES

Valid for Extract and Replicat

Use the GETUPDATES and IGNOREUPDATES parameters to control whether or not Oracle GoldenGate processes update operations. The parameters are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements, until the other parameter is encountered.

Default GETUPDATES

Syntax `GETUPDATES | IGNOREUPDATES`

GGSCHEMA

Valid for GLOBALS

Use the GGSCHEMA parameter to specify the name of the schema that contains the database objects that are owned by Oracle GoldenGate, such as those that support the synchronization of Oracle DDL by Oracle GoldenGate. This parameter is valid for the Oracle database.

Default None

Syntax `GGSCHEMA <schema_name>`

Argument	Description
`<schema_name>`	The name of the DDL schema.

GROUPTRANSOPS

Valid for Replicat

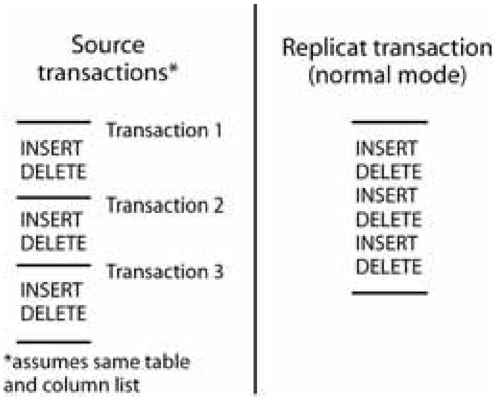
Use the GROUPTRANSOPS parameter to control the number of SQL operations that are contained in a Replicat transaction when operating in its normal mode. Increasing the number of operations in a Replicat transaction improves the performance of Oracle GoldenGate by:

- Reducing the number of transactions executed by Replicat.
- Reducing I/O activity to the checkpoint file and the checkpoint table, if used. Replicat issues a checkpoint whenever it applies a transaction to the target, in addition to its scheduled checkpoints.

Replicat accumulates operations from source transactions, in transaction order, and applies them as a group within one transaction on the target. GROUPTRANSOPS sets a minimum value rather than an absolute value, to avoid splitting apart source transactions. Replicat waits until it receives all operations from the last source transaction in the group before applying the target transaction.

For example, if transaction A contains 500 operations and transaction B contains 600, the Replicat transaction will contain all 1,100 operations even though GROUPTRANSOPS is set to the default of 1,000. Conversely, Replicat might apply a transaction before reaching the value set by GROUPTRANSOPS if there is no more data in the trail to process.

Figure 18 Replicat normal mode



Avoid setting GROUPTRANSOPS to an arbitrarily high number because the difference between source and target transaction boundaries can increase the latency of the target data.

Default 1000 operations

Syntax `GROUPTRANSOPS <min transaction count>`

Argument	Description
`<min transaction count>`	The minimum number of operations to be applied in a Replicat transaction. A value of 1 executes the operations within the same transaction boundaries as the source transaction. The value must be at least 1.

Example

```
GROUPTRANSOPS 2000
```

HANDLECOLLISIONS | NOHANDLECOLLISIONS

Valid for Replicat

Use the HANDLECOLLISIONS and NOHANDLECOLLISIONS parameters to control whether or not Replicat tries to resolve duplicate-record and missing-record errors when applying SQL on the target. These errors occur during an initial load, when data from source tables is being loaded to target tables while Oracle GoldenGate is replicating transactional changes that are being made to those tables. When Oracle GoldenGate applies the replicated changes after the load is finished, HANDLECOLLISIONS provides Replicat with error-handling logic for these collisions.

You can use HANDLECOLLISIONS and NOHANDLECOLLISIONS in the following ways:

- You can use either HANDLECOLLISIONS or NOHANDLECOLLISIONS at the root level of the parameter file to affect all MAP statements.
- You can use HANDLECOLLISIONS and NOHANDLECOLLISIONS as on-off switches for groups of tables to enable or disable error handling as needed. One remains in effect for all subsequent MAP statements until the other is encountered.
- You can use HANDLECOLLISIONS and NOHANDLECOLLISIONS within a MAP statement to enable and disable the functionality for a specific table.

Any of the preceding methods can be combined. Using this parameter within a MAP statement overrides other settings. Using it as a toggle overrides a global setting. For example, you could have a global NOHANDLECOLLISIONS setting, and then use HANDLECOLLISIONS within MAP statements to enable it only for certain tables.

How HANDLECOLLISIONS works

The following example explains how HANDLECOLLISIONS works:

- When Replicat encounters an update to a column that Oracle GoldenGate is using as a key, the handling is as follows:
  - If the row with the old key is not found in the target, the change record in the trail is converted to an insert.
  - If a row with the new key exists in the target, Replicat deletes the row that has the old key (it would not exist if the update had executed successfully), and then the row with the new key is updated as an overlay where the trail values replace the current values.

  This logic requires all of the columns in the table (not just the ones that changed) to be logged to the transaction log, either by default or by force, such as by using the COLS option of ADD TRANDATA for an Oracle database. (See also "Possible solutions to avoid missing column values".)

- When Replicat encounters a duplicate-record error, the static record that was applied by the initial load is overwritten by the change record in the trail. Overlaying the change is safer from an operational standpoint than ignoring the duplicate-record error.
- When Replicat encounters a missing-record error during an update or delete operation that does not affect a key column, the change record in the trail is discarded. These errors happen when a record is changed on the source system and then the record is deleted before the table data is extracted by the initial-load process. For example:

1. The application updates record A in source table1.
2. Extract extracts the update.
3. The application deletes record A in source table1.
4. Extract extracts the delete.
5. Oracle GoldenGate extracts initial-load data from source table1, without record A.
6. Oracle GoldenGate applies the initial load, without record A.
7. Replicat attempts to apply the update of record A.
8. The database returns a "record missing" error.
9. Replicat attempts to apply the delete of record A.
10. The database returns a "record missing" error.

Disable HANDLECOLLLIONS after the transactional changes captured during the initial load are applied to the target tables, so that Replicat does not automatically handle subsequent errors. Errors generated after initial synchronization indicate an abnormal condition and should be evaluated by someone who can determine how to resolve them. For example, a missing-record error could indicate that a record which exists on the source system was inadvertently deleted from the target system.

You can turn off HANDLECOLLISIONS in the following ways:

- Stop Replicat and remove HANDLECOLLISIONS from the Replicat parameter file (can cause target latency). Alternatively, you can edit the parameter file to add NOHANDLECOLLISIONS before the MAP statements for which you want to disable the error handling.
- While Replicat is running, run GGSCI and use the SEND REPLICAT command with the NOHANDLECOLLISIONS option for the tables that you want to affect (see page 61). If using SEND REPLICAT, make certain to remove HANDLECOLLISIONS from the parameter file or add a NOHANDLECOLLISIONS parameter before starting another Replicat run, so that HANDLECOLLISIONS does not activate again.

Possible solutions to avoid missing column values

When a database does not log all of the column values of a source table by default, there can be errors if the target table has NOT NULL constraints when Replicat attempts to convert a primary-key update to an insert. You can work around this scenario in the following ways:

- Use the NOCOMPRESSUPDATES parameter in the Extract parameter file to send all of the columns of the table to the trail, and configure the database to log all column values. By default, Extract only writes the primary key and the columns that changed to the trail. This is the safest method, because it writes the current values at the time when the operation is performed and eliminates the need for fetching.
- Use the FETCHOPTIONS parameter with the FETCHPKUPDATECOLS option in the Extract parameter file. This configuration causes Extract to fetch unavailable columns when a key column is updated on the source. A fetch is the *current* value, not necessarily the value at the time of a particular update, so there can be data integrity issues. See page 198 for more information and additional fetch options to handle unsuccessful fetches.
- To avoid fetches, use HANDLECOLLISIONS with _ALLOWPKMISSINGROWCOLLISIONS to skip the update instead of converting it to an insert. This configuration can also cause data integrity issues under certain conditions. For more information, see “Preventing conversion of key updates to inserts”.

Preventing conversion of key updates to inserts

In some cases, it is not appropriate to convert an operation that updates a key column to an insert if the target row does not exist. In these cases, you can use the _ALLOWPKMISSINGROWCOLLISIONS option to force Replicat to *skip* the operation, instead of applying it as an insert.

The following example illustrates such a case. This scenario performs an instantiation of Oracle GoldenGate replication, using the default HANDLECOLLISIONS logic, to show what happens if column values are missing when Replicat tries to convert the update to an insert.

Source and target tables named “s”:

```
f1  f2                   f3  f4
1   10-01-2011 11:30:45  1   1
2   10-02-2011 14:15:20  2   2
3   10-03-2011 15:12:55  3   3
```

- All columns are NOT NULL.
- f1 is the primary key.
- f2 is a date field that automatically updates whenever the record is changed.
- KEYCOLS is used in the parameter files to instruct Oracle GoldenGate to use f1 and f2 as the key.
- ADD TRANDATA was issued accordingly, to log column f2. (Column f1 is automatically logged because it is a primary key).

DML sequence of events:

1. Start Extract to capture ongoing transactions.
2. UPDATE the table as follows:

   ```
   update s set f3=3 where f1=2;
   ```

   In this operation, column f2 updates automatically with the current date and time. Oracle GoldenGate considers this to be a key update.

The row now looks like this:

```
2   10-20-2011 08:01:32  3   3
```

3. DELETE the same row.

   ```
   delete s where f1=2
   ```

   Now the table contains the following rows:

   ```
   f1  f2                   f3  f4
   1   10-01-2011 11:30:45  1   1
   3   10-03-2011 15:12:55  3   3
   ```

4. Perform an export/import of the source data to the target, using HANDLECOLLISIONS to handle missing or duplicate rows.
5. The replicated update (update s set f3=3 where f1=2) is the first operation to be applied from the trail by Replicat. It fails because the row was deleted from the source before the import/export was performed.
6. Replicat converts the UPDATE to an INSERT according to HANDLECOLLISIONS logic for operations that update a key column (the f2 date-time column).
7. In a case where all of the column values are available in the trail, the new insert succeeds. Moreover, it does not cause inconsistency, even though the row was deleted on the source, because the replicated delete (delete s where f1=2) removes it again. However, in this example, there are two problems:
   - Only columns f1 and f2, plus the changed value of f3, are logged. The value for f4 is not logged and the value is not available for the insert.
   - All columns have a NOT NULL constraint.

The missing f4 value causes the insert to fail. By using _ALLOWPKMISSINGROWCOLLISIONS, Replicat skips the UPDATE instead of converting it to an insert. This causes the subsequent DELETE to fail because the row does not exist, so Replicat skips the DELETE record as part of the default HANDLECOLLISIONS logic. The data now is consistent with that of the source.

Messages from _ALLOWPKMISSINGROWCOLLISIONS

Because of the risk of data loss, a warning is issued when _ALLOWPKMISSINGROWCOLLISIONS is used. The warning is similar to the following text:

```
Using _ALLOWPKMISSINGROWCOLLISIONS may cause data corruption under
certain conditions.
```

A warning message also is issued for when an UPDATE to a key does not contain a full after image for conversion to an insert:

```
A complete after image is not available in SOURCE.x, at RBA 123, in file
.\dirdat\aa000000, while inserting a row into TARGET.x due to a missing
target row for a key update operation. NOCOMPRESSUPDATES or FETCHOPTIONS
FETCHPKUPDATECOLS may be specified in the EXTRACT parameter file to
include a complete image for key update operations.
```

Default None

Syntax `HANDLECOLLISIONS | NOHANDLECOLLISIONS [_ALLOWPKMISSINGROWCOLLISIONS]`

Argument	Description
`_ALLOWPKMISSINGROWCOLLISIONS`	Use HANDLECOLLISIONS with _ALLOWPKMISSINGROWCOLLISIONS to skip primary-key UPDATE operations if the corresponding target row does not exist. Note: Skipping operations can cause data corruption. See the description in this topic.

Example 1 The following enables HANDLECOLLISIONS for all MAP statements in the parameter file.

```
HANDLECOLLISIONS
MAP hr.emp, TARGET hr.emp;
MAP hr.job_hist, TARGET hr.job_hist;
MAP hr.dep, TARGET hr.dep;
MAP hr.country, TARGET hr.country;
```

Example 2 The following enables HANDLECOLLISIONS for some MAP statements while disabling it for others.

```
HANDLECOLLISIONS
MAP hr.emp, TARGET hr.emp;
MAP hr.job_hist, TARGET hr.job_hist;
NOHANDLECOLLISIONS
MAP hr.dep, TARGET hr.dep;
MAP hr.country, TARGET hr.country;
```

Example 3 The following shows a combination of global and MAP-level use. The MAP specification overrides the global specification for the specified tables.

```
HANDLECOLLISIONS
MAP hr.emp, TARGET hr.emp;
MAP hr.job_hist, TARGET hr.job_hist;
MAP hr.dep, TARGET hr.dep, NOHANDLECOLLISIONS;
MAP hr.country, TARGET hr.country, NOHANDLECOLLISIONS;
```

HANDLETPKUPDATE

Valid for Replicat

Use the HANDLETPKUPDATE parameter to prevent constraint errors when an update to a primary key results in a transient duplicate. This is an Oracle parameter and is required if the target database is any version earlier than Oracle version 11.2.0.2. For target Oracle databases that are version 11.2.0.2 or later, transient primary-key duplicates are handled automatically without requiring HANDLETPKUPDATE.

A transient primary-key duplicate occurs when an update affects the primary keys of multiple rows in a transaction. This kind of statement typically uses a “SET x = *x*+*n*” formula or other manipulation that shifts the values so that a new value is the same as an existing one.

The following example illustrates a sequence of value changes that can cause this condition. The example assumes table “ITEM” where the primary key column is named

“CODE” and the current key values for the rows in the table are 1, 2, and 3.

```
update item set code = 2 where code = 1;
update item set code = 3 where code = 2;
update item set code = 4 where code = 3;
```

In this example, when the first update is applied to the target, there is an error because the primary key value of 2 already exists in the target. The target transaction returns constraint violation errors. By default, Replicat does not detect or handle these violations and abends.

When using HANDLETPKUPDATE, create the constraints as DEFERRABLE INITIALLY IMMEDIATE on the target tables. If the target constraints cannot be DEFERRABLE, Replicat handles the errors according to existing rules specified with the HANDLECOLLISIONS and REPERROR parameters, or else it abends.

Default Abend on transient primary key updates

Syntax `HANDLETPKUPDATE`

INCLUDE

Valid for Extract and Replicat

Use the INCLUDE parameter to include a macro library in a parameter file.

Default None

Syntax `INCLUDE <path name>`

Argument	Description
`<path name>`	The relative or full path to library file.

Example The following example includes macro library mdatelib.mac.

```
INCLUDE /ggs/dirprm/mdatelib.mac
```

INSERTAPPEND | NOINSERTAPPEND

Valid for Replicat

Use the INSERTAPPEND and NOINSERTAPPEND parameters to control whether or not Replicat uses an APPEND hint when it applies INSERT operations to Oracle target tables. These parameters are valid only for Oracle databases.

These parameters can be used in two ways: When used as standalone parameters at the root of the parameter file, one remains in effect for all subsequent TABLE or MAP statements, until the other is encountered. When used within a MAP statement, they override any standalone INSERTAPPEND or NOINSERTAPPEND entry that precedes the MAP statement.

INSERTAPPEND causes Replicat to use the APPEND_VALUES hint when it applies INSERT operations to Oracle target tables. It is appropriate for use as a performance improvement when the replicated transactions are large and contain multiple inserts into the same table. If the transactions are small, using INSERTAPPEND can cause a performance decrease. For more information about when APPEND hints should be used, consult the Oracle documentation.

The BATCHSQL parameter must be used when using INSERTAPPEND. Replicat will abend if BATCHSQL is not used.

For MAP syntax, see page 232.

Default NOINSERTAPPEND

Syntax `INSERTAPPEND | NOINSERTAPPEND`

Example The following excerpt from a Replicat parameter file shows how INSERTAPPEND is used for all tables in the fin schema, except for the inventory table.

```
BATCHSQL
INSERTAPPEND
MAP fin.*, TARGET fin.*;
MAPEXCLUDE fin.inventory;
NOINSERTAPPEND
MAP fin.inventory, TARGET fin.inventory;
```

INSERTALLRECORDS

Valid for Replicat

Use the INSERTALLRECORDS parameter to keep a record of all operations made to a target record, instead of maintaining just the current version. INSERTALLRECORDS causes Replicat to insert every change operation made to a record as a new record in the database. The initial insert and subsequent updates and deletes are maintained as point-in-time snapshots.

Combining historical data with special transaction information provides a way to create a more useful target reporting database. For more information about maintaining a transaction-history table, see the *Oracle GoldenGate Windows and UNIX Administrator's Guide*.

This parameter also can be used within a MAP statement. See page 232.

Default None

Syntax `INSERTALLRECORDS`

INSERTDELETES | NOINSERTDELETES

Valid for Replicat

Use the INSERTDELETES and NOINSERTDELETES parameters to control whether or not Oracle GoldenGate converts source delete operations to insert operations on the target database. The parameters are table-specific. One parameter remains in effect for all subsequent MAP statements, until the other parameter is encountered.

When using INSERTDELETES, use the NOCOMPRESSDELETES parameter so that Extract does not compress deletes.

Default NOINSERTDELETES

Syntax `INSERTDELETES | NOINSERTDELETES`

INSERTMISSINGUPDATES | NOINSERTMISSINGUPDATES

Valid for Replicat

Use the INSERTMISSINGUPDATES and NOINSERTMISSINGUPDATES parameters to control whether or not Oracle GoldenGate inserts a record based on the source record when the target record does not exist.

INSERTMISSINGUPDATES inserts the missing update but should only be used when the source database uses non-compressed updates (meaning that all column values are logged). It can work with a database that uses compressed updates if the target database allows NULL to be used for the missing column values.

When the default of NOINSERTMISSINGUPDATES is in effect, a missing record causes an error, and the transaction may abend depending on REPERROR settings.

The INSERTMISSINGUPDATES and NOINSERTMISSINGUPDATES parameters are table-specific. One parameter remains in effect for all subsequent MAP statements, until the other parameter is encountered.

Default `NOINSERTMISSINGUPDATES`

Syntax `INSERTMISSINGUPDATES | NOINSERTMISSINGUPDATES`

INSERTUPDATES | NOINSERTUPDATES

Valid for Replicat

Use the INSERTUPDATES and NOINSERTUPDATES parameters to control whether or not Oracle GoldenGate converts uncompressed update operations to insert operations. For updates to be uncompressed, the database must log all column values either by default or by means of supplemental logging.

The parameters are table-specific. One parameter remains in effect for all subsequent MAP statements, until the other parameter is encountered.

To ensure that updates are not compressed by Extract when using INSERTUPDATES, use the NOCOMPRESSUPDATES parameter.

Default `NOINSERTUPDATES`

Syntax `INSERTUPDATES | NOINSERTUPDATES`

LAGCRITICAL

Valid for Manager

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.

Default Do not report lag information

Syntax

```
LAGCRITICALSECONDS <seconds> |
LAGCRITICALMINUTES <minutes> |
LAGCRITICALHOURS <hours>
```

Argument	Description
`<seconds>`	Lag threshold, in seconds.
`<minutes>`	Lag threshold, in minutes.
`<hours>`	Lag threshold, in hours.

Example

```
LAGCRITICALSECONDS 60
```

LAGINFO

Valid for Manager

Use the LAGINFOSECONDS, LAGINFOMINUTES, or LAGINFOHOURS parameter to specify a lag threshold; if lag exceeds the specified value, Oracle GoldenGate reports lag information to the error log. If the lag exceeds the value specified with the LAGCRITICAL parameter, Manager reports the lag as critical; otherwise, it reports the lag as an informational message. A value of zero (0) forces a message at the frequency specified with the LAGREPORTMINUTES or LAGREPORTHOURS parameter.

Default Do not report lag information

Syntax

```
LAGINFOSECONDS <seconds> |
LAGINFOMINUTES <minutes> |
LAGINFOHOURS <hours>
```

Argument	Description
`<seconds>`	The threshold, in seconds, after which Oracle GoldenGate reports lag information.
`<minutes>`	The threshold, in minutes, after which Oracle GoldenGate reports lag information.
`<hours>`	The threshold, in hours, after which Oracle GoldenGate reports lag information.

Example In this example, Oracle GoldenGate reports lag when it exceeds one hour.

```
LAGINFOHOURS 1
```

LAGREPORT

Valid for Manager

Use the LAGREPORTMINUTES or LAGREPORTHOURS parameter to specify the interval at which Manager checks for Extract and Replicat lag.

Default None

Syntax
```
LAGREPORTMINUTES <minutes> | LAGREPORTHOURS <hours>
```

Argument	Description
`<minutes>`	The frequency, in minutes, to check for lag.
`<hours>`	The frequency, in hours, to check for lag.

Example
```
LAGREPORTHOURS 1
```

LIST | NOLIST

Valid for Extract and Replicat

Use the LIST and NOLIST parameters to control whether or not the macros of a macro library are listed in the report file. Listing can be turned on and off by placing the LIST and NOLIST parameters within the parameter file or within the macro library file. Using NOLIST reduces the size of the report file.

Default `LIST`

Syntax `LIST | NOLIST`

Example In the following example, NOLIST excludes the macros in the hugelib macro library from being listed in the report. Using LIST after the INCLUDE statement restores normal listing for subsequent macros.

```
NOLIST
INCLUDE /ggs/hugelib.mac
LIST
```

LOBMEMORY

Valid for Extract and Replicat for DB2 on z/OS and NonStop SQL/MX

Use the LOBMEMORY parameter to control the amount of memory and temporary disk space available for caching transactions that contain LOBs. Because Oracle GoldenGate applies only committed transactions to the target database, it requires sufficient system memory to store LOB data until either a commit or rollback indicator is received.

This parameter is for use with a DB2 database on z/OS and for a NonStop SQL/MX database. For all other databases, use the CACHEMGR parameter.

About memory management with LOBMEMORY

LOBMEMORY enables you to tune the cache size of Oracle GoldenGate for LOB transactions and define a temporary location on disk for storing data that exceeds the size of the cache. Options are available for defining the total cache size, the per-transaction memory size, the initial and incremental memory allocation, and disk storage space.

LOB transactions are added to the memory pool specified by RAM, and each is flushed to disk when TRANSRAM is reached. An initial amount of memory is allocated to each transaction based on INITTRANSRAM and is increased by the amount specified by RAMINCREMENT as needed, up to the maximum set with TRANSRAM. Consequently, the value for TRANSRAM should be evenly divisible by the sum of (INITTRANSRAM + RAMINCREMENT).

Default See option defaults

Syntax

```
LOBMEMORY
[RAM <size>]
[TRANSRAM <size>]
[TRANSALLSOURCES <size>]
[INITTRANSRAM <size>]
[RAMINCREMENT <size>]
[DIRECTORY (<directory name>, <max dir size>, <max file size>)]
```

Options	Description
`RAM <size>`	Specifies the total amount of memory to use for all cached LOB transactions. The default is 200 megabytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k
`TRANSRAM <size>`	Specifies the total amount of memory to use for a single LOB transaction. The default is 50 megabytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k TRANSRAM should be evenly divisible by both INITTRANSRAM and RAMINCREMENT for optimal results.
`TRANSALLSOURCES <size>`	Specifies the total amount of memory and disk space to use for a single LOB transaction. The default is 50% of total available memory (memory and disk). The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k
`INITTRANSRAM <size>`	Specifies the initial amount of memory to allocate for a LOB transaction. The default is 500 kilobytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k
`RAMINCREMENT <size>`	Specifies the amount of memory to increment when a LOB transaction requires more memory. The default is 500 kilobytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m |

Options	Description
`DIRECTORY (<directory name>, <max dir size>, <max file size>)`	Specifies temporary disk storage for LOB transaction data when its size exceeds the maximum specified with TRANSRAM. You can specify DIRECTORY more than once. ◆ <directory> is the fully qualified name of a directory. The default is the dirtmp sub-directory of the Oracle GoldenGate directory. ◆ <max dir size> is the maximum size of all files in the directory. The default is 2 gigabytes. If the space specified is not available, then 75% of available disk space is used. ◆ <max file size> is the maximum size of each file in the directory. The default is 200 megabytes. Values can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k The directory size and file size must be greater than the size of the memory specified with RAM. The file names use the following format. <group>_blob_00001.mem or... <PID>_blob_00001.mem A group name is used for online processes. A system process ID number (PID) is used for one-time runs specified with the SPECIALRUN parameter. The format for a threaded Extract is similar to the following, depending on the database. <group>_<thread #>_00001.mem

Example 1 The following example allows per-transaction memory to be incremented ten times before data is flushed to disk, once for the initial allocation specified with INITTRANSRAM and then nine more times as permitted by RAMINCREMENT.

```
LOBMEMORY DIRECTORY (c:\test\dirtmp, 3000000000,
300000000), RAM 8000K, TRANSRAM 1000K, INITTRANSRAM 100K,
RAMINCREMENT 100K
```

Example 2 The following is the same as the preceding example, but with the addition of a second directory.

```
LOBMEMORY DIRECTORY (c:\test\dirtmp, 3000000000,
300000000), DIRECTORY (c:\test\dirtmp2, 1000000000,
5000000), RAM 8000K, TRANSRAM 1000K, INITTRANSRAM 100K,
RAMINCREMENT 100K
```

NOTE In the previous examples, the parameter specification spans multiple lines because of space constraints. In an actual parameter file, multi-line parameter specifications must contain an ampersand (&) at the end of each line.

MACRO

Valid for Extract and Replicat

Use the MACRO parameter to create an Oracle GoldenGate macro. For information about using macros, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Default None

Syntax The following must be used in the order shown:

```
MACRO <macrochar><macro name>
PARAMS (<macrochar><paramname> [, ...])
BEGIN
<macro body>
END;
```

Argument	Description

`<macrochar>`	The macro character. Macro and parameter names must begin with a macro character. Anything in the parameter file that begins with the macro character is assumed to be either a macro or a macro parameter.
	The default macro character is the pound (#) character, as in the following examples:
	`MACRO #macro1`
	`PARAMS (#param1, #param2)`
	You can change the macro character with the MACROCHAR parameter.

`<macro name>`	The name of the macro. Macro names must be one word with alphanumeric characters (underscores are allowed) and are not case-sensitive. Each macro name in a parameter file must be unique. Do not use quotes, or else the macro name will be treated as text and ignored.
`<paramname>`	Describes parameters to the macro. Parameter names are not case-sensitive. Do not use quotes, or else the parameter name will be treated as text and ignored. A parameters clause is optional. The maximum size of a PARAMS statement is 9999 bytes and can contain no more than 99 parameters. 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 1 shows a macro that takes parameters. You can create macros without parameters, such as a macro for frequently used commands. See Example 2. You can use other macros as parameters. See Example 3.
`BEGIN`	Begins the macro body. Must be specified before the macro body.

Argument	Description

`<macro body>`	The body of the macro. The maximum size of a macro body is 99999 bytes. 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`	Concludes the macro definition.

Example 1 The following example defines a macro that takes parameters.

```
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;
```

Example 2 The following example defines a macro that does not require parameters.

```
MACRO #option_defaults
BEGIN
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
END;
```

Example 3 The following example defines a macro named #assign_date that calls another macro named #make_date.

```
MACRO #assign_date
PARAMS (#target_col, #year, #month, #day)
BEGIN
#target_col = #make_date (#year, #month, #day)
END;
```

MACROCHAR

Valid for Extract and Replicat

Use the MACROCHAR parameter to change the macro character to something other than the # character. Anything in the parameter file that begins with the specified macro character is assumed to be either a macro or a macro parameter.

You might need to change the macro character when, for example, table names include the # character.

The MACROCHAR can only be specified once, and it must precede the first macro statement.

Default # (pound symbol)

Syntax `MACROCHAR <character>`

Argument	Description
`<character>`	The character to be used as the macro character. Valid macro and parameter characters are alphanumeric and can include the underscore character (_).

Example In the following example, $ is defined as the macro character.

```
MACROCHAR $
MACRO $mymac
PARAMS ($p1)
BEGIN
col = $p1
END;
```

MAP for Extract

Valid for Extract

Use the MAP parameter for Extract when it is operating in classic capture mode and you are using the ALTID component to map an object ID to an object name. ALTID specifies the correct object ID if Extract is capturing from Oracle transaction logs that were generated by a database other than the one to which it is connected. This configuration is required when Oracle GoldenGate processes are not permitted to connect directly to the production (source) database, but must capture the production transactions.

In such a case, Extract connects to a live standby or other facsimile database, but reads transaction logs that are sent from the production database. By querying the catalog of the alternate database, Extract can get the metadata that it needs to expand the transaction data into valid SQL statements, but it cannot use the object ID from this query, because the local object ID for a given table is different from the one for that table in the production database (and transaction log). You must manually map each table name to the production (source) object ID by using a MAP statement with ALTID.

To use MAP with ALTID

- Create one MAP statement with ALTID for each table that you want to capture. Wildcarded table names are not allowed for a MAP parameter that contains ALTID.
- To specify other processing for a table, such as data filtering or manipulation, you must also create a TABLE statement. Otherwise, TABLE is not required.
- Use a regular Replicat MAP statement in the Replicat parameter file, as usual. MAP for Extract does not substitute for MAP for Replicat, which is required to map source tables to target tables.
- DDL capture and replication is not supported when using ALTID.

Default None

Syntax `MAP <table spec>, ALTID <object ID> [, object ID>]`

Component	Description
`<object ID>`	The object ID of the table as it exists in the production (source) database. If a table is partitioned, you can list the object IDs of the partitions that you want to replicate, separating each with a comma.

Example `map QASOURCE.T2, altid 75740;`

Example `map QASOURCE.T_P1, altid 75257,75258;`

MAP for Replicat

Valid for Replicat

Use the MAP parameter for Replicat to establish a relationship between one or more source and target objects. All target objects that you are synchronizing with Oracle GoldenGate must be mapped to source objects with this parameter.

NOTE To capture the source objects that you are mapping with MAP, use a TABLE parameter statement in the Extract parameter file on the source system.

Limitations of use

For tables, you can use all of the MAP options. You can:

- Select and filter rows of tables
- Map columns of tables
- Transform data
- Specify key columns
- Execute stored procedures and queries
- Specify exceptions and error handling
- Apply all operations on a table as inserts
- Pass a parameter to a user exit

For non-table objects, use MAP only to map the source object to its target object and to handle processing errors with the EXCEPTIONSONLY and REPERROR options. Do not use any of the other MAP options for those objects.

Default None

Syntax

```
MAP <table spec>, TARGET <table spec>
[, COLMAP (<column mapping expression>)]
[, DEF <definitions template>]
[, COMPARECOLS (
   {ON UPDATE | ON DELETE}
   {ALL |
   KEY |
   KEYINCLUDING (<col>[,...]) |
   ALLEXCLUDING (<col>[,...]) }
   [,...]
   )]
[, EVENTACTIONS (<action>)]
[, EXCEPTIONSONLY]
[, EXITPARAM “<parameter string>”]
[, FILTER (<filter specification>)]
[, HANDLECOLLISIONS | NOHANDLECOLLISIONS]
[, INSERTALLRECORDS]
[, INSERTAPPEND | NOINSERTAPPEND]
[, KEYCOLS (<column specification>)]
[, MAPEXCEPTION (TARGET <object spec> [, <exception_MAP_options>])]
[, REPERROR (<error> , <response>)]
[, RESOLVECONFLICT (
   {INSERTROWEXISTS |
   UPDATEROWEXISTS |
   UPDATEROWMISSING |
   DELETEROWEXISTS |
   DELETEROWMISSING}
   ({DEFAULT | <resolution name>},
   {USEMAX (<res_col>) |
   USEMIN (<res_col>) |
   USEDELTA |
   DISCARD |
   OVERWRITE |
   IGNORE})
[, COLS (<col>[,...])]
)]
[RESOLVECONFLICT (, ...)]
[, SQLEXEC (<SQL specification>)]
[, TARGETDEF <definitions template>]
[, TRIMSPACES | NOTRIMSPACES]
[, TRIMVARSPACES | NOTRIMVARSPACES]
[, WHERE (<where clause>)]
;
```

Table 36 Summary of MAP syntax components

Component	Description
MAP	Specifies the source object.
TARGET	Specifies the target object.

Table 36 Summary of MAP syntax components (continued)

Component	Description
DEF	Specifies a source-definitions template.
TARGETDEF	Specifies a target-definitions template.
COLMAP	Maps records between different source and target columns.
COMPARECOLS	Specifies columns to use for conflict detection and resolution.
EVENTACTIONS	Triggers an action based on a record that satisfies a specified filter rule.
EXCEPTIONSONLY	Specifies that the MAP statement is an exceptions MAP statement.
EXITPARAM	Passes a parameter in the form of a literal string to a user exit.
FILTER	Selects records based on a numeric operator. FILTER provides more flexibility than WHERE.
HANDLECOLLISIONS | NOHANDLECOLLISIONS	Reconciles the results of changes made to the target table by an initial load process with those applied by a change-synchronization group.
INSERTALLRECORDS	Applies all row changes as inserts.
INSERTAPPEND | NOINSERTAPPEND	Controls whether or not Replicat uses an Oracle APPEND hint for INSERT statements.
KEYCOLS	Designates columns that uniquely identify rows.
MAPEXCEPTION	Specifies that the MAP statement contains exceptions handling for wildcarded tables.
REPERROR	Controls how Replicat responds to errors when executing the MAP statement.
RESOLVECONFLICT	Specifies rules for conflict resolution.
SQLEXEC	Executes stored procedures and queries.
TRIMSPACES | NOTRIMSPACES	Controls whether trailing spaces are trimmed or not when mapping CHAR to VARCHAR columns.
TRIMVARSPACES | NOTRIMVARSPACES	Controls whether trailing spaces are trimmed or not when mapping VARCHAR to VARCHAR columns.
WHERE	Selects records based on conditional operators.
;	Terminates the MAP statement and is required.

Specifying object names in the MAP and TARGET clauses

To specify the object names in the MAP and TARGET clauses of the MAP statement, see "Getting started with the Oracle GoldenGate process interfaces" in the Oracle GoldenGate *Administration Guide.*

Using COLMAP

Use COLMAP to explicitly map source columns to target columns that have different names or to specify default column mapping when source and target names are identical. COLMAP provides instructions for selecting, translating, and moving column data.

NOTE To create *global* rules for column mapping across all tables in subsequent MAP statements, use the COLMATCH parameter.

Supporting case and special characters in column names

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

Generating data definitions

When using COLMAP for source and target tables that are not identical in structure, you must generate data definitions for the source tables, transfer them to the target, and use the SOURCEDEFS parameter to identify the definitions file.

For source and target structures to be considered identical, they must contain identical column names (including case, if applicable) and data types, and the columns must be in the same order in each table. In addition, both tables must have the same column length semantics for character columns (bytes versus characters).

If the tables have identical structures, and you are using COLMAP for other functions such as conversion, a source definitions file is not needed. You can use the ASSUMETARGETDEFS parameter instead.

For more information, see:

- SOURCEDEFS on page 330
- ASSUMETARGETDEFS on page 127
- "Creating a data-definitions file" in the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Mapping a value to a key column

If using COLMAP to map a value to a key column (which causes the operation to become a primary key update), the WHERE clause that Oracle GoldenGate uses to locate the target row will not use the correct before image of the key column. Instead, it will use the after image. This will cause errors if you are using any functions based on that key column, such as a SQLEXEC statement.

The following example illustrates what happens. In this example, there are the following objects:

- Source table TCUSTMER1
- Target table TCUSTMER2
- Column layout, both tables:

  Column 1 = Cust

  Column 2 = Name

  Column 3 = City

  Column 4 = State
- Primary key is Cust, Name, and City columns.

This is the SQLEXEC statement in the MAP statement:

```
SQLEXEC (id mytest, query "select city from TCUSTMER1 WHERE state = 'CA'",
noparams, ERROR RAISE),
```

This is the COLMAP statement in the MAP statement:

```
COLMAP ( usedefaults, city = mytest.city );
```

This is the sequence of events:

1. INSERT statement inserts the following:

   ```
   INSERT into TCUSTMER1 values (Cust = ‘1234’, Name = ‘Ace’, City = ‘SF’,
   State = ‘CA’);
   Commit;
   ```

   This succeeds, because the SQLEXEC query will return mytest.city = ‘SF, so the target table also will have a value of SF for City and CA for State.
2. UPDATE statement changes City from SF to LA on the source. This does not succeed on the target. The SQLEXEC query looks up the City column in TCUSTMER1 and returns a value of LA. Based on the COLMAP clause, the before and after versions of City both are now LA. This leads to SQL error 1403 when executing the target WHERE clause, because a value of LA does not exist for the City column in the target table.

Using default column mapping

For any corresponding source and target columns whose names are identical, you can use default mapping instead of using an explicit mapping statement. Default mapping causes Oracle GoldenGate to map those columns automatically. Data translation, if any, is automatic.

To use default mapping, use the USEDEFAULTS option. Default mapping is only enabled for columns that are not mapped already with an explicit mapping statement.

By default SQL Server and Sybase columns are compared with case-sensitivity taken into account. For all other databases, the column names are changed to upper case for name comparison. To support case-sensitive column names for those databases, use the USEANSISQLQUOTES parameter in the GLOBALS file. This applies SQL-92 rules, which require column names to be enclosed within double quotes and literals to be enclosed within single quotes.

Where case is recognized, USEDEFAULTS supports case sensitivity in the following manner:

- If a source column is found whose name and case exactly matches that 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 the target column, regardless of case.

For example, the following are source and target tables that contain case-sensitive columns.

Columns of source table USER1.SM01:

id

owner

created

changed

creator

modifiedBy

comment

COMMENT

Columns of target table USER3.SM01:

ID

owner

id

Creator

comment

ModifiedBy

creationDate

alterationDate

Comment

COMMENT

The following column map for these tables contains both explicit and default column mappings:

```
MAP USER1.SM01, TARGET USER3.SM01,
COLMAP (USEDEFAULTS,
   ID = id,
   creationDate = created,
   alterationDate = changed,
   );
```

The following is the result of this map. For default mapping, case-sensitivity is observed when applicable, but otherwise just the names are matched. Two target columns are not mapped because they were not explicitly mapped and no default map could be established.

Mapping type	Mapping result

Explicit mapping	`ID = id,`
	`creationDate = created,`
	`alterationDate = changed`

Default mapping	`owner = owner,`
	`comment = comment,`
	`COMMENT = COMMENT,`
	`Creator = creator,`
	`ModifiedBy = modifiedby`

Target columns not mapped	`id, Comment`

Syntax

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

Component	Description
`<table spec>`	The source or target table.

`<target column> =`
`<source expression>`
	Explicitly defines a source-target column map.
	<target column> is the name of the target column. For guidelines for specifying object names, see the Oracle GoldenGate *Administration Guide.*
	<source expression> can be any of the following:
	◆ The name of a source column, such as ORD_DATE
	◆ A numeric constant, such as 123
	◆ A string constant within quotes, such as “ABCD”
	◆ An expression using an Oracle GoldenGate column-conversion function, such as @STREXT (COL1, 1, 3). For descriptions of the column-conversion functions, see Chapter 4.

`BINARYINPUT`	Use BINARYINPUT when the target column is defined as a binary data type, such as RAW or BLOB, but the source input contains binary zeros in the middle of the data. Use BINARYINPUT when replicating a full Enscribe record defined as a single column into a target column. The source input is handled as binary input, and replacement of data values is suppressed.

Component	Description
`USEDEFAULTS`	Automatically maps source and target columns that have the same name if they were not specified in an explicit column map. Use an explicit map or USEDEFAULTS, but not both for the same set of columns. See “Using default column mapping” on page 236 for more information. Specify USEDEFAULTS before explicit column maps.

Example 1

```
MAP ggs.tran, TARGET ggs.tran2, COLMAP (loc2 = loc, type2 = type);
```

Example 2

```
MAP ggs.tran, TARGET ggs.tran2, COLMAP ( EUROVAL = "\u20ac0" );
```

Example 3

```
MAP ggs.tran, TARGET ggs.tran2, COLMAP (SECTION = @STRCAT("\u00a7", SECTION
));
```

Using COMPARECOLS

Use COMPARECOLS to specify the columns that Replicat uses to detect and resolve update or delete conflicts when configured with the RESOLVECONFLICT option of MAP in a multi-master configuration. A conflict is a mismatch between the before image of a record in the trail and the currect data in the target table.

The before image must be available in the trail record by means of the GETBEFORECOLS parameter in the Extract TABLE statement. The specified columns must exist in the target database and also be part of the Replicat configuration (satisfy the TARGET specification with or without a COLMAP clause).

To specify conflict resolution routines, use the RESOLVECONFLICT option of MAP. COMPARECOLS and RESOLVECONFLICT can be in any order in the MAP statement. For detailed information about conflicts and conflict resolution, see the Oracle GoldenGate *Windows and UNIX Administrator’s Guide.*

Syntax

```
MAP <source table> , TARGET <target table>,
COMPARECOLS(
{ON UPDATE | ON DELETE}
{ALL |
KEY |
KEYINCLUDING (<col>[,...]) |
ALLEXCLUDING (<col>[,...]) }
[,...]
)
```

Argument	Description
`{ON UPDATE	ON DELETE}`	Specifies whether the before image of the specified columns should be compared for updates or deletes. You can use ON UPDATE only, ON DELETE only, or both. If using both, specify them within the same COMPARECOLS clause. See the example for how to use both.

Argument	Description

`{ALL	KEY |`
	`KEYINCLUDING (<col>[,...]) |`
	`ALLEXCLUDING (<col>[,...])}`
		Specifies the columns for which a before image is captured.
		ALL: Compares using all columns in the target table. An error is generated if any corresponding before images are not available in the trail. Using ALL imposes the highest processing load for Replicat, but allows conflict-detection comparisons to be performed using all columns for maximum accuracy.
		KEY: compares only the primary key columns. This is the fastest option, but does not permit the most accurate conflict detection, because keys can match but non-key columns could be different.
		KEYINCLUDING: compares the primary key columns and the specified column or columns. This is a reasonable compromise between speed and detection accuracy.
		ALLEXCLUDING: compares all columns except the specified columns. For tables with numerous columns, ALLEXCLUDING may be more efficient than KEYINCLUDING. Do *not* exclude key columns.

Example 1 In the following example, the key columns plus the name, address, and salary columns are compared for conflicts.

```
MAP src, TARGET tgt
COMPARECOLS (
ON UPDATE KEYINCLUDING (name, address,  salary),
ON DELETE KEYINCLUDING (name, address, salary));
```

Example 2 In the following example, the comment column is ignored and all other columns are compared for conflicts.

```
MAP src, TARGET tgt
COMPARECOLS (ON UPDATE ALLEXCLUDING (comment))
```

Using DEF

Use DEF to specify a source-definitions template. The definitions template is created based on the definitions of a specific source table when DEFGEN is run for that object. Once the template is created, new source tables that have identical definitions to that table can be added without having to run DEFGEN for them, and without having to stop and start Replicat. The definitions in the template specified with DEF will be used for definitions lookups. For more information about DEFGEN, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

```
MAP <table spec>, TARGET <table spec>, DEF <definitions template>;
```

Argument	Description
`<definitions template>`	The name of a definitions template that was specified with the DEF option of TABLE in the DEFGEN parameter file. The definitions contained in the template must be identical to the definitions of the table in this MAP statement.

Example

```
MAP acct.cust*, TARGET acct.cust*, DEF custdef;
```

Using EVENTACTIONS

Use EVENTACTIONS to cause the Replicat process to take a defined action based on a record in the trail, known as the *event record*, that qualifies for a specific filter rule. You can use this system to customize processing based on database events. For example, you can suspend a process to perform a transformation or report statistics.

To use the event marker system to trigger actions that do not require data to be applied to target tables, you can use the Replicat TABLE parameter with filtering options that support EVENTACTIONS. See page 376.

WARNING EVENTACTIONS is not supported if the source database is Teradata and Extract is configured in maximum performance mode.

Many, but not all, of the EVENTACTIONS options apply both to TABLE (for Extract) and to MAP (for Replicat), so all of the options for both processes are shown here. Exceptions are noted.

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
    - SUSPEND
    - ABORT
    - CHECKPOINT AFTER
    - FORCESTOP
    - STOP

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 | SUSPEND | ABORT | FORCESTOP]
[IGNORE [RECORD | TRANSACTION [INCLUDEVENT]]
[DISCARD]
[LOG [INFO | WARNING]]
[REPORT]
[ROLLOVER]
[SHELL "<command>" |
   SHELL ("<command>", VAR <variable> = {<column name> | <expression>}
   [, ...][, ...]) ]
[TRACE[2] <trace file> [TRANSACTION] [DDL[INCLUDE] | DDLONLY]
   [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.

Action	Description
`SUSPEND`	Pauses the process so that it retains the active context of the current run and can still respond to SEND commands that are issued in GGSCI. When a process is suspended, the INFO command shows it as RUNNING, and the RBA field shows the last checkpoint position. To resume processing, issue the SEND <group> command with the RESUME option. To use the CHECKPOINT BEFORE option in conjunction with SUSPEND, the event record must be the start of a transaction for the SUSPEND to take place. That way, if the process is killed while in the suspended state, the event record with the SUSPEND action is the first record to be reprocessed upon restart. If both CHECKPOINT BERORE and SUSPEND are specified, but the event record is not the start of a transaction, the process abends before SUSPEND can take place. To use the CHECKPOINT AFTER option in conjunction with SUSPEND, the RESUME command must be issued before the checkpoint can take place, and the event record must be a COMMIT record. If the process is killed while in a SUSPEND state, the process reprocesses the transaction from the last checkpointed position upon restart. SUSPEND cannot be combined with ABORT but can be combined with all other options.
`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.
`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.

Action	Description

`IGNORE`
`[RECORD | TRANSACTION`
`[INCLUDEVENT]]`
	Ignores some or all of the transaction, depending on the selected action.
	◆ RECORD is the default. It forces the process to ignore only the specified event record, but not the rest of the transaction. 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.
	This action is not valid for DDL records. Because DDL operations are autonomous, ignoring a record is equivalent to ignoring the entire transaction.

`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.
`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.

Action	Description
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.
SHELL "<command>"	Causes the process to execute the specified shell command when the event record is encountered. SHELL "<command>" executes a basic shell command. The command string is taken at its literal value and sent to the system that way. The command is case-sensitive and must be enclosed within double quote marks, for example: `EVENTACTIONS (SHELL "echo hello world! > output.txt")` 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. Note that the error message relates only to the execution of the SHELL command itself, and not the exit status of any subordinate commands. For example, SHELL can execute a script successfully, but commands in that script could fail. SHELL can be combined with all other EVENTACTIONS options.

Action	**Description**
`SHELL ("<command>", VAR <variable> = {<column name> | <expression>} [, ...][, ...])`	Causes the process to execute the specified shell command when the event record is encountered and supports parameter passing. The command and the parameters are case-sensitive. By default, any input value that is within double quotes is treated as literal text. However, you can use the USEANSISQLQUOTES parameter in the GLOBALS file to apply SQL-92 rules, where text enclosed in single quotes is treated as a literal, and text enclosed in double quotes is treated as a name. See also USEANSISQLQUOTES. When SHELL is used with arguments, the entire command and argument strings must be enclosed within parentheses, for example: `EVENTACTIONS (SHELL ("Current timestamp: $1  SQLEXEC result is $2 ",VAR $1 = @GETENV("JULIANTIMESTAMP"),VAR $2 = mytest.description));` The input is as follows: <command> is the command, which is passed literally to the system. VAR is a required keyword that starts the parameter input. <variable> is the user-defined name of the placeholder variable where the run-time variable value will be substituted. Extra variables that are not used in the command are ignored. Note that any literal in the SHELL command that matches a VAR variable name is replaced by the substituted VAR value. This may have unintended consequences, so test your code before putting it into production. <column name > can be the before or after (current) image of a column value. <expression> can be the following, depending on whether column data or DDL is being handled. **Column data valid expressions:** ◆ The value from a TOKENS clause in a TABLE statement. ◆ A return value from any Oracle GoldenGate column-conversion function. ◆ A return value from a SQLEXEC query or procedure. **DDL valid expressions**: ◆ Return value from @TOKEN function (Replicat only). ◆ Return value from @GETENV function. ◆ Return value from other functions that do not reference column data (for example, @DATENOW). ◆ Return value from @DDL function.

Action	Description

`TRACE[2] <trace file>`
`[TRANSACTION]`
`[DDL[INCLUDE] | DDLONLY]`
`[PURGE | APPEND]`
	Causes process trace information to be written to a trace file when the specified event record is encountered. TRACE provides step-by-step processing information. TRACE2 identifies the code segments on which the process is spending the most time.
	By default (without options), standard DML tracing without consideration of transaction boundaries is enabled until the process terminates.
	◆ <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. This option is valid only for DML operations.
	◆ DDL[INCLUDE] traces DDL and also DML transactional data processing. Either DDL or DDLINCLUDE is valid.
	◆ DDLONLY traces DDL but does not trace DML transactional data.
	These options are valid only for Replicat. By default DDL tracing is disabled.
	◆ 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.
	When using EVENTACTIONS for a DDL record, note that since each DDL record is autonomous, the DDL record is guaranteed to be the start of a transaction; therefore the CHECKPOINT BEFORE event action is implied for a DDL record.
	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.
	When using EVENTACTIONS for a DDL record, note that since each DDL record is autonomous, the DDL record is guaranteed to be the end (boundary) of a transaction; therefore the CHECKPOINT AFTER event action is implied for a DDL record.
	CHECKPOINT AFTER can be combined with all EVENTACTIONS options except ABORT.

Action	Description
	◆ 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.

Example 1 The following example shows how you can configure a process to ignore certain records. When Replicat processes any trail record that has name = goldengate, it ignores the record.

```
MAP <owner.table>, TARGET <owner2.table2>, &
WHERE (name = “goldengate”), &
EVENTACTIONS (ignore);
```

Example 2 Based on the compatibility and precedence rules of EVENTACTIONS options, DISCARD takes higher precedence than ABORT, so in this example the event record gets written to the discard file before the process abends.

```
MAP <owner.table>, TARGET <owner2.table2>, &
WHERE (name = “goldengate”), &
EVENTACTIONS (DISCARD, ABORT);
```

Example 3 The following example executes a SHELL action. It gets the result of a SQLEXEC query and pairs it with the current timestamp.

```
MAP src.tab1, TARGET targ.tab1 &
SQLEXEC (id mytest, query "select description from lookup &
where pop = :mycol2", params (mycol2 = col2) ), &
EVENTACTIONS (SHELL ("Current timestamp: $1  SQLEXEC result is $2 ", &
VAR $1 = @GETENV("JULIANTIMESTAMP"), VAR $2 = mytest.description));
```

The shell command that results from this example could be similar to the following:

```
"Current timestamp: 212156002704718000  SQLEXEC result is test passed"
```

Example 4 The following example shows how invalid results can occur if a placeholder name conflicts with literal text in the command string. In this example, a placeholder named “$1” is associated with a column value, and the SHELL command echoes a literal string that includes “$1.”

```
MAP src.tab1, TARGET targ.tab1 &
EVENTACTIONS (SHELL ("echo Extra charge for $1 is $1", VAR $1 = COL1));
```

This is the unintended result, assuming the column value is “gift wrap”:

```
"Extra charge for gift wrap is gift wrap"
```

Changing the placeholder variable to “$col” results in the correct output:

```
MAP src.tab1, TARGET targ.tab1 &
EVENTACTIONS (SHELL ("echo Extra charge for $col is $1", VAR $col = COL1));
```

```
"Extra charge for gift wrap is $1"
```

The following shows similar potential for unintended results:

```
MAP src.tab1, TARGET targ.tab1 &
EVENTACTIONS (SHELL ("Timestamp: $1  Price is $13 > out.txt ", &
VAR $1 = @GETENV("JULIANTIMESTAMP")));
```

The redirected output file might contain a string like this (notice the second timestamp contains an appended 3):

```
"Timestamp: 212156002704718000 Price is 2121560027047180003"
```

The intended result is this:

```
"Timestamp: 212156002704718000 Price is $13"
```

Example 5 The following is an example of SHELL used for a DDL operation. The @DDL function is used to return the text of DDL statements.

```
DDL INCLUDE OBJNAME src.t* &
EVENTACTIONS (SHELL ("echo The DDL text is var1> out.txt ", &
VAR var1 = DDL(TEXT));
```

The redirected output file might contain a string like this:

```
"The DDL text is CREATE TABLE src.test_tab (col1 int);"
```

Example 6 These examples show different ways to configure tracing.

```
MAP tab1, TARGET tab1 EVENTACTIONS (TRACE ./dirrpt/trace1.txt);
MAP tab2, TARGET tab2 EVENTACTIONS (TRACE ./dirrpt/trace2.txt TRANSACTION);
```

- In the first MAP statement, the trace1.txt trace file will be generated just before the first tab1 event record gets applied to the target. It will contain all of the tracing information from that point forward until Replicat terminates or unless tracing gets turned off with the GGSCI SEND REPLICAT command.
- Because the second MAP statement contains the TRANSACTION option, the trace2.txt file will be generated just before the first tab2 event record gets applied to the target, but the tracing will stop automatically at the conclusion of the transaction that contains the tab2 event record.

For additional use cases and more information about the event marker system, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Example 7 The following shows how EVENTACTIONS with SUSPEND can be used.

- You are replicating DDL, and you want to ensure that there is enough space in the target database to create a new table. Use EVENTACTIONS with SUSPEND in the MAP statement that maps the CREATE TABLE DDL operation, and execute a SQL statement in that MAP statement to query the amount of space remaining in a tablespace. If there is enough space, use SEND EXTRACT with RESUME to resume processing immediately; if not, leave Replicat suspended until a DBA can add the space, and then use SEND EXTRACT with RESUME to resume processing.

- You want to fix unique key violations when they occur on any table. Because Replicat is processing thousands of tables, you do not want to stop the process each time there is a violation, because this would cause Replicat to spend time rebuilding the object cache again upon restart. By using EVENTACTIONS with SUSPEND, you can simply suspend processing until the problem is fixed.
- At the end of the day, you suspend Replicat to run daily reports, and then resume processing immediately without having to stop and restart the process.

Using EXCEPTIONSONLY

Use EXCEPTIONSONLY in an exceptions MAP statement intended for error handling. The exceptions MAP statement must follow the MAP statement for which errors are anticipated. The exceptions MAP statement executes only if an error occurs for the last record processed in the preceding regular MAP statement.

To use EXCEPTIONSONLY, use a REPERROR statement with the EXCEPTION option either within the regular MAP statement or at the root of the parameter file. For more information about REPERROR, see page 298.

NOTE If using the Oracle GoldenGate Conflict Detection and Resolution (CDR) feature, a REPERROR with EXCEPTION is not needed. CDR automatically sends all operations that cause errors to the exceptions MAP statement.

The exceptions MAP statement must specify the same SOURCE table as in the regular MAP statement, but the TARGET table in the exceptions MAP statement must be an exceptions table. When using an exceptions MAP statement, do not use wildcarded object names in the regular MAP statement.

NOTE See MAPEXCEPTION option to support wildcarded object names.

For more information about using an exceptions MAP statement, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Syntax

```
MAP <table spec>, TARGET <table spec>, EXCEPTIONSONLY
```

Using EXITPARAM

Use EXITPARAM to pass a parameter to a user exit routine whenever a record from the MAP statement is encountered. For more information about user exits, see Chapter 5.

Syntax

```
MAP <table spec>, TARGET <table spec>,
EXITPARAM "<parameter string>";
```

Component	Description
`"<parameter string>"`	A parameter that is a literal string. Enclose the parameter within double quotes. You can specify up to 100 characters for the parameter string.

Using FILTER

Use FILTER to select or exclude records based on a numeric value. A filter expression can use conditional operators, Oracle GoldenGate column-conversion functions, or both.

NOTE To filter based on a string, use a string function or use the WHERE option.

Separate all FILTER components with commas. A FILTER clause can include the following:

- 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

Syntax

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

Component	Description

`<filter clause>`	Selects records based on an expression, such as:
	`FILTER ((PRODUCT_PRICE*PRODUCT_AMOUNT)>10000))`
	You can use the column-conversion functions of Oracle GoldenGate in a filter clause, as in:
	`FILTER (@COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`
	By default, Oracle GoldenGate treats any input string within double quotes as a literal, as in `FILTER (@STRFIND(NAME, "JOE") > 0)`. To use SQL-92 rules for identifiers and literals, use the USEANSISQLQUOTES parameter in the GLOBALS file.
	Oracle GoldenGate does not support FILTER for columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.
	The maximum size of the filter clause is 5,000 bytes.

`ON INSERT |`
`ON UPDATE|`
`ON DELETE`
	Restricts record filtering to the specified operation(s). Separate operations with commas, for example:
	`FILTER (ON UPDATE, ON DELETE,`
	`@COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`
	This example executes the filter for updates and deletes, but not inserts.

`IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE`	Does not apply the filter for the specified operation(s). Separate operations with commas, for example:
	`FILTER (IGNORE INSERT, @COMPUTE`
	`(PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`
	This example executes the filter on updates and deletes, but ignores inserts.

`RAISEERROR <error>`	Raises a user-defined error number if the filter fails. Can be used as input to the REPERROR parameter to invoke error handling. Make certain that the value for <error> is outside the range of error numbers that is used by the database or by Oracle GoldenGate. For example: RAISEERROR 21000.

Using HANDLECOLLISIONS | NOHANDLECOLLISIONS

Use HANDLECOLLISIONS and NOHANDLECOLLISIONS to control whether or not Oracle GoldenGate adjusts for transactional changes to source tables that were replicated by Oracle GoldenGate while an initial load of the same tables was being applied to the target tables. HANDLECOLLISIONS is required by initial load methods where the source tables remain online and users are making data changes. When Oracle GoldenGate applies replicated changes after the load is finished, HANDLECOLLISIONS causes Replicat to overwrite duplicate records in the target tables and provides alternate handling of errors for missing records.

HANDLECOLLISIONS and NOHANDLECOLLISIONS also can be used globally in the parameter file or as an on/off switch for groups of tables. When used in a MAP statement, they override those other specifications. For more information about HANDLECOLLISIONS, see page 217.

Syntax

```
MAP <table spec>, TARGET <table spec>,
[HANDLECOLLISIONS | NOHANDLECOLLISIONS];
```

Example 1 This example shows the basic use within a MAP statement.

```
MAP dbo.tcust, TARGET dbo.tcust, HANDLECOLLISIONS;
```

Example 2 This example shows the combination of global and MAP-specific use within a parameter file. Because the MAP specification overrides the global specification, collisions will not be handled for the tcust table pair.

```
REPLICAT fin
USERID ggs, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1
HANDLECOLLISIONS
ASSUMETARGETDEFS
MAP dbo.torders, TARGET dbo.torders;
MAP dbo.tprod, TARGET dbo.tprod;
MAP dbo.tcust, TARGET dbo.tcust, NOHANDLECOLLISIONS;
```

Using INSERTALLRECORDS

Use the INSERTALLRECORDS parameter to keep a record of all operations made to a target record, instead of maintaining just the current version. INSERTALLRECORDS causes Replicat to insert every change operation made to a record as a new record in the database. The initial insert and subsequent updates and deletes are maintained as point-in-time snapshots.

Combining historical data with special transaction information provides a way to create a more useful target reporting database. For more information about maintaining a transaction-history table, see the *Oracle GoldenGate Windows and UNIX Administrator's Guide*.

INSERTALLRECORDS can also be used as a standalone parameter at the root level of the parameter file to affect multiple MAP statements at once. See page 223.

Syntax

```
MAP <table spec>, TARGET <table spec>, INSERTALLRECORDS;
```

Using INSERTAPPEND | NOINSERTAPPEND

Use the INSERTAPPEND and NOINSERTAPPEND parameters to control whether or not Replicat uses an APPEND hint when it applies INSERT operations to Oracle target tables. These parameters are valid only for Oracle databases.

These parameters can be used in two ways: When used as standalone parameters at the root of the parameter file, one remains in effect for all subsequent TABLE or MAP statements, until the other is encountered. When used within a MAP statement, they override any standalone INSERTAPPEND or NOINSERTAPPEND entry that precedes the MAP statement.

INSERTAPPEND causes Replicat to use the APPEND_VALUES hint when it applies INSERT operations to Oracle target tables. It is appropriate for use as a performance improvement when the replicated transactions are large and contain multiple inserts into the same table. If the transactions are small, using INSERTAPPEND can cause a performance decrease. For more information about when APPEND hints should be used, consult the Oracle documentation.

The BATCHSQL parameter must be used when using INSERTAPPEND. Replicat will abend if BATCHSQL is not used.

The default for Replicat INSERT statements is NOINSERTAPPEND.

See also INSERTAPPEND | NOINSERTAPPEND on page 222 for standalone usage syntax and example.

Syntax

```
MAP <table spec>, TARGET <table spec>, [INSERTAPPEND | NOINSERTAPPEND];
```

Example In the following, INSERTAPPEND is used for all of the tables in the MAP statements, except for the inventory table.

```
INSERTAPPEND
MAP fin.orders, TARGET fin.orders;
MAP fin.inventory, TARGET fin.inventory, NOINSERTAPPEND;
MAP fin.customers, TARGET fin.customers;
```

Using KEYCOLS

Use KEYCOLS to define one or more columns of the target table as unique. The primary use for KEYCOLS is to define a substitute primary key when a primary key or unique index is not available for the table.

Source and target key or unique-index columns must match, whether they are defined in the database or substitutes rendered by KEYCOLS. The source table must contain at least as many key or index columns as the target table. Otherwise, in the event of an update to the source key or index columns, Replicat will not have the before images for the extra target columns.

When defining keys, observe the following guidelines:

- If both the source and target tables lack keys or unique indexes, use KEYCOLS in both the TABLE and MAP statements, and specify matching sets of columns.
- If just one of the tables lacks a key or unique index, use KEYCOLS for that table, and specify columns that match the actual key or index columns of the other table. If a matching set cannot be defined, then use KEYCOLS in both the TABLE and MAP statements, and specify matching sets of columns that contain unique values. KEYCOLS overrides a key or unique index.
- If the target table has a larger key than the source table does (or more unique-index columns), KEYCOLS should be used in the TABLE statement to specify the actual source key or index columns, plus the source columns that match the extra target columns. Do not just specify the extra columns, because when a table has a primary key or unique index, the KEYCOLS specification will override them. Using KEYCOLS in this way ensures that before images are available for updates to the key or index columns.

When using KEYCOLS, make certain that the specified columns are logged to the transaction log so that they are available to Replicat in the trails. You can do so by using the database interface or by using the COLS option of the ADD TRANDATA command (Oracle only).

On the target tables, create a unique index on the KEYCOLS-defined key columns. An index improves the speed with which Oracle GoldenGate locates the target rows that it needs to process.

Syntax

```
MAP <table spec>, TARGET <table spec>, KEYCOLS (<column> [, ... ]);
```

Component	Description
`(<column>)`	Defines a column to be used as a substitute primary key. To specify multiple columns, create a comma-delimited list as in: `KEYCOLS (id, name` If a primary or unique key exists, those columns must be included in the KEYCOLS specification. The following column-types are not supported in KEYCOLS: **Oracle column types not supported by KEYCOLS**: Virtual columns, UDTs, function-based columns, and any columns that are explicitly excluded from the Oracle GoldenGate configuration **SQL Server, DB2 LUW, DB2 z/OS, MySQL, SQL/MX, Teradata, TimesTen column types not supported by KEYCOLS**: Columns that contain a timestamp or non-materialized computed column, and any columns excluded from the Oracle GoldenGate configuration. For SQL Server Oracle GoldenGate enforces the total length of data in rows for target tables without a primary key to be below 8000 bytes. **Sybase column types not supported by KEYCOLS**: Computed columns, function-based columns, and any columns that are explicitly excluded from the GoldenGate configuration

Using MAPEXCEPTION

Use MAPEXCEPTION to specify mapping and other options for operations that are flagged as exceptions by the REPERROR parameter. MAPEXCEPTION specifies an *exceptions table* to which Replicat can write the failed operations, and it allows for mapping and handling options.

You can use MAPEXCEPTION within the same MAP statement that includes the source-target table mapping and other standard MAP options. The source and target table names can include wildcards.

To use MAPEXCEPTION, use a REPERROR statement with the EXCEPTION option either within the same MAP statement or at the root of the Replicat parameter file.

Syntax

```
MAP <object spec>, TARGET <object spec>,
[<MAP_options>],
MAPEXCEPTION (TARGET <object spec> [, <exception_MAP_options>]);
```

Argument	Description
`TARGET`	A required keyword.
`<MAP_options>`	Regular MAP options to process non-exception operations (those that succeed) as needed.

Argument	Description
`<object spec>`	The fully qualified name of the exceptions table. Standard Oracle GoldenGate rules apply for object names. See the guidelines in this documentation for MAP.
`<exception_MAP_options>`	Any valid options of the MAP parameter. These options apply to the exceptions handling. To map non-exception data, also use MAP options outside the MAPEXCEPTION clause.

Example 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”)
)
);
```

Using REPERROR

Use REPERROR to specify an error and a response that together control how Replicat responds to the error when executing the MAP statement. You can use REPERROR at the MAP level to override and supplement global error handling rules set with the REPERROR parameter at the root level of the parameter file. Multiple REPERROR statements can be applied to the same MAP statement to enable automatic, comprehensive management of errors and interruption-free replication processing.

Syntax

```
MAP <object spec>, TARGET <object spec>,
REPERROR (
{DEFAULT | DEFAULT2 | <SQL error> | <user-defined error>},
{ABEND | DISCARD | EXCEPTION | IGNORE |
RETRYOP [MAXRETRIES <n>] |
TRANSABORT [, MAXRETRIES] [, DELAYSECS <n> | DELAYCSECS <n>] |
TRANSDISCARD | TRANSEXCEPTION }
)
[, ...];
```

Argument	Description
The following options specify the error to handle:	
`DEFAULT`	Sets a global response to all errors except those for which explicit REPERROR statements are specified.

Argument	Description
`DEFAULT2`	Provides a backup default action when the response for DEFAULT is set to EXCEPTION. Use DEFAULT2 when an exceptions MAP statement is not specified for a MAP statement for which errors are anticipated.
`<SQL error>`	A SQL error number.
`<user-defined error>`	A user-defined error that is specified with the RAISEERROR option of a FILTER clause in a MAP statement.
The following options specify a response to the error:	
`ABEND`	Roll back the transaction and terminate processing abnormally. ABEND is the default.
`DISCARD`	Log the error to the discard file but continue processing the transaction and subsequent transactions.
`EXCEPTION`	Handle the error as an exception. In anticipation of possible errors, you use this option in conjunction with an exceptions MAP statement or to work with the MAPEXCEPTION option of MAP. This option handles exceptions for an individual operation, without affecting other error-free operations in the same transaction. To handle exceptions at the transaction level, use the TRANSEXCEPTION option. For more information about how to configure error handling, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*. *Note*: When the Conflict Detection and Resolution (CDR) feature is active, CDR automatically treats all operations that cause conflicts as exceptions if an exceptions MAP statement exists for the affected table. In this case, REPERROR with EXCEPTION is not necessary, but you should use REPERROR with other options to handle conflicts that CDR cannot resolve, or for conflicts that you do not want CDR to handle.
`IGNORE`	Ignore the error.
`RETRYOP [MAXRETRIES <n>]`	Retry the operation. Use the MAXRETRIES option to control the number of retries. For example, if a table is out of extents, RETRYOP with MAXRETRIES gives you time to add extents so the transaction does not fail. Replicat abends after the specified number of MAXRETRIES. To set the amount of time between attempts, set RETRYDELAY as described on page 311.

`TRANSABORT`
`[, MAXRETRIES]`
`[, DELAYSECS <n> |`
`DELAYCSECS <n>]`
	Abort the transaction and reposition to the beginning of the transaction. This will continue either until the operation(s) are processed successfully or MAXRETRIES expires. If MAXRETRIES is not set, the TRANSABORT action will loop continuously.
	Use a DELAY option to wait a specified amount of time before the retry.

Argument	Description
TRANSDISCARD	Discard the entire replicated source transaction if any operation within that transaction, including the commit operation, causes a Replicat error that is listed in the error specification. Replicat aborts the transaction and, if the error occured on a record, writes that record to the discard file that is specified with the DISCARDFILE parameter. Replicat then replays the transaction and writes all of the records to the discard file, including the commit record. Replicat abends on errors that are caused by the discard processing. If the discarded record has already been data-mapped to a target record, Replicat writes it to the discard file in the target format; otherwise, it will be written in source format. The replayed transaction itself is always written in source format. TRANSDISCARD supports record-level errors as well as commit errors. See "REPERROR" on page 298 for additional information about this option.
TRANSEXCEPTION	Perform exceptions mapping for every record in the replicated source transaction according to its exceptions-mapping statement, as defined by a MAPEXCEPTION or EXCEPTIONSONLY clause in an exceptions MAP statement. If any record does not have a corresponding exceptions mapping specification, or if there is an error writing to the exceptions table, Replicat abends with an error message. When an error is encountered and TRANSEXCEPTION is being used, Replicat aborts the transaction and, if the error occurred on a record, writes that record to the discard file that is specified with the DISCARDFILE parameter. Replicat replays the transaction and examines the source records to find the exceptions-mapping specifications, and then executes them. TRANSEXCEPTION supports record-level errors as well as commit errors. To handle errors at the record level (for individual SQL operations), without affecting error-free operations in the same transaction, use the EXCEPTION option. See "REPERROR" on page 298 for additional information about this option.

Example The following examples show different ways that REPERROR can be used in a MAP statement in conjunction with a global REPERROR statement.

Example 1:

```
REPLICAT <group>
REPERROR (<error1> , <response1>)
MAP <src1>, TARGET <tgt1>, REPERROR (<error1>, <response2>);
MAP <src2>, TARGET <tgt2>, REPERROR (<error2>, <response3>);
```

In the preceding example, when error1 occurs for the first MAP statement, the action should be response2, not response1, because an override was specified. However, if an error1 occurs for the second MAP statement, the response should be response1, the global response. The response for error2 would be response3, which is MAP-specific.

Example 2:

```
REPLICAT <group>
REPERROR (<error1> , <response1>)
MAP <src1>, TARGET <tgt1>, REPERROR (<error2>, <response2>),
REPERROR (<error3>, <response3>);
```

In the preceding example, when replicating from src1 to src2, all errors and actions (1-3) should apply, because all REPERROR statements address different errors (there are no MAP-specific overrides).

Example 3:

```
REPLICAT <group>
REPERROR (<error1> , <response1>)
MAP <src1>, TARGET <tgt1>, REPERROR (<error1>, <response2>);
MAP <src2>, TARGET <tgt2>, REPERROR (<error2>, <response3>);

REPERROR (<error1> , <response4>)
MAP <src2>, TARGET <tgt2>, REPERROR (<error3>, <response3>);
```

In the preceding example, if error1 occurs for the first MAP statement, the action should be response2. For the second one it would be response1 (the global response), and for the third one it would be response4 (because of the second REPERROR statement). A global REPERROR statement applies to all MAP statements that follow it in the parameter file until another REPERROR statement starts new rules.

Example 4:

```
REPERROR DEFAULT ABEND
REPERROR 1403 TRANSDISCARD.
MAP src, TARGET tgt, REPERROR(600 TRANSDISCARD);
```

In the preceding example, if error 600 is encountered while applying source table src to target table tgt, the whole transaction is written to discard file. Encountering error 1403 also results in the same action based on the global REPERROR specification. On the other errors, the process simply discards only the offending record and then abends.

Using RESOLVECONFLICT

Use RESOLVECONFLICT to specify how Replicat handles conflicts on operations made to the tables in the MAP statement in a bi-directional or multi-master configuration. For detailed information about conflicts and conflict resolution, see Chapter 9 of the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

RESOLVECONFLICT supports the following resolutions:

- Resolve a uniqueness conflict for an INSERT.
- Resolve a “no data found” conflict for an UPDATE when the row exists, but the before image of one or more columns is different from the current value in the database.
- Resolve a “no data found” conflict for an UPDATE when the row does not exist.
- Resolve a “no data found” conflict for a DELETE when the row exists, but the before image of one or more columns is different from the current value in the database.
- Resolve a “no data found” conflict for a DELETE when the row does not exist.

Multiple resolutions can be specified for the same conflict type and are executed in the order listed in RESOLVECONFLICT. Multiple resolutions are limited to INSERTROWEXISTS and UPDATEROWEXISTS conflicts only.

RESOLVECONFLICT can be used multiple times in a MAP statement to specify different resolutions for different conflict types.

If BATCHSQL is being used, conflict detection is performed in batch mode, but Replicat falls back to GROUPTRANSOPS mode to perform the resolution. (For more information, see the BATCHSQL parameter.)

Supported data types and platforms

- RESOLVECONFLICT supports all databases that are supported by Oracle GoldenGate for Windows and UNIX.
- To use RESOLVECONFLICT, the database must reside on a Windows, Linux, or UNIX system. It is not supported for databases on the NonStop platform.
- CDR supports data types that can be compared with simple SQL and without explicit conversion. See the individual parameter options for details.
- Do not use RESOLVECONFLICT for columns that contain LOBs, abstract data types (ADT), or user-defined types (UDT).

Making column values available for CDR

To use CDR, a full image for updates and deletes is required.

1. Use the NOCOMPRESSDELETES and NOCOMPRESSUPDATES parameters so that all of the columns of a record are written to the trail for DELETE and UPDATE operations and available to Replicat for CDR.
2. Use the GETBEFORECOLS option of the Extrat TABLE parameter to direct Extract to capture the before images that the database logs, and to write them to the trail. For DB2 databases, use the GETUPDATEBEFORES parameter instead of GETBEFORECOLS.
3. To capture full image records from a SQL/MX transaction log (audit trail), the table must be created or altered with the attribute of no auditcompress.

Error handling

Conflict resolution is performed before HANDLECOLLSIONS, INSERTMISSINGUPDATES, and REPERROR when those parameters are used. At minimum, you should use REPERROR to handle errors that CDR cannot resolve. For example, if CDR resolution returns a “no data found” error, you could set REPLICAT to DISCARD to write the failed operation to a discard file, or you could set it to EXCEPTION to write it to the same exceptions MAP statement and exceptions table that is used for CDR conflicts. It might also be appropriate to use REPERROR to handle some errors instead of using CDR to handle them, if this better satisfies your business rules. Here is an example:

```
REPERROR (1403, EXCEPTION)
   RESOLVECONFLICT (UPDATEROWEXISTS, &
   (max_resolution_method, USEMAX (Modified_TS), COLS &
   (Address, Modified_TS)),(DEFAULT, OVERWRITE));
MAP source.Order, TARGET target.Order_ExceptionTable, EXCEPTIONSONLY, &
INSERTALLRECORDS;
```

For more detailed instructions on configuring conflict resolution, see Chapter 9 of the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

```
MAP <source table>, TARGET <target table>,
RESOLVECONFLICT (
   {INSERTROWEXISTS |
   UPDATEROWEXISTS |
   UPDATEROWMISSING |
   DELETEROWEXISTS |
   DELETEROWMISSING}
   ( {DEFAULT | <resolution name>},
      {USEMAX (<res_col>) |
      USEMIN (<res_col>) |
      USEDELTA |
      DISCARD |
      OVERWRITE |
      IGNORE})
         [, COLS (<col>[,...])]
)
[RESOLVECONFLICT (, ...)]
```

Argument	Description

`INSERTROWEXISTS |`
`UPDATEROWEXISTS |`
`UPDATEROWMISSING |`
`DELETEROWEXISTS |`
`DELETEROWMISSING`
	The type of conflict that this resolution handles.
	INSERTROWEXISTS
	An inserted row violates a uniqueness constraint on the target.
	UPDATEROWEXISTS
	An updated row exists on the target, but one or more columns have a before image in the trail that is different from the current value in the database.
	UPDATEROWMISSING
	An updated row does not exist in the target.
	DELETEROWEXISTS
	A deleted row exists in the target, but one or more columns have a before image in the trail that is different from the current value in the database.
	DELETEROWMISSING
	A deleted row does not exist in the target.

Argument	Description
`DEFAULT | <resolution name>`	DEFAULT The default column group. The resolution that is associated with the DEFAULT column group is used for all columns that are not in an explicitly named column group. You must define a DEFAULT column group. <resolution name> A name for a specific column group that is linked to a specific resolution type. Supply a name that identifies the resolution type. Valid values are alphanumeric characters. Avoid spaces and special characters, but underscores are permitted, for example: `delta_res_method` Use either a named resolution or DEFAULT, but not both.

`USEMAX (<res_col>) |`
`USEMIN (<res_col>) |`
`USEDELTA |`
`DISCARD |`
`OVERWRITE |`
`IGNORE`
	The conflict-handler logic that is used to resolve the conflict. Valid resolutions are:
	USEMAX
	If the value of <res_col> in the trail record is greater than the value of the column in the database, then the appropriate action performed. (See “Action”)
	USEMIN
	If the value of <res_col> in the trail record is less than the value of the column in the database, then the appropriate action is performed. (See “Action”)
	Action:
	◆ (INSERTROWEXISTS conflict) Apply the trail record, but change the insert to an update to avoid a uniqueness violation, and overwrite the existing values.
	◆ (UPDATEROWEXISTS conflict) Apply the update from the trail record.

Argument	Description
	<res_col> The name of a NOT NULL column that serves as the resolution column. This column must be part of the column group that is associated with this resolution. The value of the resolution column compared to the current value in the target database determines how a resolution should be applied. The after image of the resolution column is used for the comparison, if available; otherwise the before image value is used. Use a column that can be compared through simple SQL: ◆ NUMERIC ◆ DATE ◆ TIMESTAMP ◆ CHAR/NCHAR ◆ VARCHAR/ NVARCHAR To use a latest-timestamp resolution, use a timestamp column as the <res_col> and set the timestamp column to the current time when a row is inserted or updated. If possible, define the resolution column with the SYSTIMESTAMP data type, which supports fractional seconds. When comparisons are performed with sub-second granularity, there is little need for tie-breaking conflict handlers that resolve cases where the value of the resolution column is identical in both trail and target. If you ensure that the value of the timestamp column can only increase or only decrease (depending on the resolution), then USEMAX and USEMIN does not lead to data divergence. USEDELTA (UPDATEROWEXISTS conflict only) Add the difference between the before and after values in the trail record to the current value of the column in the target database. If any of the values is NULL, an error is raised. Base USEDELTA on columns that contain NUMERIC data types. USEDELTA is useful in a multi-node configuration when a row is getting simultaneously updated on multiple nodes. It propagates only the difference in the column values to the other nodes, so that all nodes become synchronized. DISCARD (Valid for all conflict types) Retain the current value in the target database, and write the data in the trail record to the discard file. Requires a discard file to be specified with the DISCARDFILE parameter. Use DISCARD with caution, because it can lead to data divergence.

Argument	Description
	OVERWRITE (Valid for all conflict types except DELETEROWMISSING) Apply the trail record as follows: ◆ (INSERTROWEXISTS conflict) Apply the trail record but change the insert to an update to avoid a uniqueness violation, and overwrite the existing values. ◆ (UPDATEROWEXISTS conflict) Apply the update from the trail record. ◆ (UPDATEROWMISSING conflict) Apply the trail record but change the update to an insert. To convert an update to an insert, the before image of all columns of the row must be available in the trail. Use supplemental logging if the database does not log before images by default, and specify ALL for the Extract GETBEFORECOLS parameter. ◆ (DELETEROWEXISTS conflict) Apply the delete from the trail record, but use only the primary key columns in the WHERE clause. Use OVERWRITE with caution, because it can lead to data divergence. IGNORE (Valid for all conflict types) Retain the current value in the target database, and ignore the trail record: Do not apply to the target table or a discard file.
`COLS (<col>[,...])`	A non-default column group. This is a list of columns in the target database (after mapping) that are linked to, and operated upon by, a specific resolution type. If no column group is specified for a conflict, then all columns are affected by the resolution that is specified for the given conflict. Alternatively, you can specify a DEFAULT column group, which includes all columns that are not listed in another column group. See the DEFAULT option. You can specify multiple column groups, each with a different resolution. For example, you could use OVERWRITE for col2 and col3, while you could use USEDELTA for col4. No column in any group can be in any other group. Conflicts for columns in different column groups are resolved separately according to the specified resolution, and in the order listed.

Argument	Description
	Column groups work as follows: ◆ For INSERTROWEXISTS and UPDATEROWEXISTS conflicts, you can use different column groups to specify more than one of these conflict types and resolutions per table. Conflicts for columns in different column groups are resolved separately, according to the conflict resolution method specified for the column group. ◆ For UPDATEROWMISSING, DELETEROWEXISTS, and DELETEROWMISSING, you can use only one column group, and all columns of the table must be in this column group (considered the *default* column group).

Using SQLEXEC

Use SQLEXEC to execute a SQL stored procedure or query from within a MAP statement during Oracle GoldenGate processing. SQLEXEC enables Oracle GoldenGate to communicate directly with the database to perform any function supported by the database. The database function can be part of the synchronization process, such as retrieving values for column conversion, or it can be independent of extracting or replicating data.

When used within a MAP statement, the procedure or query that is executed can accept input parameters from source or target rows and pass output parameters.

NOTE A query or procedure must be structured correctly when executing a SQLEXEC statement. If Replicat encounters a problem with the query or procedure, the process will immediately abend, regardless of any error-handling rules that are in place.

Supported data types

The following are the data types that are supported by SQLEXEC for input and output parameters.

- Numeric data types
- Date data types
- Character data types

For additional instructions for using stored procedures and queries with Oracle GoldenGate, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

SQLEXEC dependencies and restrictions

- The SQL is executed by the database user under which the Oracle GoldenGate process is running. This user must have the privilege to execute stored procedures and call database-supplied procedures.
- A query or procedure must be structured correctly when executing a SQLEXEC statement, with legal SQL syntax for the database; otherwise Replicat will abend, regardless of any error-handling rules that are in place. Refer to the SQL reference guide provided by the database vendor for permissible SQL syntax.

- Do not use SQLEXEC to change a value in a primary key column. The primary key value is passed from Extract to Replicat. Without it, Replicat operations cannot be completed. If primary key values must be changed with SQLEXEC, you may be able to avoid errors by mapping the original key value to another column and then defining that column as a substitute key with the KEYCOLS option. See "Using KEYCOLS" on page 255.
- For DB2 on z/OS, 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 for z/OS documentation for more information.

When using Oracle GoldenGate to replicate DDL, all objects that are affected by a 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.

Using SQLEXEC with stored procedures

To execute a stored procedure from within a MAP statement, use the SPNAME clause.

Syntax

```
SQLEXEC (
SPNAME <sp name>
[, ID <logical name>]
{, PARAMS <param spec> | NOPARAMS}
[, <option>] [, ...]
)
```

Component	Description
`<sp name>`	Specifies the name of the procedure to execute.
`ID <logical name>`	Defines a logical name for the procedure. Use this option to execute the procedure multiple times within a MAP statement. Up to 20 stored procedures can be executed per MAP statement. ID is not required when executing a procedure once.
`PARAMS <param spec> | NOPARAMS`	Defines whether or not the procedure accepts parameters. Either PARAMS <param spec> or NOPARAMS must be used. <param spec> defines input parameters and the source of the input.

`<option>`	Represents one of the following options that can be used alone or in conjunction with other options to control the effects of the stored procedure.
	`AFTERFILTER | BEFOREFILTER`
	`ALLPARAMS`
	`DBOP`
	`ERROR`
	`EXEC`
	`MAXVARCHARLEN`
	`PARAMBUFSIZE`
	`TRACE`

Descriptions of SQLEXEC components begin alphabetically on page 270.

Using SQLEXEC with queries

To execute a query from within a MAP statement, use the ID and QUERY clauses.

Syntax

```
SQLEXEC (
ID <logical name>
, QUERY "<sql query>"
{, PARAMS <param spec>| NOPARAMS}
[, <option>] [, ...]
)
```

Component	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. The query must be valid, standard query language for the database against which it is being executed. The query can either return results with a SELECT statement or execute an INSERT, UPDATE, or DELETE statement. For any query that produces output with a SELECT statement, only the first row returned by the SELECT is processed. Do not specify an "INTO ..." clause for any SELECT statements. The query must be contained on one line, within quotes. To use quoted object names within a SQLEXEC query, the SQL query must be enclosed within single quotes, rather than double quotes, and the USEANSISQLQUOTES parameter must be used in the GLOBALS file to enforce SQL-92 rules for object and literal identifiers. The following is an example of using quoted object names in a query: `SQLEXEC 'SELECT "col1" from "schema"."table"'` For best results, type a space after each begin quote and before each end quote.
`PARAMS <param spec>| NOPARAMS`	Defines whether or not the query accepts input parameters. One of these options must be used. <param spec> defines input parameters and the source of the input.

`<option>`	Represents one of the following options that you can use alone or in conjunction with other options to control the effects of the query.
	`AFTERFILTER | BEFOREFILTER`
	`ALLPARAMS`
	`DBOP`
	`ERROR`
	`EXEC`
	`MAXVARCHARLEN`
	`PARAMBUFSIZE`
	`TRACE`

Descriptions of SQLEXEC components begin alphabetically on page 270.

Using placeholders for input parameters

Most queries require placeholders for input parameters. How parameters are specified within the query depends on the database type.

- For Oracle, input parameters are specified by using a colon (:) followed by the parameter name, as in the following example.

  ```
  "SELECT NAME FROM ACCOUNT WHERE SSN = :SSN AND ACCOUNT = :ACCT"
  ```

- For other databases, input parameters are specified by using a question mark, as in the following example.

  ```
  "SELECT NAME FROM ACCOUNT WHERE SSN = ? AND ACCOUNT = ?"
  ```

Note that quotation marks are not required around the parameter name for any database.

Passing parameter values

Oracle GoldenGate provides options for passing input and output values to and from a procedure or query.

- To pass data values to input parameters within a stored procedure or query, use the PARAMs option of SQLEXEC (see page 275).
- To pass values from a stored procedure or query as input to a FILTER or COLMAP clause, use the following syntax:

  ```
  {<procedure name> | <logical name>}.<parameter>
  ```

  Where:

  - <procedure name> is the actual name of a stored procedure, which must match the value given for SPNAME in the SQLEXEC statement. Use this argument only if executing a procedure one time during the course of the Oracle GoldenGate run.
  - <logical name> is the logical name specified with the ID option of SQLEXEC. Use this argument to pass values from either a query or an instance of a stored procedure when the procedure executes multiple times within a MAP statement.
  - <parameter> is either the name of the parameter, such as a column in a lookup table, or RETURN_VALUE if extracting returned values.

As an alternative to the preceding syntax, you can use the @GETVAL function. For more information, see page 456.

There are different constructs for naming input parameters, as follows:

- Oracle permits naming an input parameter any logical name, for example:

  ```
  SQLEXEC (ID appphone, QUERY " select per_type from ps_personal_data "
     " where emplid = :vemplid "
     " and per_status = 'N' and per_type = 'A' ",
     PARAMS (vemplid = emplid)),
  TOKENS (applid = @GETVAL(appphone.per_type));
  ```

- Other databases require the input parameters to be named p1, p2, and so forth, increasing the number for each input parameter. Consider whether the database requires the "p" to be upper or lower case. The following is an example of this type of input parameter:

  ```
  SQLEXEC (ID appphone, QUERY " select per_type from ps_personal_data "
     " where emplid = ? "
     " and per_status = 'N' and per_type = 'A' ",
     PARAMS (p1 = emplid)),
  TOKENS (applid = @GETVAL(appphone.per_type));
  ```

Example The following shows a set of Oracle source and target tables, a lookup table, and examples of how parameters for these tables are passed for a single instance of a stored procedure and multiple instances of a stored procedure.

Source table "cust":

```
custid                      Number
current_residence_state     Char(2)
birth_state                 Char(2)
```

Target table "cust_extended":

```
custid                           Number
current_residence_state_long     Varchar(30)
birth_state_long                 Varchar(30)
```

Lookup table "state_lookup"

```
abbreviation     Char(2)
long_name        Varchar(30)
```

Example 1 The following example shows the use of a stored procedure that executes once to get a value from the lookup table. The value is mapped to the target column in the COLMAP statement.

```
MAP sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, &
PARAMS (long_name = birth_state)), &
COLMAP (custid = custid, &
birth_state_long = lookup.long_name);
```

Example 2 The following example shows multiple executions of a stored procedure that gets values from a lookup table. The values are mapped to target columns.

```
MAP sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, ID lookup1, &
PARAMS (long_name = current_residence_state)), &
SQLEXEC (SPNAME lookup, ID lookup2, &
PARAMS (long_name = birth_state)), &
COLMAP (custid = custid, current_residence_state_long = lookup1.long_name, &
birth_state_long = lookup2.long_name);
```

Using AFTERFILTER and BEFOREFILTER

Use AFTERFILTER and BEFOREFILTER to specify when to execute the stored procedure or query in relation to the FILTER clause of a MAP statement.

Syntax

```
AFTERFILTER | BEFOREFILTER
```

Rule	Description
AFTERFILTER	Causes the SQL to execute after the FILTER statement. This enables you to skip the overhead of executing the SQL unless the filter is successful. This is the default.
BEFOREFILTER	Causes the SQL to execute before the FILTER statement, so the results can be used in the filter.

Example

```
SQLEXEC (SPNAME check, NOPARAMS, BEFOREFILTER)
```

Using ALLPARAMS

Use ALLPARAMS as a global rule that determines whether or not all of the specified parameters must be present for the stored procedure or query to execute. Rules for individual parameters established within the PARAMS clause override the global rule set with ALLPARAMS.

Syntax

```
ALLPARAMS {OPTIONAL | REQUIRED}
```

Rule	Description
OPTIONAL	Permits the SQL to execute whether or not all of the parameters are present. This is the default.
REQUIRED	Requires all of the parameters to be present for the SQL to execute.

Example

```
SQLEXEC (SPNAME lookup,
PARAMS (long_name = birth_state, short_name = state),
ALLPARAMS OPTIONAL)
```

Using DBOP

Use DBOP to commit INSERT, UPDATE, DELETE, and SELECT statements executed within the stored procedure or query. Otherwise, they could potentially be rolled back. Oracle GoldenGate issues the commit within the same transaction boundaries as the source transaction.

WARNING Use caution when executing SQLEXEC procedures against the database, especially against the production database. Any changes that are committed by the procedure can result in overwriting existing data.

Syntax

```
DBOP
```

Example

```
SQLEXEC (SPNAME check, NOPARAMS, DBOP)
```

Using ERROR

Use ERROR to define a response to errors associated with the stored procedure or query. Without explicit error handling, the Oracle GoldenGate process abends on errors. Make certain your procedures return errors to the process and specify the responses with ERROR.

Syntax

```
ERROR <action>
```

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 “column missing” conditions. This is the default.
REPORT	Ensures that all errors associated with the stored procedure or query are reported to the discard file. (To report errors, a discard file must be specified with the DISCARDFILE parameter.) 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. Oracle GoldenGate continues processing other stored procedures or queries associated with the current MAP statement before processing the error.
FINAL	Is similar to RAISE except that when an error associated with a procedure or query is encountered, 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.

Example The following is an Oracle example exception statement that uses the RAISE option. It uses the Oracle function RAISE_APPLICATION_ERROR. If the error was defined in the Replicat parameter file with REPERROR (-20000, DISCARD), then Oracle GoldenGate will discard this record and continue processing.

```
EXCEPTION
WHEN no_match_rec THEN
RAISE_APPLICATION_ERROR(-20000, 'No Matching Update In Target');
```

Using EXEC

Use EXEC to control the frequency with which a stored procedure or query in a MAP statement executes and how long the results are considered valid, if extracting output parameters.

Syntax `EXEC <frequency>`

Frequency	Description
MAP	Executes the procedure or query once for each source-target table map for which it is specified. Using MAP renders the results invalid for any subsequent maps that have the same source table. For example, if a source table is being synchronized with more than one target table, the results would only be valid for the first source-target map. MAP is the default.

Frequency	Description
ONCE	Executes the procedure or query once during the course of the Oracle GoldenGate run, upon the first invocation of the associated MAP statement. The results remain valid for as long as the process remains running.
TRANSACTION	Executes the procedure or query once per source transaction. The results remain valid for all operations of the transaction.
SOURCEROW	Executes the procedure or query once per source row operation. Use this option when you are synchronizing a source table with more than one target table, so that the results of the procedure or query are invoked for each source-target mapping.

Example 1 The following is an example of using ONCE.

```
MAP sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, PARAMS (long_name = birth_state), EXEC ONCE), &
COLMAP (custid = custid, &
birth_state_long = lookup.long_name);
```

Example 2 The following is an example of using TRANSACTION.

```
MAP sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, PARAMS (long_name = birth_state), EXEC TRANSACTION), &
COLMAP (custid = custid, &
birth_state_long = lookup.long_name);
```

Example 3 The following is an example of using the default (MAP) incorrectly. The two MAP statements synchronize the same source table with two different target tables. However, the results of the procedure lookup will be expired by the time the second map executes, so the second map will result in a “column missing” condition. To implement this correctly, SOURCEROW should be used.

```
MAP sales.srctab, TARGET sales.targtab, &
SQLEXEC (SPNAME lookup, PARAMS (param1 = srccol)), &
COLMAP (targcol = lookup.param2);

MAP sales.srctab, TARGET sales.targtab2, &
COLMAP (targcol2 = lookup.param2);
```

Example 4 The following is an example of using SOURCEROW. In this case, the second map returns a valid value because the procedure executes on every source row operation.

```
MAP sales.srctab, TARGET sales.targtab, &
SQLEXEC (SPNAME lookup, PARAMS (param1 = srccol), EXEC SOURCEROW), &
COLMAP (targcol = lookup.param2);

MAP sales.srctab, TARGET sales.targtab2, &
COLMAP (targcol2 = lookup.param2);
```

Using ID

Use ID for queries and stored procedures within a MAP statement as follows.

- For a query, use ID <logical name> so that a name can be used by Oracle GoldenGate to reference the column values returned by the query.
- For a stored procedure, use ID <logical name> to invoke the procedure multiple times within a MAP statement, for example for two different column maps. Otherwise, it is not required. Up to 20 stored procedures can be executed per MAP statement. They execute in the order listed in the parameter file.

Syntax

```
ID <logical name>
```

Component	Description
`<logical name>`	A logical name for the procedure or query. For example, logical names for a procedure named "lookup" might be "lookup1," "lookup2," and so forth.

Example 1 The following example illustrates the use of ID <logical name>. It enables each column map to call a stored procedure named lookup separately and refer to its own results by means of lookup1 and lookup2.

```
MAP sales.srctab, TARGET sales.targtab, &
SQLEXEC (SPNAME lookup, ID lookup1, PARAMS (param1 = srccol)), &
COLMAP (targcol1 = lookup1.param2), &
SQLEXEC (SPNAME lookup, ID lookup2, PARAMS (param1 = srccol)), &
COLMAP (targcol = lookup2.param2);
```

Example 2 The following example shows a single execution of a stored procedure named lookup. In this case, the actual name of the procedure is used. A logical name is not needed.

```
MAP sales.tab1, TARGET sales.tab2, &
SQLEXEC (SPNAME lookup), PARAMS (param1 = srccol)), &
COLMAP (targcol = lookup.param1);
```

Example 3 The following examples illustrate the use of ID <logical name> for Oracle and SQL Server queries, respectively. Note that in this illustration, the SQLEXEC statement spans multiple lines due to space constraints in this documentation. An actual SQLEXEC statement must be contained on one line only.

```
MAP sales.account, TARGET sales.newacct,
SQLEXEC (ID lookup,
QUERY “select desc_col into desc_param from lookup_table
where code_col = :code_param”,
PARAMS (code_param = account_code)),
COLMAP (newacct_id = account_id,
newacct_val = lookup.desc_param);

MAP sales.account, TARGET sales.newacct,
SQLEXEC (ID lookup,
QUERY "select desc_col into desc_param from lookup_table
where code_col = ?",
PARAMS (p1 = account_code)),
COLMAP (newacct_id = account_id,
newacct_val = lookup.desc_param);
```

Using MAXVARCHARLEN

Use MAXVARCHARLEN to specify the maximum length allocated for any output parameter in a stored procedure or query. Beyond this maximum, output values are truncated.

Syntax

```
MAXVARCHARLEN <num bytes>
```

Component	Description
`<num bytes>`	Defines the maximum number of bytes allowed for an output parameter. The default is 255 bytes without an explicit MAXVARCHARLEN clause.

Example

```
MAXVARCHARLEN 100
```

Using NOPARAMS

Use NOPARAMS instead of PARAMS if a stored procedure or query does not require parameters. Either a PARAMS clause or NOPARAMS is required.

Syntax

```
NOPARAMS
```

Example

```
SQLEXEC (SPNAME check, NOPARAMS)
```

Using PARAMBUFSIZE

Use PARAMBUFSIZE to specify the maximum size of the memory buffer that stores parameter information, including both input and output parameters. Oracle GoldenGate issues a warning whenever the memory allocated for parameters is within 500 bytes of the maximum.

Syntax

```
PARAMBUFSIZE <num bytes>
```

Component	Description
`(<num bytes>)`	Defines the maximum number of bytes allowed for the memory buffer. The default is 10,000 bytes without an explicit PARAMBUFSIZE clause.

Example

```
PARAMBUFSIZE 15000
```

Using PARAMS

Use PARAMS to supply the names of parameters in a stored procedure or query that accept input and the name of a source column or Oracle GoldenGate column-conversion function that is supplying the input. Either a PARAMS clause or NOPARAMS is required.

By default, Oracle GoldenGate treats a string that is enclosed within double quotes as a literal. To use double quotes for column names and single quotes for literals (SQL-92 rules), use the USEANSISQLQUOTES parameter in the GLOBALS parameter file.

The following are the databases that are supported by SQLEXEC and the data types that are supported for input and output parameters.

- Numeric data types
- Date data types
- Character data types

By default, output parameters are truncated at 255 bytes per parameter. If a procedure requires longer parameters, use the MAXVARCHARLEN option.

Syntax

```
PARAMS (
[OPTIONAL | REQUIRED]
<param name> = {<source column> | <GG function>}
[, ...]
)
```

Component	Description
`OPTIONAL | REQUIRED`	Determines whether or not the procedure or query executes when parameter values are missing. ◆ 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. OPTIONAL is the default for all databases except Oracle. For Oracle, whether or not a parameter is optional is automatically determined when retrieving the stored procedure definition. ◆ REQUIRED indicates that a parameter value must be present. If the parameter value is not present, the SQL will not be executed.

`<param name> = {`
`<source column> |`
`<GG function`
`}`
	Maps the parameter name to the column or function that provides the input.
	Where:
	◆ <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 P*n*, 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. Consider whether the database requires the “p” to be upper or lower case.
	◆ <source column> is the name of a source column. By default, if the specified column is not present in the log (because it is a compressed update) the parameter assumes any default value specified by the procedure or query for the parameter.
	◆ <GG function> is the name of an Oracle GoldenGate column-conversion function. For more information about column-conversion functions, see Chapter 4.

Example The following example maps data from the account table to the target table newacct. When processing records from the account table, Oracle GoldenGate executes the lookup stored procedure before executing the column map. The code_param parameter in the procedure accepts input from the account_code source column.

```
MAP sales.account, TARGET sales.newacct, &
SQLEXEC (SPNAME lookup, PARAMS (code_param = account_code)), &
COLMAP (newacct_id = account_id, &
newacct_val = lookup.desc_param);
```

Using TRACE

Use TRACE to log input and output parameters to the report file.

Sample discard file with SQLEXEC tracing enabled:

```
Input parameter values...

LMS_TABLE: INTERACTION_ATTR_VALUES
  KEY1: 2818249
  KEY2: 1
Report File:

From Table MASTER.INTERACTION_ATTR_VALUES to
MASTER.INTERACTION_ATTR_VALUES:
       #  inserts:      0
       #  updates:      0
       #  deletes:      0
       #  discards:     1

  Stored procedure GGS_INTERACTION_ATTR_VALUES:
       attempts:        2
       successful:      0
```

Syntax

```
TRACE {ALL | ERROR}
```

Action	Description
ALL	Writes the input and output parameters for each invocation of the procedure or query to the report file. This is the default.
ERROR	Writes the input and output parameters for each invocation of the procedure or query to the report file only after a SQL error occurs.

Example

```
SQLEXEC (SPNAME lookup, PARAMS (long_name = birth_state,
short_name = state), TRACE ERROR)
```

Using TARGETDEF

Use TARGETDEF to specify a target-definitions template. The definitions template is created based on the definitions of a specific target table when DEFGEN is run for that table. Once the template is created, new target tables that have identical definitions to that table can be added without having to run DEFGEN for them, and without having to stop and start Replicat. The definitions in the template specified with TARGETDEF will be used for definitions lookups. For more information about DEFGEN, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax
```
MAP <table spec>, TARGET <table spec>, TARGETDEF <definitions template>;
```

Argument	Description
`<definitions template>`	The name of a definitions template that was specified with the DEF option of TABLE in the DEFGEN parameter file. The definitions contained in the template must be identical to the definitions of the table in this MAP statement.

Example
```
MAP acct.cust*, TARGET acc.cust*, DEF custdef, TARGETDEF tcustdef;
```

Using TRIMSPACES and NOTRIMSPACES

Use TRIMSPACES and NOTRIMSPACES to control whether or not trailing spaces in a source CHAR column are truncated when applied to a target CHAR or VARCHAR column. The default is TRIMSPACES.

NOTE Sybase treats all CHAR types as VARCHAR types, and therefore TRIMSPACES will have no effect. For Sybase, use the TRIMVARSPACES parameter.

TRIMSPACES and NOTRIMSPACES also can be used at the root level of a parameter file to turn the trim feature on or off for different MAP statements or groups of statements.

Syntax
```
MAP <table spec>, TARGET <table spec>, {TRIMSPACES | NOTRIMSPACES};
```

Example The following keeps the default of trimming trailing spaces for the first two targets, but does not trim spaces for the last two targets.

```
MAP fin.src1, TARGET fin.tgt1;
MAP fin.src1, TARGET fin.tgt2;
MAP fin.src1, TARGET fin.tgt3, NOTRIMSPACES;
MAP fin.src1, TARGET fin.tgt4, NOTRIMSPACES;
```

Using TRIMVARSPACES and NOTRIMVARSPACES

Use TRIMVARSPACES and NOTRIMVARSPACES to control whether or not trailing spaces in a source VARCHAR column are truncated when applied to a target CHAR or VARCHAR column.

The default is NOTRIMVARSPACES because spaces in a VARCHAR column could actually be part of the data. Before using TRIMVARSPACES, make certain that trailing spaces are not an integral part of the target data.

TRIMVARSPACES and NOTRIMVARSPACES also can be used at the root level of a parameter file to turn the trim feature on or off for different MAP statements or groups of statements.

Syntax
```
MAP <table spec>, TARGET <table spec>, {TRIMVARSPACES | NOTRIMVARSPACES};
```

Example The following trims trailing spaces only for the last two targets.

```
MAP fin.src1, TARGET fin.tgt1;
MAP fin.src1, TARGET fin.tgt2;
MAP fin.src1, TARGET fin.tgt3, TRIMVARSPACES;
MAP fin.src1, TARGET fin.tgt4, TRIMVARSPACES;
```

Using WHERE

Use WHERE to select records based on a conditional statement. For more information on using a WHERE clause and the column data that can be used, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Oracle GoldenGate does not support WHERE for columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.

Literal strings used in WHERE must be enclosed within double quotes unless the USEANSISQLQUOTES parameter is used in the GLOBALS file. This parameter enforces SQL-92 rules for identifiers and literals.

Syntax

```
MAP <table spec>, TARGET <table spec>,
WHERE (<where clause>);
```

Component	Description
`<where clause>`	Selects records based on a condition, such as: `WHERE (branch = "NY")` The following table shows permissible WHERE operators. WHERE does not support evaluating the before image of a primary key column in the conditional statement as part of a primary key update operation.

Table 37 Permissible WHERE operators

Operator	Example
Column names	`PRODUCT_AMT`
Numeric values	`-123, 5500.123`
Literal strings enclosed in quotes	`"AUTO", "Ca"`
Column tests	`@NULL`, `@PRESENT`, `@ABSENT` (column is null, present or absent in the record). These tests are built into Oracle GoldenGate.
Comparison operators	`=, <>, >, <, >=, <=`
Conjunctive operators	`AND, OR`

Table 37 Permissible WHERE operators

Operator	Example
Grouping parentheses	Use open and close parentheses for logical grouping of multiple elements.

Example The following WHERE 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)
```

MAPEXCLUDE

Valid for Replicat

Use the MAPEXCLUDE parameter with the MAP parameter to explicitly exclude tables from a wildcard specification. MAPEXCLUDE must precede all MAP statements that contain the table or tables being excluded.

Default None

Syntax `MAPEXCLUDE <exclude specification>`

Argument	Description
`<exclude specification>`	The name or wildcard specification of the table to exclude. The same wildcard rules apply for MAPEXCLUDE as for specifying tables with wildcards in a MAP statement.

Example In the following example, the MAP statement retrieves all tables except for the table named TEST.

```
MAPEXCLUDE fin.TEST
MAP fin.*, TARGET fin.*;
```

MARKERTABLE

Valid for GLOBALS

Use the MARKERTABLE parameter to specify the name of the DDL marker table that supports Oracle DDL synchronization, if other than the default of GGS_MARKER. The marker table stores information about DDL operations. This parameter is only valid for Oracle.

The name of the marker table must also be specified with the marker_table_name parameter in the params.sql script. This script resides in the root Oracle GoldenGate installation directory.

For more information about the marker table and params.sql, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Default `GGS_MARKER`

Syntax `MARKERTABLE <table_name>`

Argument	Description
`<table_name>`	The name of the marker table.

MAXDISCARDRECS

Valid for Replicat

Use the MAXDISCARDRECS parameter to limit the number of errors reported to the discard file per MAP statement.

Use this parameter for the following reasons:

- When you expect a large number of errors but do not want them reported.
- To manage the size of the discard file.

This parameter is table-specific and applies to all subsequent MAP statements. More than one instance of MAXDISCARDRECS can be used in a parameter file.

Default None

Syntax `MAXDISCARDRECS <number>`

Argument	Description
`<number>`	The maximum number of errors to report.

Example `MAXDISCARDRECS 1000`

MAXFETCHSTATEMENTS

Valid for Extract

Use the MAXFETCHSTATEMENTS parameter to control the maximum allowable number of prepared queries that can be used by Extract to fetch row data from an Oracle source database. The fetched data is used when not enough information is available to construct a logical SQL statement from a transaction log record.

Queries are prepared and cached as needed. When the value set with MAXFETCHSTATEMENTS is reached, the oldest query is replaced by the newest one. The value of this parameter controls the number of open cursors maintained by Extract for fetch queries only. Additional cursors may be used by Extract for other purposes, such as those required for stored procedures. This parameter is only valid for Oracle databases.

Default 100

Syntax `MAXFETCHSTATEMENTS <number>`

Argument	Description
`<number>`	The maximum number of cursors that Extract will use for prepared queries. Make certain that the database can support the number of cursors specified with MAXFETCHSTATEMENTS, plus cursors used by other applications and processes.

Example `MAXFETCHSTATEMENTS 150`

MAXGROUPS

Valid for GLOBALS

Use the MAXGROUPS parameter to specify the maximum number of process groups that will be permitted in an instance of Oracle GoldenGate. The Manager process checks this parameter to determine its resource allocations. The GGSCI process checks this parameter to control the maximum number of groups that it allows to be created.

The actual number of processes that can run on a given system depends on the system resources that are available. If system resources are exceeded, errors will be returned regardless of the setting of MAXGROUPS.

Default 300 groups maximum

Syntax `MAXGROUPS <number_of_groups>`

Argument	Description
`<number_of_groups>`	The number of groups to be allowed in the instance of Oracle GoldenGate. Specify a value from 300 to 5000.

Example `MAXGROUPS 600`

MAXSQLSTATEMENTS

Valid for Replicat

Use the MAXSQLSTATEMENTS parameter to control the number of prepared SQL statements that can be used by Replicat both in regular processing mode and in BATCHSQL mode (see page 129). The value for MAXSQLSTATEMENTS determines the number of open cursors that Replicat maintains. Make certain that the database can support the specified number of cursors, plus the cursors used by other applications and processes. Before changing MAXSQLSTATEMENTS, contact Oracle Support. For more information, go to http://support.oracle.com.

MAXSQLSTATEMENTS requires the DYNSQL parameter to be enabled. DYNSQL is the default.

Default 250 cursors

Syntax `MAXSQLSTATEMENTS <number>`

Argument	Description
`<number>`	The maximum number of cursors that Replicat will use. The maximum value is 250. The minimum is 1.

Example `MAXSQLSTATEMENTS 200`

MAXTRANSOPS

Valid for Replicat

Use the MAXTRANSOPS parameter to split large source transactions into smaller ones on the target system. This parameter can be used when the target database is not configured to accommodate large transactions. For example, if the Oracle rollback segments are not large enough on the target to reproduce a source transaction that performs one million deletes, you could specify MAXTRANSOPS 10000, which forces Replicat to issue a commit after each group of 10,000 deletes.

Limitations of use

You should have a valid business case for using MAXTRANSOPS other than simply for the purpose of expediency. To use MAXTRANSOPS is to alter the transactional integrity that is imposed by the boundaries that are defined by the source application, even though Replicat applies the operations in the correct order.

By default, when Extract recovers from a failure, it re-sends the entire source transaction that it was processing at the time of the failure, and it appends this transaction to the end of the trail file instead of overwriting the old data. The new transaction is flagged by a restart record, alerting Replicat that it must roll back and start the transaction over again. However, if MAXTRANSOPS has forced Replicat to split apart that transaction, Replicat can only roll back what it has not yet committed to the target database. When Replicat processes the committed operations again, they will result in duplicate-row errors or missing-row errors, depending on the SQL operation type.

To avoid these conditions while using MAXTRANSOPS, you can set the RECOVERYOPTIONS parameter to OVERWRITEMODE to configure Extract so that it overwrites the old transaction data with the new data. However, this mode can make recovery more difficult after certain failure conditions, because the overwrites can corrupt the trail records. In these cases, it is recommended that you open a service request with Oracle. For more information, go to http://support.oracle.com.

NOTE When troubleshooting Replicat abend errors, Oracle Support may request GROUPTRANSOPS to be set to 1 and MAXTRANSOPS to be set to 1. This is only a temporary configuration for troubleshooting purposes and should not be used permanently in production, or it will cause data integrity errors.

Default 100,000,000

Syntax `MAXTRANSOPS <transaction count>`

Argument	Description
`<transaction count>`	The number of operations to portion into a single transaction group.

Example `MAXTRANSOPS 10000`

MGRSERVNAME

Valid for GLOBALS

Use the MGRSERVNAME parameter in a GLOBALS parameter file to specify the name of the Manager process when it is installed as a Windows service. This parameter is only required when installing multiple instances of Manager as a service on the same system, for example when installing multiple Oracle GoldenGate instances or when also installing the Oracle GoldenGate Veridata Agent, which uses a Manager process.

There must be a GLOBALS file containing MGRSERVNAME for each Manager service that is installed. The files must be created *before the services are installed*, because the installer references MGRSERVNAME when registering the service name on the system.

Default None

Syntax `MGRSERVNAME <name>`

Argument	Description
`<name>`	A one-word name for the Manager service.

Example `MGRSERVNAME GoldenGate`

NAMEMATCHIGNORECASE | NAMEMATCHNOWARNING | NAMEMATCHEXACT

Valid for GLOBALS

Use these parameters to control the behavior of fallback name mapping. Fallback name mapping is enabled by default when the source database is case-sensitive and the target database supports both case-sensitive and case-insensitive object names, such as Oracle, DB2 and SQL/MX.

Fallback name matching works as follows: When a source table name is case-sensitive, Oracle GoldenGate applies case-sensitive wildcard mapping on the target database to find an exact match. If the target database does not contain the exact target table name, including case, fallback name mapping performs a case-insensitive target table mapping to find a name match. This behavior is controlled by the NAMEMATCHIGNORECASE parameter.

To allow fallback name mapping, but output a warning message, use the NAMEMATCHNOWARNING parameter.

To disable fallback name matching, use the NAMEMATCHEXACT parameter. With NAMEMATCHEXACT, if an exact, case-sensitive match is not found, Oracle GoldenGate returns an error and abends.

Default `NAMEMATCHIGNORECASE`

Syntax `NAMEMATCHIGNORECASE | NAMEMATCHNOWARNING | NAMEMATCHEXACT`

NOHEADERS

Valid for Extract

Use the NOHEADERS parameter to indicate that an extract file contains no record headers. In such cases, Replicat assumes that the input file contains only insert records pertaining to a single table and that each record is of the same length and type.

The FORMATASCII parameter with the NOHDRFIELDS option must be used in the Extract parameter file when using NOHEADERS. When NOHEADERS is used, only one source table can be specified across all MAP statements.

Default None

Syntax `NOHEADERS`

NUMFILES

Valid for Extract and Replicat

Use the NUMFILES parameter to control the initial number of memory structures allocated to contain information about tables specified in TABLE or MAP statements. NUMFILES must occur before any TABLE or MAP entries, and before the SOURCEDEFS or TARGETDEFS parameter, to have any effect.

To control the number of additional memory structures that are allocated dynamically once the NUMFILES value is reached, use the ALLOCFILES parameter (see page 123). The default values should be sufficient for both NUMFILES and ALLOCFILES, because memory is allocated by the process as needed, system resources permitting.

Default 1000

Syntax `NUMFILES <number of structures>`

Argument	Description
`<number of structures>`	The number of memory structures to be allocated. Do not set NUMFILES to an arbitrarily high number, or memory will be consumed unnecessarily. The memory of Oracle GoldenGate supports up to two million tables.

Example `NUMFILES 4000`

OBEY

Valid for Extract and Replicat

Use the OBEY parameter to retrieve parameter settings from a file other than the current parameter file.

To use OBEY

1. Create and save a parameter file containing the parameters that you want to retrieve. You can create a library of text files that contain different frequently used parameter settings.
2. Use OBEY in the active parameter file to invoke the other file. OBEY statements cannot be nested within other OBEY statements. 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.

Instead of using OBEY, or in addition to using it, you can use Oracle GoldenGate macros to call frequently used parameters. For more information about using macros, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Default None

Syntax

```
OBEY <file name>
```

Argument	Description
`<file name>`	The relative or fully qualified name of the file from which to retrieve parameters or commands.

```
OBEY /home/ggs/myparams
```

OUTPUTFILEUMASK

Valid for GLOBALS

Use the OUTPUTFILEUMASK parameter to specify an octal umask that will be used by Oracle GoldenGate processes to create trail files and discard files. OUTPUTFILEUMASK is not valid for WIN32 systems.

Default Umask of 0 (all privileges)

Syntax

```
OUTPUTFILEUMASK <umask>
```

Argument	Description
`<umask>`	The umask value. Must be between 0 and 0777; otherwise there will be an error: `"Missing or invalid option for OUTPUTFILEUMASK."`

Example

```
OUTPUTFILEUMASK 066
```

OVERRIDEDUPS | NOOVERRIDEDUPS

Valid for Replicat

Use the OVERRIDEDUPS and NOOVERRIDEDUPS parameters to control whether or not Replicat overwrites an existing record in the target database with a replicated one if both records have the same key.

- OVERRIDEDUPS overwrites the existing record. It can be used for initial loads in which you do not want to truncate target tables beforehand, or for the resynchronization of a target table with a trusted source.
- NOOVERRIDEDUPS, the default, does not overwrite the existing record, but instead generates a duplicate-record error. You can use an exceptions MAP statement to execute a SQL procedure with a SQLEXEC clause to initiate a response to the error. Otherwise, the transaction may abend. For more information about exceptions maps, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

To bypass duplicate records without causing Replicat to abend when an exceptions map is not available, specify a REPERROR parameter statement similar to the following, where <duplicate key error> is the database error number for primary key constraint errors.

```
REPERROR (<duplicate key error>, IGNORE)
```

For example, the statement for an Oracle database would be:

```
REPERROR (1, IGNORE)
```

Duplicate records are output to the discard file.

OVERRIDEDUPS and NOOVERRIDEDUPS are specific to a TABLE or MAP statement, so you can create different rules for each table or group of tables. Use the SQLDUPERR parameter (see page 332) with OVERRIDEUPS to specify the numeric error code that is returned by the database for duplicate inserts.

OVERRIDEDUPS is automatically enabled when HANDLECOLLISIONS is specified.

When OVERRIDEDUPS is in effect, records might not be processed in chronological order across all Replicat processes.

Default NOOVERRIDEDUPS

Syntax OVERRIDEDUPS | NOOVERRIDEDUPS

PASSTHRU | NOPASSTHRU

Valid for Extract

Use the PASSTHRU and NOPASSTHRU parameters to control whether a data-pump Extract processes tables in pass-through mode or normal mode. In pass-through mode, the Extract process does not look up table definitions, either from the database or from a data-definitions file. Normally, the Extract process logs into the database to retrieve data definitions and, if the target is NonStop, reads a data-definitions file. The definitions are used to perform mapping and conversion functions.

Using pass-through mode, you can cascade the captured data to a data pump on an intermediary system that has no database installed on it. Source and target table names and structures must be identical; no filtering, column mapping, SQLEXEC functions, transformation, or other functions requiring data manipulation or translation can be used. The WILDCARDRESOLVE parameter must be set to DYNAMIC (the default).

The PASSTHRU and NOPASSTHRU parameters are table-specific. One parameter remains in effect for all subsequent MAP statements and trails, until the other parameter is encountered. This enables you to specify pass-through behavior for one set of tables while using normal processing, including data manipulation, for other tables. For the tables requiring manipulation, a source definitions file is required if filtering is to be performed, and a target definitions file is required if column mapping or conversion is to be performed. These files provide the metadata needed by Oracle GoldenGate to perform those actions.

In PASSTHRU mode, the data pump will not perform automatic ASCII-to-EBCDIC or EBCDIC-to-ASCII conversion.

PASSTHRU in DDL replication

DDL is propagated through a data pump or VAM-sort Extract in PASSTHRU mode automatically. 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.

Default NOPASSTHRU

Syntax PASSTHRU | NOPASSTHRU

Example The following parameter file passes through all data from fin.acct, but allows normal processing for fin.sales.

```
EXTRACT fin
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC, &
   AES128, ENCRYPTKEY securekey
RMTHOST sysb, MGRPORT 7809, ENCRYPT AES192 KEYNAME mykey
ENCRYPTTRAIL AES192 KEYNAME mykey2
RMTTRAIL /ggs/dirdat/rt
PASSTHRU
TABLE fin.acct;
NOPASSTHRU
TABLE fin.sales, WHERE (ACCOUNT-CODE < 100);
```

PASSTHRUMESSAGES | NOPASSTHRUMESSAGES

Use the PASSTHRUMESSAGES and NOPASSTHRUMESSAGES parameters to control whether or not messages for tables being processed in pass-through mode are written to the Extract report file. If enabled, messages similar to the following are written:

```
"PASSTHRU mapping resolved for source table <table name>"
```

Default PASSTHRUMESSAGES

Syntax PASSTHRUMESSAGES | NOPASSTHRUMESSAGES

PORT

Valid for Manager

Use the PORT parameter to specify a TCP/IP port number for the Manager process on which to interact with remote processes requesting dynamic services, typically either an initial-load Replicat or the Collector process. Use the default port number when possible.

Default Port 7809

Syntax `PORT <number>`

Argument	Description
`<number>`	An available port number.

Example `PORT 7809`

PURGEDDLHISTORY

Valid for Manager

Use the PURGEDDLHISTORY parameter to control the size of the DDL history table in an Oracle database by purging rows. Use caution when purging the history table. It is critical to the integrity of the DDL synchronization processes and must not be purged prematurely, because the purges are non-recoverable through Oracle GoldenGate. To prevent any possibility of permanent DDL data loss, make regular backups of the history table.

You can specify maximum and minimum lengths of time to keep a row, based on the last modification date. Both maximum and minimum rules should be specified; otherwise Manager does not have a complete criteria for when to delete the row. For example, MINKEEPHOURS 3 used with MAXKEEPHOURS 5 specifies to keep rows that have not been modified in the past three hours, but delete them when they have not been modified for at least five hours.

This parameter does not require you to supply a table name. Oracle GoldenGate first looks for a name specified with the DDLTABLE <table> parameter in the GLOBALS file or, if that parameter does not exist, Oracle GoldenGate uses the default name of GGS_DDL_HIST.

NOTE For additional information about purging the DDL history table, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

PURGEDDLHISTORY requires a logon to be specified with the USERID parameter and, if required, the SOURCEDB parameter.

This parameter is only valid for Oracle.

Default Purge every hour

Syntax

```
PURGEDDLHISTORY
{, <max rule>}
[, <min rule>]
[, <frequency>]
```

Argument	Description
`<max rule>`	Required. Can be one of the following to set the maximum amount of time to keep rows. `MAXKEEPHOURS <n>` Purges if the row has not been modified for <n> number of hours.

Argument	Description
	`MAXKEEPDAYS <n>` Purges if the row has not been modified for <n> number of days.

`<min rule>`	Optional, but recommended. Can be one of the following to set the minimum amount of time to keep rows.
	`MINKEEPHOURS <n>`
	Keeps an unmodified row for at least the specified number of hours.
	`MINKEEPDAYS <n>`
	Keeps an unmodified row for at least the specified number of days.

`<frequency>`	Sets the frequency with which to purge DDL history. The default time for Manager to process maintenance tasks is 10 minutes, as specified with the CHECKMINUTES parameter (see page 148). Every 10 minutes, Manager evaluates the PURGEOLDEXTRACTS frequency and conducts the purge after the specified interval. <frequency> can be one of the following:
	`FREQUENCYMINUTES <n>`
	Sets the frequency, in minutes, with which to purge DDL history. The default purge frequency is 60 minutes.
	`FREQUENCYHOURS <n>`
	Sets the frequency, in hours, at which to purge DDL history.

Example The following example keeps all rows that have not been modified in the past three days and deletes them when they have not been modified for at least five days. The purge frequency is 30 minutes.

```
PURGEDDLHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 5, FREQUENCYMINUTES 30
```

PURGEDDLHISTORYALT

Valid for Manager

Use the PURGEDDLHISTORYALT parameter to control the size of the internal DDL history table in an Oracle database that tracks partitioned object IDs that are associated with the object ID of a given table. This parameter purges rows that are not needed any more.

Use caution when purging this table. The purges are non-recoverable through Oracle GoldenGate. To prevent any possibility of permanent DDL data loss, make regular backups of the history table.

This parameter does not require a table name as input. The default name is either GGS_DDL_HIST_ALT or the custom name that is specified with the DDLTABLE <table> parameter in the GLOBALS file, if used.

For syntax options and usage, see “PURGEDDLHISTORY” on page 289.

PURGEMARKERHISTORY

Valid for Manager

Use the PURGEMARKERHISTORY parameter to control the size of the Oracle GoldenGate marker table by purging rows. You can purge the marker table at any time.

This parameter does not require you to supply a table name. Oracle GoldenGate first looks for a name specified with the MARKERTABLE <table> parameter in the GLOBALS file or, if that parameter does not exist, Oracle GoldenGate uses the default name of GGS_MARKER.

You can specify maximum and minimum lengths of time to keep a row, based on the last modification date. Both maximum and minimum rules should be specified; otherwise Manager does not have complete criteria for when to delete the row. For example, MINKEEPHOURS 3 used with MAXKEEPHOURS 5 specifies to keep rows that have not been modified in the past three hours, but delete them when they have not been modified for at least five hours.

NOTE For additional information about purging the marker table, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

PURGEMARKERHISTORY requires a logon with the USERID parameter and, if required, the SOURCEDB parameter.

Default Purge every hour

Syntax

```
PURGEMARKERHISTORY
{, <max rule>}
[, <min rule>]
[, <frequency>]
```

Argument	Description

`<max rule>`	Required. Can be one of the following to set the maximum amount of time to keep rows.
	`MAXKEEPHOURS <n>`
	Purges if the row has not been modified for <n> number of hours.
	`MAXKEEPDAYS <n>`
	Purges if the row has not been modified for <n> number of days.

`<min rule>`	Optional, but recommended. Can be one of the following to set the minimum amount of time to keep rows.
	`MINKEEPHOURS <n>`
	Keeps an unmodified row for at least the specified number of hours.
	`MINKEEPDAYS <n>`
	Keeps an unmodified row for at least the specified number of days.

Argument	Description

`<frequency>`	Sets the frequency with which to purge marker history. The default time for Manager to process maintenance tasks is 10 minutes, as specified with the CHECKMINUTES parameter (see page 148). Every 10 minutes, Manager evaluates the PURGEOLDEXTRACTS frequency and conducts the purge after the specified interval. <frequency> can be one of the following:
	`FREQUENCYMINUTES <n>`
	Sets the frequency, in minutes, with which to purge marker history. The default purge frequency is 60 minutes.
	`FREQUENCYHOURS <n>`
	Sets the frequency, in hours, at which to purge marker history.

Example The following example keeps all rows that have not been modified in the past three days and deletes them when they have not been modified for at least five days. The purge frequency is 30 minutes.

```
PURGEMARKERHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 5, FREQUENCYMINUTES 30
```

PURGEOLDEXTRACTS

Valid for Manager, Extract, and Replicat

The implementation of this parameter varies, depending on the process.

PURGEOLDEXTRACTS for Extract and Replicat

Use the PURGEOLDEXTRACTS parameter in an Extract or Replicat parameter file to delete old trail files whenever Oracle GoldenGate starts processing from a new one. Preventing the accumulation of trail files conserves disk space. Purges are conducted after the process is done with the file as indicated by checkpoints.

Purging by Extract is appropriate if the process is a data pump. After the data is sent to the target system, the files can be purged. Otherwise, purging would ordinarily be done by Replicat.

PURGEOLDEXTRACTS should only be used in an Extract or Replicat parameter file if there is only one instance of the process. If multiple groups are reading the same set of trail files, one process could purge a file before another is finished with it. Instead, use the Manager version of PURGEOLDEXTRACTS, which is the preferred use of the parameter in all Oracle GoldenGate configurations because it allows you to manage trail files in a centralized fashion.

Default Purge the trail file when moving to the next file in the sequence.

Syntax `PURGEOLDEXTRACTS`

PURGEOLDEXTRACTS for Manager

Use the PURGEOLDEXTRACTS parameter in a Manager parameter file to purge 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. As a Manager parameter, PURGEOLDEXTRACTS allows you to manage trail files in a centralized fashion and take into account multiple processes.

How to use this parameter

To control the purging, follow these rules:

- Use USECHECKPOINTS to purge when all processes are finished with a file as indicated by checkpoints. This is the default, but it can be turned off with the NOUSECHECKPOINTS option. Basing the purging on checkpoints ensures that no data is deleted until all processes are finished with it. USECHECKPOINTS is checked whether or not the parameter is explicitly defined with PURGEOLDEXTRACTS, unless there is an explicit NOUSECHECKPOINTS entry. Basing purges on checkpoints is essential in a production environment to ensure data integrity. USECHECKPOINTS considers the checkpoints of both Extract and Replicat before purging.
- Use the MINKEEP rules to set a minimum amount of time to keep unmodified data:
    - Use MINKEEPHOURS or MINKEEPDAYS to keep data for <n> hours or days.
    - Use MINKEEPFILES to keep at least <n> trail files including the active file. The default is 1.

  Use only *one* of the MINKEEP options. If more than one is used, Oracle GoldenGate selects one of them based on the following:

    - If both MINKEEPHOURS and MINKEEPDAYS are specified, only the last one is accepted, and the other will be ignored.
    - If either MINKEEPHOURS or MINKEEPDAYS is used with MINKEEPFILES, then MINKEEPHOURS or MINKEEPDAYS is accepted, and MINKEEPFILES is ignored.

Manager purges based on the value set for the CHECKMINUTES parameter (see page 148). When that value is reached, the purge rules are evaluated as follows:

1. USECHECKPOINTS only. If no MINKEEP rules are specified, and USECHECKPOINTS is enabled, the minimum number of files to keep is 1. If checkpoints indicate that a file has been processed, that file will be purged unless it would fall below the one-file minimum.
2. USECHECKPOINTS with MINKEEP rules. If USECHECKPOINTS is enabled and checkpoints indicate that a file has been processed, it will be purged unless doing so would violate the applicable MINKEEP rules.
3. NOUSECHECKPOINTS only. If there are no MINKEEP rules and NOUSECHECKPOINTS is specified, then checkpoints are not considered and the file will be purged unless doing so will violate the default rule to keep one file.
4. NOUSECHECKPOINTS with MINKEEP rules. If there are MINKEEP rules and NOUSECHECKPOINTS is specified, a file will be purged unless doing so will violate the MINKEEP rule.

Manager determines which files to purge based on Extract and Replicat processes configured on the local system. If at least one process reads a trail file, Manager applies the specified rules; otherwise, the rules do not take effect.

Additional guidelines for PURGEOLDEXTRACTS for Manager

- Do not use more than 500 PURGEOLDEXTRACTS parameter statements in the same Manager parameter file.
- When using this parameter, do not permit trail files to be deleted by any user or program other than Oracle GoldenGate. It will cause PURGEOLDEXTRACTS to function improperly.
- When trails are stored on NFS, there is a difference in system time between the NFS drive and the local system time where Manager is running. The trail is created with the NFS time, but the timestamps of the records in the trail are compared with the local system time to determine whether to purge them or not. Take into account any time differences when you create your MINKEEP rules.

Default `USECHECKPOINTS`

Syntax

```
PURGEOLDEXTRACTS <trail name>
[, USECHECKPOINTS | NOUSECHECKPOINTS]
[, <minkeep rule>]
[, <frequency>]
```

Argument	Description
`<trail name>`	The trail to purge. Use a relative or fully qualified name.
`USECHECKPOINTS`	Allows purging after all Extract and Replicat processes are done with the data as indicated by checkpoints, according to any MINKEEP rules.
`NOUSECHECKPOINTS`	Allows purging without considering checkpoints, based on keeping a minimum of either: ◆ one file if no MINKEEP rule is used or... ◆ the number of files specified with a MINKEEP rule.

`<MINKEEP rule>`	Can be one of the following to set rules for the minimum amount of time to keep data.
	`MINKEEPHOURS <n>`
	Keeps an unmodified file for at least the specified number of hours.
	`MINKEEPDAYS <n>`
	Keeps an unmodified file for at least the specified number of days.
	`MINKEEPFILES <n>`
	Keeps at least <n> unmodified trail files, including the active file.

`<frequency>`	Sets the frequency with which to purge old trail files. The default time for Manager to process maintenance tasks is 10 minutes, as specified with the CHECKMINUTES parameter (see page 148). Every 10 minutes, Manager evaluates the PURGEOLDEXTRACTS frequency and conducts the purge after the specified interval. <frequency> can be one of the following:

Argument	Description
	`FREQUENCYMINUTES <n>` Sets the frequency, in minutes, with which to purge old trail files. The default purge frequency is 60 minutes.
	`FREQUENCYHOURS <n>` Sets the frequency, in hours, at which to purge old trail files.

Example 1 Status: Trail files AA000000, AA000001, and AA000002 exist. Replicat has been stopped for four hours and is not finished processing any of the files. The Manager parameters include:

```
PURGEOLDEXTRACTS /ggs/dirdat/AA*, USECHECKPOINTS, MINKEEPHOURS 2
```

Result: The amount of time that unmodified files must be retained was exceeded. No files will be purged, however, because checkpoints indicate that Replicat is not finished processing them.

Example 2 Status: Trail files AA000000, AA000001, and AA000002 exist. Replicat has been stopped for four hours and is not finished processing any of the files. The Manager parameters include:

```
PURGEOLDEXTRACTS /ggs/dirdat/AA*, NOUSECHECKPOINTS, MINKEEPHOURS 2
```

Result: All of the trail files will be purged because the minimum time to keep them was satisfied.

Example 3 Status: Replicat and Extract are finished processing data. There has been no access to the trail files for the last five hours. Trail files AA000000, AA000001, and AA000002 exist. The Manager parameters include:

```
PURGEOLDEXTRACTS /ggs/dirdat/AA*, USECHECKPOINTS, MINKEEPHOURS 4, &
MINKEEPFILES 4
```

Result: This is an example of why only one of the MINKEEP options should be set. USECHECKPOINTS requirements were satisfied, so the minimum rules are considered when deciding whether to purge AA000002. Only two files will remain if AA000002 is purged, and that violates the MINKEEPFILES rule. Because both MINKEEPFILES and MINKEEPHOURS are specified, however, MINKEEPFILES is ignored. The file will be purged because it has not been modified for five hours, and that satisfies the MINKEEPHOURS requirement of four hours.

PURGEOLDTASKS

Valid for Manager

Use the PURGEOLDTASKS parameter to purge Extract and Replicat tasks after a specific amount of time or after they have stopped gracefully. You can indicate when to delete a task according to the following rules:

- The task was last started a specific number of days or hours ago. If the task never was started, then its creation time is used as the basis for applying the rules.
- The task stopped gracefully or never was started. This rule takes precedence over the time the task was last started. Use this rule to prevent abnormally terminated tasks from being purged.

No more than 300 PURGEOLDTASKS parameter statements may be used in the same Manager parameter file.

Default None

Syntax

```
PURGEOLDTASKS <process> <group name>
[, <purge option>]
[USESTOPSTATUS]
```

Argument	Description
`<process>`	Valid values: ◆ `EXTRACT` ◆ `REPLICAT` ◆ `ER` (for both processes)
`<group name>`	The group name or a wildcard to specify multiple groups.
`<purge option>`	Purges if the task has not been updated for a specific number of hours or days. Valid values: ◆ `AFTER <number> DAYS` ◆ `AFTER <number> HOURS`
`USESTOPSTATUS`	Purges if the task was stopped gracefully or never was started.

Example The following example deletes all Extract tasks that have not been updated for at least three days, and it deletes the test_rep Replicat task if it stopped gracefully and has not been updated for at least two hours.

```
PURGEOLDTASKS EXTRACT *, AFTER 3 DAYS
PURGEOLDTASKS REP test_rep, AFTER 2 HOURS, USESTOPSTATUS
```

RECOVERYOPTIONS

Valid for Extract

Use the RECOVERYOPTIONS parameter to control whether Extract overwrites the content of an existing trail file when it restarts, or whether Extract appends new records to the existing data in the file after it restarts.

Append mode

By default, Extract operates in *append mode*, where if there is a process failure, a recovery marker is written to the trail and Extract appends recovery data to the file so that a history of all prior data is retained for recovery purposes.

In append mode, the Extract initialization determines the identity of the last complete transaction that was written to the trail at startup time. With that information, Extract ends recovery when the commit record for that transaction is encountered in the data source; then it begins new data capture with the next committed transaction that qualifies for extraction and begins appending the new data to the trail. A data pump or Replicat starts reading again from that recovery point.

Overwrite mode

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.

In overwrite mode, there is a probability that the overwrite might not deposit precisely the same record images, in precisely the same sequence, as those that are being overwritten. This variation can happen when fetches for large objects must be performed, or when Extract configuration parameters have been changed. Between the time that the overwrite activity begins and the time that the end of the trail file has been reached, any change in Extract processing will create a misalignment at the leading edge of the overwritten portion of the file, where the end of the last record that was rewritten does not fall on a boundary that is shared by the beginning of a record that was written by the previous instance of Extract. A Replicat or data pump that is trying to read from one trail record to the next will land somewhere in the middle of a corrupted record, and it will abend with an error.

Recommendations

Do not change RECOVERYOPTIONS from the default unless instructed to do so by an Oracle Support analyst. If Extract is permitted to overwrite existing data in the trail, it might be harder for Oracle GoldenGate to recover after a failure and it might cause the loss of data that needs to be sent to the target.

In some cases, Extract will automatically revert to overwrite mode to support backward compatibility if the version of Oracle GoldenGate that is being used on the target is older than Oracle GoldenGate version 10. Older versions do not support append mode.

Parameter dependencies

There is a dependency between the RECOVERYOPTIONS parameter and the FORMAT option of EXTFILE, EXTTRAIL, RMTFILE, and RMTTRAIL. When RECOVERYOPTIONS is set to APPENDMODE, the FORMAT option must be set to RELEASE 10.0 or greater. When RECOVERYOPTIONS is set to OVERWRITEMODE, the FORMAT option must be set to RELEASE 9.5 or less.

Default `APPENDMODE`

Syntax `RECOVERYOPTIONS {APPENDMODE | OVERWRITEMODE}`

Argument	Description
`APPENDMODE`	Appends new data to the existing data in a trail file. This is the default.
`OVERWRITEMODE`	Overwrites old data with new data, starting at the most recent checkpoint position.

Example `RECOVERYOPTIONS OVERWRITEMODE`

REPERROR

Valid for Replicat

Use the REPERROR parameter to control how Replicat responds to errors. The default response of Replicat to any error is to abend.

You can use one REPERROR statement to handle most errors in a default manner, while using one or more other REPERROR statements to handle specific errors differently. For example, you can ignore duplicate-record errors but abend processing in all other cases.

In the syntax shown, note that the <error>, <response> specification must be within parentheses. For example:

```
REPERROR (DEFAULT, ABEND)
REPERROR (-1, IGNORE)
```

However, the RESET option cannot be within parentheses:

```
REPERROR RESET
```

Options for record-level error handling

All REPERROR options except TRANSDISCARD and TRANSEXCEPTION apply an error-handling action in response to an individual SQL operation on an individual record. Other, error-free records in the same transaction are processed as configured in the MAP statement and other parameters in the parameter file, as applicable.

Options for transaction-level error handling

The TRANSDISCARD, TRANSEXCEPTION, and ABEND options apply an error-handling action to an entire transaction. The triggering error can occur on an individual record in the transaction or on the commit operation. (Commit errors do not have a particular record associated with them.) These options can be used to:

- prevent an entire source transaction from being replicated to the target when any error is associated with it.
- respond to a commit error when deferred constraint checking is enabled on the target.

TRANSDISCARD and TRANSEXCEPTION are mutually exclusive.

Affect of other parameters on transaction-level options

TRANSDISCARD and TRANSEXCEPTION honor the boundaries of the source transaction; however, the presence of other parameters in the parameter file that alter transaction boundaries may affect the error-handling logic or outcome.

BATCHSQL and GROUPTRANSOPS

BATCHSQL or GROUPTRANSOPS (the default) both group SQL operations from different transactions into larger transactions to improve performance, while maintaining transactional order. When these parameters are in effect and any error occurs, Replicat first tries to resolve it by entering an alternate processing mode (see the documentation for those parameters). If the error persists, TRANSDISCARD or TRANSEXCEPTION comes into effect, and Replicat reverts to source-processing mode as follows:

1. It rolls back the grouped or arrayed transaction.
2. It replays the offending transaction one SQL operation at a time, using the same transaction boundaries as the source transaction.
3. It performs the discard logic (TRANSDISCARD) or exceptions-mapping (TRANSEXCEPTION). (See those option descriptions for more detail.)
4. It resumes BATCHSQL or GROUPTRANSOPS mode after the TRANSDISCARD error handling is completed.

MAXTRANSOPS

The integrity of TRANSDISCARD and TRANSEXCEPTION transaction-level error handling can be adversely affected by the setting of the MAXTRANSOPS parameter. MAXTRANSOPS causes Replicat to split very large replicated source transactions into smaller transactions when it applies them on the target.

The TRANSDISCARD and TRANSEXCEPTION logic cause Replicat to roll back to the first record after the last successful commit. This may or may not be the actual beginning of the offending transaction. It depends on whether that transaction was split up and parts of it are in the previously committed transactions. If that is the case, Replicat cannot apply the TRANSDISCARD or TRANSEXCEPTION action to the whole transaction as it was issued on the source, but only to the part that was rolled back from the target.

If you use MAXTRANSOPS, make certain that it is set to a value that is larger than the largest transaction that you expect to be handled by TRANSDISCARD and TRANSEXCEPTION. This will ensure that transactions are not be split apart into smaller ones on the target.

Affect of transaction-level options on statistics

The output of informational commands in GGSCI, such as STATS REPLICAT, will show the total number of records in the transaction that was processed by TRANSDISCARD or TRANSEXCEPTION logic. This number may reflect the following:

- Replicat writes all records of the transaction to the discard file, including any records that were excluded from Oracle GoldenGate processing by means of a FILTER or WHERE clause.
- If a source table in the transaction has multiple targets, the discarded transaction will contain multiple copies of each record, one for each target.
- Replicat ignores any exceptions mapping statements (as specified with EXCEPTIONSONLY or MAPEXCEPTION) when discarding the transaction.

Replicat abends on errors that are caused by the discard processing (TRANSDISCARD) or exceptions mapping (TRANSEXCEPTION).

Default `TRANSABORT` for deadlocks; `ABEND` for all others

Syntax

```
REPERROR { (
{DEFAULT | DEFAULT2 | <SQL error> | <user-defined error>},
{ABEND | DISCARD | EXCEPTION | IGNORE |
RETRYOP [MAXRETRIES <n>] |
TRANSABORT [, MAXRETRIES] [, DELAYSECS <n> | DELAYCSECS <n>] |
TRANSDISCARD |
TRANSEXCEPTION
}) |
RESET }
```

Table 38 Error specification

Argument	Description
`DEFAULT`	Sets a global response to all errors except those for which explicit REPERROR statements are specified.
`DEFAULT2`	Provides a backup default action when the response for DEFAULT is set to EXCEPTION. Use DEFAULT2 when an exceptions MAP statement is not specified for a MAP statement for which errors are anticipated.
`<SQL error>`	A SQL error number. This can be a record-level error or a commit-level error if using TRANSDISCARD and TRANSEXCEPTION.
`<user-defined error>`	A user-defined error that is specified with the RAISEERROR option of a FILTER clause within a MAP statement.

Table 39 Error Response Options

`ABEND`	Rolls back the transaction and terminates processing abnormally. ABEND is the default.
`DISCARD`	Logs the offending operation to the discard file but continue processing the transaction and subsequent transactions. Specify a discard file with the DISCARDFILE parameter.

Table 39 Error Response Options

`EXCEPTION`	Handles an individual operation that causes an error as an exception, but processes the other operations in the transaction normally. Use an exceptions MAP statement that executes only after an exception and maps the failed operation to an exceptions table. For example, you can map columns from failed update statements into a “missing updates” table. In the parameter file, specify the exceptions MAP statement after the MAP statement for which the error is anticipated. EXCEPTION applies exception handling only to an individual SQL operation on an individual record. To apply exception handling to the entire transaction, use the TRANSEXCEPTION option. *Note*: When the Conflict Detection and Resolution (CDR) feature is active, CDR automatically treats all operations that cause errors as exceptions if an exceptions MAP statement exists for the affected table. In this case, REPERROR with EXCEPTION is not necessary, but you should use REPERROR with other options to handle conflicts that CDR cannot resolve, or for conflicts that you do not want CDR to handle. For more information about error handling, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.
`IGNORE`	Ignores the error.

`RETRYOP`
`[MAXRETRIES <n>]`
	Retries the offending operation. Use the MAXRETRIES option to control the number of retries. For example, if a table is out of extents, RETRYOP with MAXRETRIES gives you time to add extents so the transaction does not fail. Replicat abends after the specified number of MAXRETRIES. To set the amount of time between attempts, set RETRYDELAY as described on page 311.

`TRANSABORT`
`[, MAXRETRIES <n>]`
`[, DELAYSECS <n> |`
`DELAYCSECS <n>]`
	Aborts the transaction and repositions to the beginning of the transaction. This will continue either until the record(s) are processed successfully or MAXRETRIES expires. If MAXRETRIES is not set, the TRANSABORT action will loop continuously.
	Use a DELAY option to delay the retry. The default delay is 60 seconds.
	The TRANSABORT option is useful for handling timeouts and deadlocks on databases that support those conditions.

Table 39 Error Response Options

`TRANSDISCARD`	Discards the entire source transaction if any operation within that transaction, including the commit operation, causes a Replicat error that is listed in the REPERROR error specification. Replicat aborts the transaction and, if the error occured on a record, writes that record to the discard file that is specified with the DISCARDFILE parameter. Replicat then replays the transaction and writes all of the records to the discard file, including the commit record. Replicat abends on errors that are caused by the discard processing. If the discarded record has already been data-mapped to a target record, Replicat writes it to the discard file in the target format; otherwise, it will be written in source format. The replayed transaction itself is always written in source format. TRANSDISCARD supports record-level errors as well as commit errors. Additional information is at the beginning of this topic.
`TRANSEXCEPTION`	If an error specified with REPERROR occurs on any record in a transaction, performs exceptions mapping for every record in the transaction according to its corresponding exceptions-mapping specification, as defined by a MAPEXCEPTION or EXCEPTIONSONLY clause in an exceptions MAP statement. If any record does not have a corresponding exceptions mapping specification, or if there is an error writing to the exceptions table, Replicat abends with an error message. When an error is encountered and TRANSEXCEPTION is being used, Replicat aborts the transaction and, if the error occurred on a record, writes that record to the discard file that is specified with the DISCARDFILE parameter. Replicat replays the transaction and examines the source records to find the exceptions-mapping specifications, and then executes them. TRANSEXCEPTION supports record-level errors as well as commit errors. Additional information is at the beginning of this topic.

Example 1 The following example demonstrates how to stop processing for most errors, but ignore duplicate-record errors.

```
REPERROR (DEFAULT, ABEND)
REPERROR (-1, IGNORE)
```

Example 2 The following example invokes an exceptions MAP statement created to handle errors on the account table. Errors on the product table cause Replicat to end abnormally because an exceptions MAP statement was not defined.

```
REPERROR (DEFAULT, EXCEPTION)
REPERROR (DEFAULT2, ABEND)
MAP sales.product, TARGET sales.product;
MAP sales.account, TARGET sales.account;
INSERTALLRECORDS
MAP sales.account, TARGET sales.account_exception,
EXCEPTIONSONLY,
COLMAP (account_no = account_no,
optype = @GETENV ("lasterr", "optype"),
dberr = @GETENV ("lasterr", "dberrnum"),
dberrmsg = @GETENV ("lasterr", "dberrmsg"));
```

Example 3 The following applies error rules for the first MAP statement and then restores the default of ABEND to the second one.

```
REPERROR (-1, IGNORE)
MAP sales.product, TARGET sales.product;
REPERROR RESET
MAP sales.account, TARGET sales.account;
```

Example 4 The following discards the offending record and then replays the entire transaction if any operation on a record within it generates an error 1403. Other error types cause Replicat to abend.

```
REPERROR DEFAULT ABEND
REPERROR 1403 TRANSDISCARD
```

Example 5 The following discards the offending record and then replays the entire transaction to search for an exceptions-mapping specification that writes to the exceptions table that is named “tgtexception.” Other errors cause Replicat to discard the offending record (if applicable) and then abend.

```
REPERROR DEFAULT ABEND
REPERROR 1403 TRANSEXCEPTION
MAP src, TARGET tgt, &
MAPEXCEPTION (TARGET tgtexception, INSERTALLRECORDS, COLMAP (…) );
```

REPFETCHEDCOLOPTIONS

Valid for Replicat

Use the REPFETCHEDCOLOPTIONS parameter to determine how Replicat responds to operations for which a fetch from the source database was required. The Extract process fetches column data when the transaction record does not contain enough information to construct a SQL statement or when a FETCHCOLS clause is used (see page 354).

Default None

Syntax

```
REPFETCHEDCOLOPTIONS
[, INCONSISTENTROW]
[, LATESTROWVERSION {IGNORE | REPORT | DISCARD | ABEND}]
[, MISSINGROW {IGNORE | REPORT | DISCARD | ABEND}]
[, NOFETCH <action>]
[, REDUNDANTROW]
[, SETIFMISSING [<string>]]
[, SNAPSHOTROW]
```

Argument	Description
INCONSISTENTROW	Indicates that column data was successfully fetched by row ID, but the key did not match. Either the row ID was recycled or a primary key update occurred after this operation (and prior to the fetch). By default, Replicat logs the row to the discard file, and continues processing subsequent data.
LATESTROWVERSION <action>	Provides a response when column data was fetched from the current row in the table. Valid values are: IGNORE Ignore the condition and continue processing. REPORT Report the condition and contents of the row to the discard file, but continue processing the row. A discard file must be specified with the DISCARDFILE parameter. DISCARD Discard the data and do not process the row. A discard file must be specified with the DISCARDFILE parameter. ABEND Discard the data and quit processing. A discard file must be specified with the DISCARDFILE parameter.
NOFETCH <action>	Prevents fetching. One use for this option is when the database is a standby and Oracle GoldenGate does not have a database connection. In this case, an attempt to fetch from the database would result an error. Other scenarios may warrant the use of this parameter as well. When Oracle GoldenGate cannot fetch data it normally would fetch, it probably will cause data integrity issues on the target.

Argument	Description
	The following are valid actions that can be taken when a NOFETCH is encountered: ABEND Write the operation to the discard file and abend the Replicat process. This is the default. ALLOW Process the operation unless the record length is 0. IGNORE Ignore the operation. If fetch statistics are being reported in the process report (based on STATOPTIONS settings) they will be updated with this result. REPORT Write the record to the discard file and process the operation. DISCARD Write the record to the discard file, but do not process the operation. If fetch statistics are being reported in the process report (based on STATOPTIONS settings) they will be updated with this result.
`MISSINGROW <action>`	Provides a response when only part of a row (the changed values) is available to Replicat for processing. The column data that is missing from the trail typically could not be fetched because the row was deleted between the time the change record was created and when the fetch was triggered, or because the row image required was older than the undo retention specification. Valid values are: IGNORE Ignore the condition and continue processing. REPORT Report the condition and contents of the row to the discard file, but continue processing the partial row. A discard file must be specified with the DISCARDFILE parameter. DISCARD Discard the data and do not process the partial row. A discard file must be specified with the DISCARDFILE parameter. ABEND Discard the data and quit processing. A discard file must be specified with the DISCARDFILE parameter.
`REDUNDANTROW`	Indicates that column data was not fetched because column data was previously fetched for this record.

Argument	Description
`SETIFMISSING [<string>]`	Provides a value when a fetch was unsuccessful (and the value is missing from the trail record) but the target column has a not-null constraint. It takes an optional ASCII string as a value for CHAR and BINARY data types or defaults to the following. CHAR, VARCHAR: Single space BINARY, VARBINARY: A NULL byte TIMESTAMP: Current date/time FLOAT, INTEGER: Zero Besides SETIFMISSING, you can use the COLMAP clause of the MAP statement to map a value for the target column. See page 235.)
`SNAPSHOTROW`	Indicates that column data was fetched from a snapshot. Generally, this option would only be used for reporting or discarding operations.

REPLACEBADCHAR

Valid for Extract and Replicat

Use the REPLACEBADCHAR parameter to specify a substitution value for invalid character data that is encountered when mapping character columns. Unless a replacement value is specified, character columns with unprintable characters are output as hex strings. REPLACEBADCHAR applies globally.

Note that REPLACEBADCHAR replaces all unprintable non-ASCII single-byte characters to a single-byte value. It does not support multi-byte or 8-bit characters.

Default `UNPRINTABLE`

Syntax `REPLACEBADCHAR {<char> | SPACE | NULL | UNPRINTABLE | NONE}`

Argument	Description
`<char>`	Replace with the specified single-byte character.
`SPACE`	Replace with spaces.
`NULL`	Replace with NULL if the target column accepts NULL values; otherwise replace with spaces.
`UNPRINTABLE`	Reject any column that contains invalid data.
`NONE`	Suppress transformation of double-byte character set values to default characters.

Example 1 The following example replaces invalid characters with spaces.

```
REPLACEBADCHAR SPACE
```

Example 2 The following example replaces unprintable characters with carots.

```
REPLACEBADCHAR ^
```

REPLACEBADNUM

Valid for Replicat

Use the REPLACEBADNUM parameter to specify a substitution value for invalid numeric data encountered when mapping number columns. REPLACEBADNUM applies globally.

Default Replace invalid numbers with NULL.

Syntax

```
REPLACEBADNUM {<number> | NULL | UNPRINTABLE}
```

Argument	Description
`<number>`	Replace with the specified number.
`NULL`	Replace with NULL if the target column accepts NULL values; otherwise replace with zero.
`UNPRINTABLE`	Reject any column with unprintable data. The process stops and reports the bad value.

Example 1

```
REPLACEBADNUM 1
```

Example 2

```
REPLACEBADNUM NULL
```

REPLICAT

Valid for Replicat

Use the REPLICAT parameter to specify a Replicat group for online change synchronization. This parameter links the current run with previous runs, so that data changes are continually processed to maintain synchronization between source and target tables. Replicat will run continuously and maintain checkpoints in the data source and trail to ensure data integrity and fault tolerance throughout planned or unplanned process termination, system outages, or network failure.

Either REPLICAT or SPECIALRUN is required in the Replicat parameter file and must be the first entry. For more information about SPECIALRUN, see page 331.

Default None

Syntax

```
REPLICAT <group name>
```

Argument	Description
`<group name>`	The group name as defined with the ADD REPLICAT command.

Example

```
REPLICAT finance
```

REPORT

Valid for Extract and Replicat

Use the REPORT parameter to specify when Extract or Replicat generates interim runtime statistics in a process report. The statistics are added to the existing report. By default, runtime statistics are displayed at the end of a run unless the process is intentionally killed.

The statistics for REPORT are carried over from the previous report. For example, if the process performed 10 million inserts one day and 20 million the next, and a report is generated at 3:00 each day, then the first report would show the first 10 million inserts, and the second report would show those plus the current day's 20 million inserts, totalling 30 million. To reset the statistics when a new report is generated, use the STATOPTIONS parameter with the RESETREPORTSTATS option. See page 335.

For more information about using process reports, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Default Generate runtime statistics at the end of each run.

Syntax

```
REPORT
{AT <hh:mi> |
ON <day> |
AT <hh:mi> ON <day>}
```

Argument	Description
`AT <hh:mi>`	Generate the report at a specific time of the day. Using AT without ON generates a report at the specified time every day.

`ON <day>`	Generate the report on a specific day of the week. Valid values:
	`SUNDAY`
	`MONDAY`
	`TUESDAY`
	`WEDNESDAY`
	`THURSDAY`
	`FRIDAY`
	`SATURDAY`
	They are not case-sensitive.

Example 1

```
REPORT AT 17:00
```

Example 2

```
REPORT ON SUNDAY AT 1:00
```

REPORTCOUNT

Valid for Extract and Replicat

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.

NOTE This count might differ from the number of records that are contained in the Oracle GoldenGate trail. If an operation affects data that is larger than 4K, it must be stored in more than one trail record. Hence, a report count might show 1,000 records (the database operations) but a trail count might show many more records than that. To obtain a count of the records in a trail, use the Logdump utility.

You can schedule record counts at regular intervals or after a specific number of records. Record counts are carried over from one report to the other.

REPORTCOUNT can be used only once in a parameter file. If there are multiple instances of REPORTCOUNT, Oracle GoldenGate uses the last one.

Default None

Syntax

```
REPORTCOUNT [EVERY] <count>
{RECORDS | SECONDS | MINUTES | HOURS} [, RATE]
```

Argument	Description
`<count>`	The interval after which to output a count.
`RECORDS | SECONDS | MINUTES | HOURS`	The unit of measure for <count>, in terms of records, seconds, minutes, or hours.
`RATE`	Reports the number of operations per second and the change in rate, as a measurement of performance. See Example 2. The “rate” statistic is the total number of records divided by the total time elapsed since the process started. The “delta” statistic is the number of records since the last report divided by the time since the last report. Note: The calculations are done using microsecond time granularity. The time intervals are shown without fractional seconds, and the rate values are shown as whole numbers.

Example 1 This example generates a record count every 5,000 records.

```
REPORTCOUNT EVERY 5000 RECORDS
```

Example 2 This example generates a record count every ten minutes and also reports processing statistics.

```
REPORTCOUNT EVERY 10 MINUTES, RATE
```

The processing statistics are similar to this:

```
12000 records processed as of 2011-01-01 12:27:40 (rate 203,delta 308)
```

REPORTROLLOVER

Valid for Extract and Replicat

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.

NOTE Report statistics are carried over from one report to the other. To reset the statistics in the new report, use the STATOPTIONS parameter with the RESETREPORTSTATS option.

You can specify a time of day, a day of the week, or both. Specifying just a time of day (AT option) without a day of the week (ON option) generates a report at the specified time every day.

Rollovers caused by this parameter do not generate runtime statistics in the process report:

- To control when runtime statistics are generated to report files, use the REPORT parameter.
- To generate new runtime statistics on demand, use the SEND EXTRACT or SEND REPLICAT command with the REPORT option.

Default Roll reports at startup

Syntax

```
REPORTROLLOVER
{AT <hh:mi> |
ON <day> |
AT <hh:mi> ON <day>}
```

Argument	Description
`AT <hh:mi>`	The time of day to age the file. Valid values: ◆ hh is based on a 24-hour clock and accepts values of 1 through 23. ◆ mi accepts values from 00 through 59.
`ON <day>`	The day of the week to age the file. Valid values are: SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY They are not case-sensitive.

Example 1 `REPORTROLLOVER AT 05:30`

Example 2 `REPORTROLLOVER ON friday`

Example 3 `REPORTROLLOVER AT 05:30 ON friday`

RESTARTCOLLISIONS | NORESTARTCOLLISIONS

Valid for Replicat

Use the RESTARTCOLLISIONS and NORESTARTCOLLISIONS parameters to control whether or not Replicat applies HANDLECOLLISIONS logic after Oracle GoldenGate has stopped because of a conflict. By default, NORESTARTCOLLISIONS applies. However, there might be circumstances when you would want Oracle GoldenGate to apply HANDLECOLLISIONS logic for the first transaction after startup. For example, if the server is forcibly shut down, the database might have committed the last Replicat transaction, but Oracle GoldenGate might not have received the acknowledgement. Consequently, Replicat will retry the transaction upon startup. HANDLECOLLISIONS automatically handles the resultant errors that occur.

RESTARTCOLLISIONS enables HANDLECOLLISIONS functionality until the first Replicat checkpoint (transaction) is complete. You need not specify the HANDLECOLLISIONS parameter in the parameter file. After the first checkpoint, HANDLECOLLISIONS is automatically turned off.

For more information about HANDLECOLLISIONS, see page 217.

Default `NORESTARTCOLLISIONS`

Syntax `RESTARTCOLLISIONS | NORESTARTCOLLISIONS`

RETRYDELAY

Valid for Replicat

Use the RETRYDELAY parameter to specify the delay between attempts to retry a failed operation. Use this parameter when using the RETRYOP option of the REPERROR parameter (see page 298).

Default 60 seconds

Syntax `RETRYDELAY <seconds>`

Argument	Description
`<seconds>`	The number of seconds between retry attempts.

Example `REPERROR (100, RETRYOP MAXRETRIES 3) RETRYDELAY 30`

RMTFILE

Valid for Extract

Use the RMTFILE parameter to define the name of an extract file on a remote system to which extracted data will be written. Use this parameter for initial-load configurations. For online change synchronization, use the RMTTRAIL parameter.

The size of an extract file cannot exceed 2GB .

RMTFILE must be preceded by a RMTHOST statement, and it must precede any TABLE statements.

You can encrypt the data in this file by using the ENCRYPTTRAIL parameter (page 191).

Default None

Syntax

```
RMTFILE <file name>
[, APPEND]
[, PURGE]
[, MAXFILES <number>]
[, MEGABYTES <megabytes>]
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<file name>`	The relative or fully qualified name of the file.
`APPEND`	Adds the current data to existing data in the file. If you use APPEND, do not use PURGE.
`PURGE`	Deletes an existing file before creating a new one. If you use PURGE, do not use APPEND.
`MAXFILES <number>`	Forces a sequence of files to be created, rather than a single file. Use this option when you expect the size of a file to exceed the limit permitted by the operating system. MAXFILES permits as many files to be created as needed. Aged files are appended with a six-digit sequence number, for example datafile000002. When using MAXFILES, also use MEGABYTES to set the maximum size of each file in the sequence. Checkpoints are not maintained in these files.
`MEGABYTES <megabytes>`	Defines the maximum size of the file (or of each file created when MAXFILES is used). The size of a remote file cannot exceed 2GB .
`FORMAT RELEASE <major>.<minor>`	Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones. ◆ FORMAT is a required keyword. ◆ RELEASE specifies an Oracle GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current Oracle GoldenGate version number. (If you use an Oracle GoldenGate version that is earlier than 9.0, specify either 9.0 or 9.5.) The release version is programatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail.

Argument	Description
	There is a dependency between FORMAT and the RECOVERYOPTIONS parameter. When RECOVERYOPTIONS is set to APPENDMODE, FORMAT must be set to RELEASE 10.0 or greater. When RECOVERYOPTIONS is set to OVERWRITEMODE, FORMAT must be set to RELEASE 9.5 or less. See Appendix 2 on page 562 for additional information about Oracle GoldenGate trail file versioning and recovery modes.

Example 1 `RMTFILE /ggs/dirdat/salesny, MEGABYTES 2, PURGE`

Example 2 `RMTFILE /ggs/dirdat/salesny, MEGABYTES 2, FORMAT RELEASE 10.4`

RMTHOST

Valid for Extract

Use the RMTHOST parameter to:

- Identify a remote system to which the local Extract process connects
- Specify the TCP/IP port number on that system where the Manager process is running
- Control various attributes of the TCP/IP connections

This parameter controls compression, data encryption, buffer attributes, TCP/IP streaming, connection timeout threshold, and the wait period for a connection request. It also can be used to set Collector parameters.

To identify multiple remote systems in a parameter file, use one RMTHOST statement for each one, followed by the associated trails and table maps, for example:

```
EXTRACT sales
USERID ggs, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1
RMTHOST ny, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey
RMTTRAIL /ggs/dirdat/aa
TABLE ora.orders;
RMTHOST la, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey2
RMTTRAIL /ggs/dirdat/bb
TABLE ora.orders;
```

Do not use RMTHOST for an Extract created in PASSIVE mode. See page 17 for more information about a passive Extract.

Determining the optimum buffer size

The TCPBUFSIZE option controls the size of the TCP socket buffer that Extract will try to maintain, allowing larger packet sizes to be sent to the target system. You can use the following formula as a guideline for further experimentation to determine the optimum buffer size for your network.

1. Use the ping command from the command shell obtain the average round trip time (RTT), shown in the following example:

```
C:\home\ggs>ping ggsoftware.com
Pinging ggsoftware.com [192.168.116.171] with 32 bytes of data:
Reply from 192.168.116.171: bytes=32 time=31ms TTL=56
Reply from 192.168.116.171: bytes=32 time=61ms TTL=56
Reply from 192.168.116.171: bytes=32 time=32ms TTL=56
Reply from 192.168.116.171: bytes=32 time=34ms TTL=56
Ping statistics for 192.168.116.171:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 31ms, Maximum = 61ms, Average = 39ms
```

2. Multiply that value by the network bandwidth. For example, if average RTT is .08 seconds, and the bandwidth is 100 megabits per second, then the optimum buffer size is:

```
0.08 second * 100 megabits per second = 8 megabits
```

3. Divide the result by 8 to determine the number of bytes (8 bits to a byte). For example:

```
8 megabits / 8 = 1 megabyte per second
```

The required unit for TCPBUFSIZE is bytes, so you would set it to a value of 1000000.

The maximum socket buffer size for non-Windows systems is usually limited by default. Ask your system administrator to increase the default value on the source and target systems so that Oracle GoldenGate can increase the buffer size configured with TCPBUFSIZE.

NOTE Performance improvements are seen only when the target Oracle GoldenGate version is 8.0.4 or higher.

Supported internet protocols

Oracle GoldenGate supports IPv4 and IPv6 protocols. See the USEIPV6 parameter for more information about the selection of internet protocol.

Default None

Syntax

```
RMTHOST
{<host name> | <IP address>}
{, MGRPORT <port> | PORT <port>}
[, COMPRESS]
[, COMPRESSTHRESHOLD]
[, ENCRYPT <algorithm> KEYNAME <keyname>
[, KEYNAME <keyname>]
[, PARAMS <collector parameters>]
[, STREAMING | NOSTREAMING]
[, TCPBUFSIZE <bytes>]
[, TCPFLUSHBYTES <bytes>]
[, TIMEOUT <seconds>]
```

Argument	Description
`{<host name> | <IP address>}`	The DNS host name or IP address of the target system. You can use either one to define the host. If using an IP address, use either an IPv6 or IPv4-mapped address, depending on the stack of the destination system.
`COMPRESS`	Compresses outgoing blocks of records to reduce bandwidth requirements. Oracle GoldenGate decompresses the data before writing it to the trail. COMPRESS typically results in compression ratios of at least 4:1 and sometimes better. However, compressing data can consume CPU resources.
`COMPRESSTHRESHOLD`	Sets the minimum block size for which compression is to occur. Valid values are from 0 and through 28000. The default is 1,000 bytes.

`ENCRYPT <algorithm>`
`KEYNAME <keyname>`
	Encrypts the data stream sent over TCP/IP to the target system.
	◆ <algorithm> specifies the encryption algorithm to use:
	◆ AES128 uses the AES-128 cipher, which has a key size of 128 bits.
	◆ AES192 uses the AES-192 cipher, which has a key size of 192 bits.
	◆ AES256 uses the AES-256 cipher, which has a key size of 256 bits.
	◆ BLOWFISH uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32 bits to 128 bits. Use BLOWFISH only for backward compatibility with earlier Oracle GoldenGate versions. It is less secure than AES.
	AES128 is the default if no algorithm is specified.
	◆ KEYNAME <keyname> specifies the logical name of an encryption key in the ENCKEYS lookup file. The key name is used to look up the actual key in the ENCKEYS file.
	
	To use AES encryption for any database other than Oracle, the path of the lib sub-directory of the Oracle GoldenGate installation directory must be specified as an environment variable before starting any processes:
	◆ UNIX: Specify the path as an entry to the LD_LIBRARY_PATH or SHLIB_PATH variable. For example:
	`setenv LD_LIBRARY_PATH ./lib:$LD_LIBRARY_PATH`
	◆ Windows: Add the path to the PATH variable.
	You can use the SETENV parameter to set it as a session variable for the process.
	For more information about using data encryption, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

`KEYNAME <keyname>`	A key name in the ENCKEYS file. Oracle GoldenGate uses the key to decrypt the data. Unless a matching key name exists in the ENCKEYS file on the target system, Oracle GoldenGate abends. For more information about creating an ENCKEYS file, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Argument	Description
`MGRPORT <port>`	The port on the remote system where Manager runs. Either MGRPORT or PORT is required.
`PORT <port>`	The port number of a static Collector process. Either a Manager port (if using a dynamic Collector) or a static Collector port must be specified. For information about a static collector, see Chapter 3.
`PARAMS <collector parameters>`	Specifies Collector parameters on a NonStop target system. Note: Do not specify a Collector port (-p argument) if Manager will be starting Collector dynamically. For more information about Collector parameters on the NonStop platform, see the Oracle GoldenGate *HP NonStop Reference Guide*.
`STREAMING | NOSTREAMING`	Controls TCP/IP streaming. ◆ STREAMING enables the asynchronous internet streaming protocol and is the default. In STREAMING mode, the receiver (Collector) does not send an acknowledgement to the sender (primary Extract or data pump) for any data packet unless the packet contains a flag requesting a response, typically when the sender must checkpoint or determine a write position. Because this method omits acknlowledgements, the sender or receiver process terminates if there is a network disruption; therefore, when using STREAMING, use the AUTORESTART parameter in the Manager parameter file to restart Extract and Collector if they terminate. ◆ NOSTREAMING enables the synchronous internet protocol. In NOSTREAMING mode, the sender sends a packet and then waits for the receiver to acknowledge it, before sending the next packet. This method is more reliable, because it enables the sender or receiver process to recover if there is a network disruption. Extract falls back to the synchronous protocol automatically if the host system of the receiver process is not configured to use streaming. Keep the STREAMING default unless you are requested to disable it, because streaming reduces transmission latency, especially in networks where latency is a problem already. Streaming is not supported for initial-load tasks where Extract communicates directly with Replicat.
`TCPBUFSIZE <bytes>`	Controls the size of the TCP socket buffer, in bytes, that Extract will try to maintain. By increasing the size of the buffer, you can send larger packets to the target system. The actual size of the buffer depends on the TCP stack implementation and the network. The default is 30,000 bytes, but modern network configurations usually support higher values. Valid values are from 1000 to 200000000 (two hundred million) bytes. Work with your network administrator to determine an optimal value. See also “Determining the optimum buffer size” on page 313.

Argument	Description
	If the Oracle GoldenGate installation on the target system is a version earlier than 8.0.4, only a buffer of 30,000 bytes will be used regardless of what is specified with TCPBUFSIZE. Earlier versions of the Collector process do not support packets larger than that. Do not use this parameter for an initial load Extract. It is valid only for an online Extract group. Do not use this parameter if the target system is NonStop.
`TCPFLUSHBYTES <bytes>`	Controls the size of the buffer, in bytes, that collects data that is ready to be sent across the network. When either this value or the value of the FLUSHSECS parameter is reached, the data is flushed to the target. The default is 30,000 bytes. Valid values are from 1000 to 200000000 (two hundred million) bytes, but should be at least the value of TCPBUFSIZE. Do not use this parameter for an initial load Extract. It is valid only for an online Extract group. Do not use this parameter if the target system is NonStop.
`TIMEOUT <seconds>`	Specifies how long Collector waits to get a connection from Extract, and how long Collector waits for a heartbeat signal from Extract before terminating a connection. Valid values are 1 second to 1800 seconds (30 minutes). The default value is 300 seconds (5 minutes). Setting the timeout to a very low value is not recommended in a production setting. You might need to increase the TIMEOUT value if you see a warning in the error log that there was a TCP/IP error 10054 (existing connection forcibly closed by remote host). This error typically occurs when the Collector terminates itself after the TIMEOUT value is exceeded. This parameter does not affect a static Collector.

Example 1
```
RMTHOST 20.20.20.17, MGRPORT 7809, ENCRYPT AES192, KEYNAME newyork
```

Example 2
```
RMTHOST newyork, MGRPORT 7809, COMPRESS, COMPRESSTHRESHOLD 750, NOSTREAMING
```

Example 3
```
RMTHOST newyork, MGRPORT 7809, TCPBUFSIZE 100000, TCPFLUSHBYTES 300000
```

RMTHOSTOPTIONS

Valid for Passive Extract

Use the RMTHOSTOPTIONS parameter to control attributes of a TCP/IP connection made between an Extract group running in PASSIVE mode on a less trusted source to a target system in a more secure network zone. This parameter controls compression, data encryption, buffer attributes, streaming, and the wait period for a connection request. It also can be used to set Collector parameters.

This parameter differs from the RMTHOST parameter because it does not provide the host information needed to establish a remote connection. When Extract is running in PASSIVE mode, all connections between source and target are established by an alias Extract group on the target. For more information about using Oracle GoldenGate in a zoned network, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

All parameter options must be specified in one RMTHOSTOPTIONS statement. If multiple RMTHOSTOPTIONS statements are used, the last one in the parameter file is used, and the others are ignored. RMTHOSTOPTIONS overrides any RMTHOST statements in the file.

See RMTHOST for additional information about supported IP protocols.

Default None

Syntax

```
RMTHOSTOPTIONS
[, COMPRESS]
[, COMPRESSTHRESHOLD]
[, ENCRYPT <algorithm> KEYNAME <keyname>]
[, PARAMS <collector parameters>]
[, STREAMING | NOSTREAMING]
[, TCPBUFSIZE <bytes>]
[, TCPFLUSHBYTES <bytes>]
[, TIMEOUT <seconds>]
```

Argument	Description
`COMPRESS`	Compresses outgoing blocks of records to reduce bandwidth requirements. Oracle GoldenGate decompresses the data before writing it to the trail. COMPRESS typically results in compression ratios of at least 4:1 and sometimes better. However, compressing data can consume CPU resources.
`COMPRESSTHRESHOLD`	Sets the minimum block size for which compression is to occur. Valid values are from 0 and through 28000. The default is 1,000 bytes.

`ENCRYPT <algorithm>`
`KEYNAME <keyname>`
	Encrypts the data stream sent over TCP/IP to the target system.
	◆ <algorithm> specifies the encryption algorithm to use:
	◆ AES128 uses the AES-128 cipher, which has a key size of 128 bits.
	◆ AES192 uses the AES-192 cipher, which has a key size of 192 bits.
	◆ AES256 uses the AES-256 cipher, which has a key size of 256 bits.
	◆ BLOWFISH uses Blowfish encryption with a 64-bit block size and a variable-length key size from 32 bits to 128 bits. Use BLOWFISH only for backward compatibility with earlier Oracle GoldenGate versions. It is less secure than AES.
	AES128 is the default if no algorithm is specified.
	◆ KEYNAME <keyname> specifies the logical name of an encryption key in the ENCKEYS lookup file. The key name is used to look up the actual key in the ENCKEYS file.For more information about using data encryption, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Argument	Description
	To use AES encryption for any database other than Oracle, the path of the lib sub-directory of the Oracle GoldenGate installation directory must be specified as an environment variable before starting any processes: ◆ UNIX: Specify the path as an entry to the LD_LIBRARY_PATH or SHLIB_PATH variable. For example: `setenv LD_LIBRARY_PATH ./lib:$LD_LIBRARY_PATH` ◆ Windows: Add the path to the PATH variable. You can use the SETENV parameter to set it as a session variable for the process.
`PARAMS <collector parameters>`	Specifies Collector parameters on a NonStop target system. Note: Do not specify a Collector port (-p argument) if Manager will be starting Collector dynamically. For more information about Collector parameters on the NonStop platform, see the Oracle GoldenGate *HP NonStop Reference Guide*.
`STREAMING | NOSTREAMING`	Controls TCP/IP streaming. ◆ STREAMING enables the asynchronous internet streaming protocol and is the default. In STREAMING mode, the receiver (Collector) does not send an acknowledgement to the sender (primary Extract or data pump) for any data packet unless the packet contains a flag requesting a response, typically when the sender must checkpoint or determine a write position. Because this method omits acknlowledgements, the sender or receiver process terminates if there is a network disruption; therefore, when using STREAMING, use the AUTORESTART parameter in the Manager parameter file to restart Extract and Collector if they terminate. ◆ NOSTREAMING enables the synchronous internet protocol. In NOSTREAMING mode, the sender sends a packet and then waits for the receiver to acknowledge it, before sending the next packet. This method is more reliable, because it enables the sender or receiver process to recover if there is a network disruption. Extract falls back to the synchronous protocol automatically if the host system of the receiver process is not configured to use streaming. Keep the STREAMING default unless you are requested to disable it, because streaming reduces transmission latency, especially in networks where latency is a problem already. Streaming is not supported for initial-load tasks where Extract communicates directly with Replicat.

Argument	Description
`TCPFLUSHBYTES <bytes>`	Controls the size of the buffer, in bytes, that collects data that is ready to be sent across the network. When either this value or the value of the FLUSHSECS parameter is reached, the data is flushed to the target. The default is 30,000 bytes. Valid values are from 1000 to 200000000 (two hundred million) bytes, but should be at least the value of TCPBUFSIZE. Do not use this parameter for an initial load Extract. It is valid only for an online Extract group. Do not use this parameter if the target system is NonStop.
`TIMEOUT <seconds>`	Specifies how long an Extract running in PASSIVE mode waits to get a connection from Collector, and how long Extract waits for a heartbeat signal from Collector before terminating a connection. Valid values are 1 second to 1800 seconds (30 minutes). The default value is 300 seconds (5 minutes). Setting the timeout to a very low value is not recommended in a production setting. You might need to increase the TIMEOUT value if you see a warning in the error log that there was a TCP/IP error 10054 (existing connection forcibly closed by remote host). This error typically occurs when the Extract terminates itself after the TIMEOUT value is exceeded.

Example

```
RMTHOSTOPTIONS ENCRYPT AES192, KEYNAME newyork, COMPRESS, COMPRESSTHRESHOLD
750, TCPBUFSIZE 100000, TCPFLUSHBYTES 300000, NOSTREAMING
```

RMTTASK

Valid for Extract

Use the RMTTASK parameter for an initial-load Extract to initiate a Replicat processing task during an Oracle GoldenGate direct load or a direct bulk load to SQL*Loader. RMTTASK directs Extract to communicate directly with Replicat over TCP/IP and bypasses the use of a Collector process or disk storage. RMTTASK also directs Extract to request that Manager start Replicat automatically, and then stop Replicat when the run is finished. Tasks do not use checkpoints.

Dependent parameters are as follows:

- A RMTHOST statement must follow each RMTTASK statement in the initial-load Extract parameter file.
- EXTRACT must be used in the initial-load Extract parameter file.
- REPLICAT must be used in the initial-load Replicat parameter file.
- SOURCEISTABLE must be used in the ADD EXTRACT command.
- SPECIALRUN must be used in the ADD REPLICAT command.

RMTTASK does not support encryption of any kind. To use encryption, you can use the initial load method that writes data to a file, which is read by Replicat to load the data.

RMTTASK 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 4k in size.

When using RMTTASK, do not start Replicat with the START REPLICAT command. Replicat is started automatically during the task.

For more information about performing initial data loads, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Default None

Syntax

```
RMTTASK REPLICAT, GROUP <group name>
[FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`GROUP <group name>`	The group name of the Initial Load Replicat on the target system.
`FORMAT RELEASE <major>.<minor>`	Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones. ◆ FORMAT is a required keyword. ◆ RELEASE specifies an Oracle GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current Oracle GoldenGate version number. (If you use an Oracle GoldenGate version that is earlier than 9.0, specify either 9.0 or 9.5.) The release version is programatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail. There is a dependency between FORMAT and the RECOVERYOPTIONS parameter. When RECOVERYOPTIONS is set to APPENDMODE, FORMAT must be set to RELEASE 10.0 or greater. When RECOVERYOPTIONS is set to OVERWRITEMODE, FORMAT must be set to RELEASE 9.5 or less. See Appendix 2 on page 562 for additional information about Oracle GoldenGate trail file versioning and recovery options.

Example

```
RMTTASK REPLICAT, GROUP initrep, FORMAT RELEASE 10.0
```

RMTTRAIL

Valid for Extract

Use the RMTTRAIL parameter to specify a remote trail that was created with the ADD RMTTRAIL command in GGSCI. A trail specified with RMTTRAIL must precede its associated TABLE statements. Multiple RMTTRAIL statements can be used to specify different remote trails. RMTTRAIL must be preceded by a RMTHOST parameter.

You can encrypt the data in this trail by using the ENCRYPTTRAIL parameter (page 191).

Default None

Syntax

```
RMTTRAIL <trail name>
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<name>`	The relative or fully qualified path name of the trail. Use two characters for the name. As trail files are aged, a six-character sequence number will be added to this name, for example /ggs/dirdat/rt000001.
`FORMAT RELEASE <major>.<minor>`	Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones. ◆ FORMAT is a required keyword. ◆ RELEASE specifies an Oracle GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current Oracle GoldenGate version number. (If you use an Oracle GoldenGate version that is earlier than 9.0, specify either 9.0 or 9.5.) The release version is programatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail. There is a dependency between FORMAT and the RECOVERYOPTIONS parameter. When RECOVERYOPTIONS is set to APPENDMODE, FORMAT must be set to RELEASE 10.0 or greater. When RECOVERYOPTIONS is set to OVERWRITEMODE, FORMAT must be set to RELEASE 9.5 or less. See Appendix 2 on page 562 for additional information about Oracle GoldenGate trail file versioning and recovery modes.

Example 1 `RMTTRAIL dirdat/ny`

Example 2 `RMTTRAIL /ggs/dirdat/ny, FORMAT RELEASE 10.4`

ROLLOVER

Valid for Extract

Use the ROLLOVER parameter to specify when trail files are aged and new ones are created. ROLLOVER is global and applies to all trails defined with RMTTRAIL or RMTFILE statements in a parameter file.

Use ROLLOVER to create trail files representing distinct periods of time (for example, each day). It facilitates continuous processing while providing a means for organizing the output. It also provides a means for organizing batch runs by deactivating one file and starting another for the next run.

Files roll over between transactions, not in the middle of one, ensuring data integrity. Checkpoints are recorded when files roll over to ensure that previous files are no longer required for processing.

Rollover occurs only if the rollover conditions are satisfied during the run. For example, if ROLLOVER ON TUESDAY is specified, and data extraction starts on Tuesday, the rollover does not occur until the next Tuesday (unless more precise ROLLOVER rules are specified). You can specify up to 30 rollover rules.

Either the AT or ON option is required. Both options can be used together, and in any order. Using AT without ON creates a new trail file at the specified time every day.

A trail sequence number can be incremented from 000001 through 999999, and then the sequence numbering starts over at 000000.

Default Roll over when the default file size is reached or the size specified with the MEGABYTES option of the ADD RMTTRAIL or ADD EXTTRAIL command is reached.

Syntax

```
ROLLOVER {AT <hh:mi> | ON <day> | AT <hh:mi> ON <day>} [REPORT]
```

Argument	Description
`AT <hh:mi>`	The time of day to age the file. Valid values: ◆ hh is based on a 24-hour clock, with valid values of 1 through 23. ◆ mi accepts values from 00 through 59.
`ON <day>`	The day of the week to age the file. Valid values: SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY They are not case-sensitive.
`REPORT`	Generates a report for the number of records extracted from each table since the last report was generated. The report represents the number of records output to the corresponding trail unless other reports are generated by means of the REPORT parameter.

Example 1 The following ages trails every day at 3:00 p.m.

```
ROLLOVER AT 15:00
```

Example 2 The following ages trails every Sunday at 8:00 a.m.

```
ROLLOVER AT 08:00 ON SUNDAY
```

SEQUENCE

Valid for Extract

Use the SEQUENCE parameter to extract sequence values from the transaction log for propagation to an Oracle GoldenGate trail and delivery to another database. Currently, Oracle GoldenGate supports sequences for the Oracle database.

NOTE DDL support for sequences (CREATE, ALTER, DROP, RENAME) is compatible with, but not required for, replicating sequence values. To replicate just sequence values, you *do not* need to install the Oracle GoldenGate DDL support environment. You can just use the SEQUENCE parameter.

Oracle GoldenGate ensures that the values of a target sequence are:

- higher than the source values if the increment interval is positive
- lower than the source values if the increment interval is negative

Depending on the increment direction, Replicat applies one of the following formulas as a test when it performs an insert:

source_highwater_value + (source_cache_size * source_increment_size) <= target_highwater_value

Or...

source_highwater_value + (source_cache_size * source_increment_size) >= target_highwater_value

If the formula evaluates to FALSE, the target sequence is updated to be higher than the source value (if sequences are incremented) or lower than the source value (if sequences are decremented). The target must always be ahead of, or equal to, the expression in the parentheses in the formula. For example, if the source highwater value is 40, and CACHE is 20, the target highwater value should be at least 60:

40 + (20*1) <60

If the target highwater value is less than 80, Oracle GoldenGate updates the sequence to increase the highwater value, so that the target remains ahead of the source. To get the current highwater value, perform this query:

```
SELECT last_number FROM all_sequences WHERE
sequence_owner=upper('SEQUENCEOWNER') AND
sequence_name=upper('SEQUENCENAME');
```

Supported processing modes

- Oracle GoldenGate initial load methods that contain the SOURCEISTABLE parameter, either as an Extract parameter or within ADD EXTRACT, do not support the replication of sequence values.
- Oracle GoldenGate does not support the replication of sequence values in an active-active bi-directional configuration.
- Oracle GoldenGate supports sequences in a high-availability configuration. This configuration includes a primary Extract, a data pump, and a Replicat on both servers, one as primary, the other as the target failover server. In this configuration, the Extract process on the failover server must be inactive, which includes not capturing

sequences. For more information about how to configure and operate Oracle GoldenGate in a high-availability configuration, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

- If using SEQUENCE for a primary Extract that writes to a data pump, you must also use an identical SEQUENCE parameter in the data pump, whether the data pump is in PASSTHRU or NOPASSTHRU mode. However, if the DDL parameter is being used to propagate DDL operations (for sequences or any other objects) through the same data pump, the data pump *must* operate in PASSTHRU mode.

Guidelines for using SEQUENCE

- The cache size and the increment interval of the source and target sequences must be identical.
- The cache can be any size, including 0 (NOCACHE).
- The sequence can be set to cycle or not cycle, but the source and target databases must be set the same way.
- To add SEQUENCE to a configuration in which DDL support is enabled, you must re-install the Oracle GoldenGate DDL objects in INITIALSETUP mode.

Error handling

- If Extract cannot resolve a sequence name, it ignores the operation.
- To enable Replicat error handling for sequences, use the REPERROR parameter. This parameter is available as an option in the MAP parameter and also as a standalone parameter. REPERROR can detect if a sequence has been dropped on the target and can be used to retry a sequence operation until the sequence is recreated.
- REPERROR does not handle missing objects on startup. Use DDLERROR with IGNOREMISSINGTABLES.

Other important information

- Gaps are possible in the values of the sequences that Oracle GoldenGate replicates because gaps are inherent, and expected, in the way that sequences are maintained by the database. However, the target values will always be greater than those of the source.
- If Extract is running in single-threaded mode on a RAC system, and if sequences are updated on a node that has lag, it might take more time to capture a sequence. This is normal behavior.
- In a failover, any problem that causes the loss or corruption of data in a transaction log or Oracle GoldenGate trail file will cause the loss of the replicated sequence updates.
- The statistics shown by SEND EXTRACT and SEND REPLICAT when used with the REPORT option will show the sequence operation as an UPDATE.

Default None

Syntax

```
SEQUENCE <owner>.<sequence>;
```

Argument	Description
`SEQUENCE <owner>.<sequence>`	Specifies the owner (schema) and name of the source sequence. For supported characters and globalization support, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.
`;`	Terminates the SEQUENCE parameter statement. The semi-colon is optional.

Example `SEQUENCE hr.employees_seq;`

SESSIONCHARSET

Valid for GLOBALS

Use the SESSIONCHARSET parameter to set the database session character set for all database connections that are initiated by Oracle GoldenGate processes in the local Oracle GoldenGate instance. Processes that log into the database include GGSCI, DEFGEN, Extract, and Replicat.

This parameter supports Sybase, Teradata and MySQL. The database character set for other databases is obtained programatically.

The SESSIONCHARSET option of the DBLOGIN command can be used to override this setting for any commands issued in the same GGSCI session. The SESSIONCHARSET option of the SOURCEDB and TARGETDB parameters can be used to override this setting for individual process logins.

Default Character set of the operating system

Syntax `SESSIONCHARSET <character set>`

Argument	Description
`<character set>`	The database session character set.

Example `SESSIONCHARSET ISO-8859-11`

SETENV

Valid for Extract and Replicat

Use the SETENV parameter to set a value for any environment variable. When Extract or Replicat starts, it uses the specified value instead of the one set in the environment.

Use one SETENV statement per variable to be set. Any variables set in the SETENV statement override any existing variables set at the operating system level.

Default None

Syntax

```
SETENV (
{<environment_variable> |
   GGS_CacheRetryCount |
   GGS_CacheRetryDelay}
= "<value>"
)
```

Options	Description
`<environment_variable>`	The name of the environment variable to be set.
`"<value>"`	A value for the specified variable. The value must be within quotes.
`GGS_CacheRetryCount`	(SQL Server) Oracle GoldenGate environment parameter that controls the number of times that Extract tries to read the source transaction log files when they are blocked because of excessive system activity. The default is 10 retries. After trying the specified number of times, Extract abends with an error that begins as follows: `GGS ERROR 600 [CFileInfo::Read] Timeout expired after 10 retries with 1000 ms delay waiting to read transaction log or backup files.` If you continue to see timeout messages in the report file or error log, increase this parameter to allow more retries.
`GGS_CacheRetryDelay`	(SQL Server) Oracle GoldenGate environment parameter that controls the number of milliseconds that Extract waits before trying again to read the transaction logs when the previous attempt has failed. The default is 1000 milliseconds delay.

Example 1 Using separate SETENV statements allows a single instance of Oracle GoldenGate to connect to multiple Oracle database instances without having to change environment settings. The following parameter statements set a value for ORACLE_HOME and ORACLE_SID.

```
SETENV (ORACLE_HOME = "/home/oracle/ora9/product")
SETENV (ORACLE_SID = "ora9")
```

Example 2 The following parameter statements set values for Oracle GoldenGate in a SQL Server environment where Extract tries to read the transaction log for a maximum of 20 times before abending, with a delay of 3000 milliseconds between tries.

```
SETENV (GGS_CacheRetryCount = 20)
SETENV (GGS_CacheRetryDelay = 3000)
```

SHOWSYNTAX

Valid for Replicat

Use the SHOWSYNTAX parameter to start an interactive session where you can view each Replicat SQL statement before it is applied. By viewing the syntax of SQL statements that failed, you might be able to diagnose the cause of the problem. For example, you could find out that the WHERE clause is using a non-indexed column.

Requirements for using SHOWSYNTAX

- The first time that you use SHOWSYNTAX, request guidance from an Oracle Support analyst. It is a debugging parameter and can cause unwanted results if used improperly. It requires manual intervention, so automated processing is suspended, and it slows down processing, which can cause backlogs and latency.
- To use SHOWSYNTAX, Replicat must be started from the command shell of the operating system. Do not use SHOWSYNTAX if Replicat is started through GGSCI.
- Use SHOWSYNTAX in a test environment. Create duplicates of your Replicat groups and target tables so that the production environment is not affected.

Using SHOWSYNTAX

1. In the Replicat parameter file, include the following parameters in the order shown here, each on its own line:
   - NOBINARYCHARS
   - NODYNSQL
   - SHOWSYNTAX

   NOTE NOBINARYCHARS is an undocumented parameter that causes Oracle GoldenGate to treat binary data as a null-terminated string. Contact Oracle Support before using it. NODYNSQL causes Replicat to use literal SQL statements instead of using dynamic SQL with bind variables. For support, go to http://support.oracle.com.

2. From the Oracle GoldenGate home directory, start Replicat from the command shell of the operating system using the syntax shown here. Do not specify a reportfile option. Output must go to screen.

   ```
   replicat paramfile dirprm/<Replicat_name>.prm
   ```

3. The first SQL statement is displayed with some prompts.
   - Choose Keep Displaying (the default) to execute the current statement and display the next one.
   - Choose Stop Display to resume normal processing and stop printing SQL statements to screen.

4. When finished viewing syntax, remove SHOWSYNTAX, NOBINARYCHARS, and NODYNSQL from the parameter file.

Default None

Syntax `SHOWSYNTAX`

SOURCEDB

Valid for Manager, Extract, DEFGEN

Use the SOURCEDB parameter for databases or data sets that require a data source name or identifier as part of the connection information. Tables specified in TABLE statements that follow SOURCEDB are assumed to be from the specified data source.

You might need to use the USERID parameter (see page 416) with SOURCEDB, depending on the authentication that is required for the data source,

- If a database login is required, SOURCEDB (if required) must be used with the USERID parameter within the same parameter statement.
- For SQL/MX databases, SOURCEDB specifies the catalog, and USERID specifies the schema. If the schema is omitted, SOURCEDB defaults to the schema that is associated with the group.
- For databases that allow authentication at the operating-system level, you can specify SOURCEDB without USERID.

For Manager, use SOURCEDB only when using Oracle GoldenGate parameters that cause Manager to interact with a source database, such as PURGEOLDEXTRACTS.

For DB2 LUW, the SOURCEDB statement must refer to the database by its real name, rather than by any alias.

Default None

Syntax

```
SOURCEDB <data source>[, SESSIONCHARSET <character set>]
```

Argument	Description
`<data source>`	The name of the data source. For MySQL databases, you can use the format of SOURCEDB <database_name>@<host_name> to avoid connection issues caused by the incorrect configuration of localhost in the local hosts file.
`SESSIONCHARSET <character set>`	Supports Sybase, Teradata and MySQL. Sets the database session character set for the process login session. This parameter overrides any SESSIONCHARSET that is specified in the GLOBALS file.

Example 1 This example shows SOURCEDB with and without the USERID parameter.

```
SOURCEDB mydb

SOURCEDB mydb, USERID ggs, &
   PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1
```

Example 2 This example sets a character set for the user session.

```
SOURCEDB mydb, USERID ggs, &
   PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1, SESSIONCHARSET ISO-8859-11
```

SOURCEDEFS

Valid for Extract data pump and Replicat.

Use the SOURCEDEFS parameter to specify the name of a file that contains definitions of source tables or files. Source definitions are required when using Oracle GoldenGate to replicate data between heterogeneous source and targets. Use SOURCEDEFS for one or more of the following processes, depending on your Oracle GoldenGate configuration:

- A Replicat process on the target system
- A data pump on a source or intermediary system.

To generate the source-definitions file, use the DEFGEN utility. Transfer the file to the intermediary or target system before starting a data pump or Replicat.

You can have multiple SOURCEDEFS statements in the parameter file if more than one source-definitions file will be used, for example if each SOURCEDEFS file holds the definitions for a distinct application.

See also ASSUMETARGETDEFS on page 127.

For more information about how Oracle GoldenGate make use of metadata, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Default None

Syntax `SOURCEDEFS <file name>`

Argument	Description
`<file name>`	The relative or fully qualified name of the file containing the source data definitions.

Example 1 `SOURCEDEFS dirdef\tcust.def`

Example 2 `SOURCEDEFS /ggs/dirdef/source_defs`

SOURCEISTABLE

Valid for Extract

Use the SOURCEISTABLE parameter to extract complete records directly from source tables in preparation for loading them into another table or file. SOURCEISTABLE extracts all column data specified within a TABLE statement.

This parameter applies to the following initial load methods:

- Loading data from file to Replicat.
- Loading data from file to database utility.

Do not use this parameter for the following initial load methods:

- An Oracle GoldenGate direct load, where Extract sends load data directly to the Replicat process without use of a file.
- An Oracle GoldenGate direct bulk load to SQL*Loader.

For those processes, SOURCEISTABLE is specified as an ADD EXTRACT argument instead of being used in the parameter file. For more information about initial data loads, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

When used, SOURCEISTABLE must be the first parameter statement in the Extract parameter file.

To use SOURCEISTABLE, disable DDL extraction and replication by omitting the DDL parameter from the Extract and Replicat parameter files. For more information, see page 164.

Default None

Syntax `SOURCEISTABLE`

SPACESTONULL | NOSPACESTONULL

Valid for Replicat

Use the SPACESTONULL and NOSPACESTONULL parameters to control whether or not a source column containing only spaces is converted to NULL in the target table. SPACESTONULL converts spaces to NULL if the target column accepts NULL values. NOSPACESTONULL converts spaces to a single space character in the target column.

The parameters are table-specific. One parameter applies to all subsequent MAP statements, until the other parameter is encountered. This parameter supports Oracle only.

Default `NOSPACESTONULL`

Syntax `SPACESTONULL | NOSPACESTONULL`

SPECIALRUN

Valid for Replicat

Use the SPECIALRUN parameter in a Replicat parameter file for a one-time processing run to direct Replicat not to create checkpoints. A one-time run has a beginning and an end, so checkpoints are not needed. Use SPECIALRUN for certain initial data load methods. For more information, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

When Replicat is in SPECIALRUN mode, do not start it with the START REPLICAT command in GGSCI. It is started automatically during the initial load.

SPECIALRUN requires the use of the END parameter. Either REPLICAT (see page 307) or SPECIALRUN is required in the Replicat parameter file. REPLICAT specifies online processing.

Default None

Syntax `SPECIALRUN`

SQLDUPERR

Valid for Replicat

Use the SQLDUPERR parameter to specify the numeric error code returned by the database when a duplicate row is encountered. A duplicate-record error indicates that an insert operation was attempted with a primary key that matches an existing record in the database.

You must use SQLDUPERR when you specify special handling of duplicate records with the OVERRIDEDUPS parameter. Use multiple instances of SQLDUPERR when replicating to multiple database types.

Default None

Syntax `SQLDUPERR <error number>`

Argument	Description
`<error number>`	The numeric error code to return for duplicate records.

Example The following statements indicate the duplicate-record error codes for Microsoft Access and SQL Server.

```
SQLDUPERR -1605
SQLDUPERR -2601
```

SQLEXEC

Valid for Extract and Replicat

Use the SQLEXEC parameter as follows:

- as a standalone statement at the root level of a parameter file to execute a SQL stored procedure or query. As a standalone statement, SQLEXEC executes independently of a TABLE or MAP statement during Oracle GoldenGate processing.
- as a standalone statement to execute a database command.

NOTE You also can use SQLEXEC as part of a TABLE (Extract) or MAP (Replicat) statement to execute a SQL stored procedure or query. For this usage, see the alphabetical listings for TABLE and MAP in this chapter.

SQLEXEC enables Oracle GoldenGate to communicate with the database to perform any function supported by the database. The database function can be integrated with the data extraction and replication processes, or independent of them.

Data types supported by SQLEXEC

The following are the databases that are supported by SQLEXEC and the data types that are supported for input and output parameters.

- Numeric data types
- Date data types
- Character data types

Guidelines for using a standalone SQLEXEC parameter

- A standalone SQLEXEC statement executes in the order in which it appears in the parameter file relative to other parameters.
- A SQLEXEC procedure or query must contain all exception handling.
- A query or procedure must be structured correctly when executing a SQLEXEC statement, with legal SQL syntax for the database; otherwise Replicat will abend, regardless of any error-handling rules that are in place. Refer to the SQL reference guide provided by the database vendor for permissible SQL syntax.
- A database login by the Oracle GoldenGate user must precede the SQLEXEC clause. For Extract, use the SOURCEDB and USERID parameters as appropriate for the database. For Replicat, use the TARGETDB and USERID parameters, as appropriate.
- The user under which the Oracle GoldenGate process is running is the user that executes the SQL. This user must have the privilege to execute commands and stored procedures and call database-supplied procedures.
- A standalone SQLEXEC statement cannot be used to get input parameters from records or pass output parameters. You can use stored procedures and queries with parameters by using a SQLEXEC statement within a TABLE or MAP statement.
- 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 of, or delete an object, before the SQLEXEC procedure or query executes on it.

For additional instructions for using stored procedures and queries with Oracle GoldenGate, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

NOTE For DB2 on z/OS, 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.

Syntax

```
Procedures:

SQLEXEC “call <procedure name>()”
[EVERY <n> {SECONDS | MINUTES | HOURS | DAYS}]
[ONEXIT]
```

Syntax

```
Queries:

SQLEXEC “<sql query>”
[EVERY <n> {SECONDS | MINUTES | HOURS | DAYS}]
[ONEXIT]
```

Syntax

```
Database commands:

SQLEXEC “<database command>”
[EVERY <n> {SECONDS | MINUTES | HOURS | DAYS}]
[ONEXIT]
```

Component	Description
`“call <procedure name> ()”`	Specifies the name of a stored procedure to execute. The statement must be enclosed within double quotes. The call keyword is required. Example: `SQLEXEC “call prc_job_count ()”`

`“<sql query>”`	Specifies the name of a query to execute. Enclose the query within quotes. To use quoted object names within a SQLEXEC query, the SQL query must be enclosed within single quotes, rather than double quotes, and the USEANSISQLQUOTES parameter must be used in the GLOBALS file to enforce SQL-92 rules for object and literal identifiers. The following is an example of using quoted object names in a query:
	`SQLEXEC 'SELECT "col1" from "schema"."table"'`
	For a multi-line query, use quotes on each line. For best results, type a space after each begin quote and before each end quote (or at least before each end quote).
	Example:
	`SQLEXEC “ select x from dual ”`

`EVERY <n>`
`{SECONDS | MINUTES |`
`HOURS | DAYS}`
	Causes a standalone stored procedure or query to execute at a defined interval, for example:
	`SQLEXEC "call prc_job_count ()" EVERY 30 SECONDS`
	The interval must be a whole, positive integer.

`ONEXIT`	Executes the SQL when the Extract or Replicat process stops gracefully.
`“<database command>”`	Executes a database command.

Example 1 `SQLEXEC “call prc_job_count ()”`

Example 2 `SQLEXEC “ select x from dual ”`

Example 3 `SQLEXEC "call prc_job_count ()" EVERY 30 SECONDS`

Example 4 `SQLEXEC “call prc_job_count ()” ONEXIT`

Example 5 `SQLEXEC “SET TRIGGERS OFF”`

STARTUPVALIDATIONDELAY[CSECS]

Valid for Manager

Use the STARTUPVALIDATIONDELAY or STARTUPVALIDATIONDELAYCSECS parameter to set a delay time after which Manager validates the status of a process that was started with the START EXTRACT or START REPLICAT command. If a process is not running after the specified delay time, an error message is displayed at the GGSCI prompt.

These parameters account for processes that fail before they can generate an error message or report, for example when there is not enough memory to launch the processes. Startup validation makes Oracle GoldenGate users aware of such failures.

Default 0 seconds (do not validate startup status)

Syntax

```
STARTUPVALIDATIONDELAY <seconds> | STARTUPVALIDATIONDELAYCSECS <centiseconds>
```

Argument	Description
`<seconds>	<centiseconds>`	The number of seconds or centiseconds to delay before checking the status of a process.

Example In the following example, Manager waits ten centiseconds after a START command is issued and then checks the status of the process.

```
STARTUPVALIDATIONDELAYCSECS 10
```

STATOPTIONS

Valid for Extract and Replicat

Use the STATOPTIONS parameter to specify information to be included in statistical displays generated by the STATS EXTRACT or STATS REPLICAT command. These options also can be enabled as needed as arguments to those commands.

Default See individual options.

Syntax

```
STATOPTIONS
[, REPORTDETAIL | NOREPORTDETAIL]
[, REPORTFETCH | NOREPORTFETCH]
[, RESETREPORTSTATS | NORESETREPORTSTATS]
```

Argument	Description

`REPORTFETCH |`
`NOREPORTFETCH`
	Valid for Extract. REPORTFETCH returns statistics on row fetching, such as that triggered by a FETCHCOLS clause (see page 354) or fetches that must be performed when not enough information is in the transaction record. NOREPORTFETCH turns off reporting of fetch statistics. The default is NOREPORTFETCH .

`REPORTDETAIL |`
`NOREPORTDETAIL`
	Valid for Replicat. REPORTDETAIL returns statistics on operations that were not replicated as the result of collision errors. These operations are reported in the regular statistics (inserts, updates, and deletes performed) plus as statistics in the detail display, if enabled. For example, if 10 records were insert operations and they were all ignored due to duplicate keys, the report would indicate that there were 10 inserts and also 10 discards due to collisions.
	NOREPORTDETAIL turns off reporting of collision statistics. The default is REPORTDETAIL.

Argument	Description
`RESETREPORTSTATS | NORESETREPORTSTATS`	Valid for Extract and Replicat. Controls whether or not statistics generated by the REPORT parameter are reset when a new process report is created. The default of NORESETREPORTSTATS continues the statistics from one report to another as the report rolls over based on the REPORTROLLOVER parameter. To reset statistics, use RESETREPORTSTATS.

SYSLOG

Valid for GLOBALS, Manager

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.

Default Write all Oracle GoldenGate messages to the system log.

Syntax

```
SYSLOG {[ALL | NONE] | [, INFO] [, WARN] [, ERROR]}
```

Argument	Description
`ALL`	Sends all INFO (information), WARN (warning), and ERROR (error) messages to the system logs. This is the default and is the same as: `SYSLOG INFO, WARN, ERROR` Cannot be combined with other options.
`NONE`	Prevents Oracle GoldenGate messages from being written to the system logs. Cannot be combined with other options.
`INFO`	Sends messages that are reported as INFO to the system logs. Can be combined with WARN and ERROR in any order.
`WARN`	Sends messages that are reported as WARN to the system logs. Can be combined with INFO and ERROR in any order.
`ERROR`	Sends messages that are reported as INFO to the system logs. Can be combined with INFO and WARN in any order.

Example Either of the following statements sends warning and error messages to the system logs, but does not send informational messages.

```
SYSLOG WARN, ERROR
```

or:

```
SYSLOG ERROR, WARN
```

TABLE for DEFGEN

Use the TABLE parameter in a DEFGEN parameter file to identify a source table or tables for which you want to run the utility. Each TABLE statement must be terminated with a semi-colon.

Default None

Syntax `TABLE <[owner.]table>[, DEF <definitions template>];`

Argument	Description
`<[owner.]table>`	The owner (optional) and name of the table. The owner is optional except for Oracle tables. This parameter accepts wildcard (*) arguments for tables only. Oracle GoldenGate will automatically increase internal storage to track up to 100,000 wildcard entries.
`DEF <definitions template>`	Specifies a definitions template to be based on this table's definitions. Enables new tables with the exact same definitions to be added later, without having to run DEFGEN for them and without having to stop and start the Oracle GoldenGate process. The template is specified with the DEF or TARGETDEF option of the TABLE or MAP statement. This option is not supported for initial loads.

Example 1 `TABLE fin.account;`

Example 2 `TABLE fin.acc*;`

Example 3 `TABLE fin.acct1, DEF acctdefs;`

TABLE for Extract

Valid for Extract

Use the TABLE parameter in an Extract parameter file to specify objects for extraction by Oracle GoldenGate. TABLE is valid for:

- Online Extract to support the capture of data changes.
- Initial-load Extract to support the extraction of complete data records from source tables.

NOTE To map the source objects that you capture with TABLE to target objects for the purpose of replication, specify the source and target tables with a MAP parameter statement in the Replicat parameter file.

You can specify the following objects with TABLE:

- Indexes
- Triggers
- Materialized views
- Tables

NOTE To specify a sequence for capture, use the SEQUENCE parameter.

Limitations of support

For tables, you can use all of the TABLE options. These include, but are not limited to:

- Select and filter rows of tables
- Map columns of tables
- Transform data
- Specify key columns and before values
- Execute stored procedures and queries
- Define user tokens
- Trim trailing spaces
- Pass a parameter to a user exit

For non-table objects, use TABLE only to specify an object for capture.

NOTE Oracle GoldenGate supports the replication of the actual data values of Oracle materialized views. Oracle GoldenGate supports the replication of Oracle and Teradata DDL for indexes and triggers, but not the content of those objects. See the Oracle GoldenGate *Windows and UNIX Administrator's Guide* for more information about DDL support.

Default None

Syntax

```
TABLE <table spec> [, TARGET <table spec>]
[, DEF <definitions template>]
[, TARGETDEF <definitions template>]
[, COLMAP (<column mapping expression>)]
[, {COLS | COLSEXCEPT} (<column specification>)]
[, EVENTACTIONS <action>]
[, EXITPARAM "<parameter string>"]
[, {FETCHCOLS | FETCHCOLSEXCEPT} (column specification)]
[, {FETCHMODCOLS | FETCHMODCOLSEXCEPT} (<column spec>)]
[, FETCHBEFOREFILTER]
[, FILTER (<filter specification>)]
[, GETBEFORECOLS(
   {ON UPDATE | ON DELETE}
   {ALL |
   KEY |
   KEYINCLUDING (<col>[,...]) |
   ALLEXCLUDING (<col>[,...]) }
   [,...]
   )]
[, KEYCOLS (<column specification>)]
[, SQLEXEC (<SQL specification>)]
[, SQLPREDICATE "WHERE <where clause>"]
[, TOKENS (<token specification>)]
[, TRIMSPACES | NOTRIMSPACES]
[, TRIMVARSPACES | NOTRIMVARSPACES]
[, WHERE (<where clause>)]
;
```

Table 40 Summary of TABLE syntax components

Component	Description
TABLE `<table spec>`	Specifies the source table. For supported characters and wildcards, see the Oracle GoldenGate *Administration Guide*.
TARGET `<table spec>`	Specifies a target table to which the source table will be mapped. For guidelines for specifying object names, see the Oracle GoldenGate *Administration Guide*.
DEF `<definitions template>`	Specifies a source-definitions template.
TARGETDEF `<definitions template>`	Specifies a target-definitions template.
COLMAP	Maps records between different source and target columns.
COLS | COLSEXCEPT	Selects or excludes columns for processing.
EVENTACTIONS (`<action>`)	Triggers an action based on a record that satisfies a specified filter rule.

Table 40 Summary of TABLE syntax components (continued)

Component	Description
EXITPARAM	Passes a parameter in the form of a literal string to a user exit.
FETCHCOLS FETCHCOLSEXCEPT	Enables the fetching of column values from the source database when the values are not in the transaction record.
FETCHBEFOREFILTER	Directs the FETCHCOLS or FETCHCOLSEXCEPT action to be performed before a filter is executed.
FETCHMODCOLS | FETCHMODCOLSEXCEPT	Forces column values to be fetched from the database when the columns are present in the transaction log.
FILTER	Selects records based on a numeric value. FILTER provides more flexibility than WHERE.
GETBEFORECOLS	Forces before images of columns to be captured and written to the trail.
KEYCOLS	Designates columns that uniquely identify rows.
SQLEXEC	Executes stored procedures and queries.
SQLPREDICATE	Enables a WHERE clause to select rows for an initial load.
TOKENS	Defines user tokens.
TRIMSPACES | NOTRIMSPACES	Controls whether trailing spaces are trimmed or not when mapping CHAR to VARCHAR columns.
TRIMVARSPACES | NOTRIMVARSPACES	Controls whether trailing spaces are trimmed or not when mapping VARCHAR to CHAR or VARCHAR columns.
WHERE	Selects records based on conditional operators.

Specifying object names in the TABLE clause

To specify the object names in the TABLE clause, see “Getting started with the Oracle GoldenGate process interfaces” in the Oracle GoldenGate *Administration Guide*.

Using TARGET

TARGET is required for TABLE statements when Extract must refer to a target definitions file (specified with the TARGETDEFS parameter) to perform conversions or when the COLMAP option is used. Otherwise, it can be omitted. Using TARGET identifies the extracted data by the target structure, rather than that of the source, to reflect the structure of the record that is reflected in the definitions file or the column map.

Column mapping and conversion can be performed on the target system to prevent added overhead on the source system. However, replication from a Windows or UNIX system to a NonStop system require these functions to be performed on the source.

In addition, it may be preferable to perform the mapping and conversion on the source when there are multiple sources and one target. In that case, it could be easier to manage one target definitions file that is transferred to each source, rather than having to manage source definitions for each source database that must be transferred to the target, especially when there are frequent application changes that require new files to be generated.

Using COLMAP

Use COLMAP to explicitly map source columns to target columns that have different names or to specify default column mapping when source and target names are identical. COLMAP provides instructions for selecting, translating, and moving column data. Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

NOTE To create *global* rules for column mapping across all tables in subsequent TABLE statements, use the COLMATCH parameter.

Supporting case and special characters in column names

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

Generating data definitions

When using COLMAP for source and target tables that are not identical in structure, you must generate data definitions for the source tables, transfer them to the target, and use the SOURCEDEFS parameter to identify the definitions file.

For source and target structures to be considered identical, they must contain identical column names (including case, if applicable) and data types, and the columns must be in the same order in each table. In addition, both tables must have the same column length semantics for character columns (bytes versus characters).

If the tables have identical structures, and you are using COLMAP for other functions such as conversion, a source definitions file is not needed. You can use the ASSUMETARGETDEFS parameter instead.

For more information, see:

- SOURCEDEFS on page 330
- ASSUMETARGETDEFS on page 127
- "Creating a data-Definitions file" in the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Using default column mapping

For any corresponding source and target columns whose names are identical, you can use default mapping instead of using an explicit mapping statement. Default mapping causes Oracle GoldenGate to map those columns automatically. Data translation, if any, is automatic.

To use default mapping, use the USEDEFAULTS option. Default mapping is only enabled for columns that are not mapped already with an explicit mapping statement.

By default SQL Server and Sybase columns are compared with case-sensitivity taken into account. For all other databases, the column names are changed to upper case for name comparison. To support case-sensitive column names for those databases, use the USEANSISQLQUOTES parameter in the GLOBALS file. This applies SQL-92 rules, which require column names to be enclosed within double quotes and literals to be enclosed within single quotes.

Where case is recognized, USEDEFAULTS supports case sensitivity in the following manner:

- If a source column is found whose name and case exactly matches that 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 the target column, regardless of case.

For example, the following are source and target tables that contain case-sensitive columns.

Source table USER1.SM01:

```
id
owner
created
changed
creator
modifiedBy
comment
COMMENT
```

Target table USER3.SM01:

```
ID
owner
id
Creator
comment
ModifiedBy
creationDate
alterationDate
Comment
COMMENT
```

The following column map for these tables contains both explicit and default column mappings:

```
TABLE USER1.SM01, TARGET USER3.SM01,
COLMAP (USEDEFAULTS,
   ID = id,
   creationDate = created,
   alterationDate = changed,
   );
```

The following is the result of this map. For default mapping, case-sensitivity is observed when applicable, but otherwise just the names are matched. Two target columns are not mapped because they were not explicitly mapped and no default map could be established.

Mapping type	**Mapping result**

Explicit mapping:	`ID = id,`
	`creationDate = created,`
	`alterationDate = changed`

Default mapping:	`owner = owner,`
	`comment = comment,`
	`COMMENT = COMMENT,`
	`Creator = creator,`
	`ModifiedBy = modifiedby`

Target columns not mapped:	`id, Comment`

For more information about column mapping, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

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

Component	Description
`<table spec>`	The source or target table.

`<target column> =`
`<source expression>`
	Explicitly defines a source-target column map. For supported characters in column names, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.
	<target column> is the name of the target column.
	<source expression> can be any of the following:
	◆ The name of a source column, such as ORD_DATE
	◆ Numeric constant, such as 123
	◆ String constant within quotes, such as "ABCD"
	◆ An expression using an Oracle GoldenGate column-conversion function, such as @STREXT (COL1, 1, 3). For descriptions of the column-conversion functions, see Chapter 4.

Component	Description
`USEDEFAULTS`	Automatically maps source and target columns that have the same name if they were not specified in an explicit column map. Use an explicit map or USEDEFAULTS, but not both for the same set of columns. See "Using default column mapping" on page 342 for more information. Specify USEDEFAULTS before explicit column maps.

Example 1

```
TABLE ggs.tran, TARGET ggs.tran2, COLMAP (loc2 = loc, type2 = type);
```

Example 2

```
TABLE ggs.tran, TARGET ggs.tran2, COLMAP COLMAP ( EUROVAL = "\u20ac0" );
```

Example 3

```
TABLE ggs.tran, TARGET ggs.tran2, COLMAP (SECTION = @STRCAT("\u00a7", SECTION
));
```

Using COLS and COLSEXCEPT

Use COLS and COLSEXCEPT to control column selection.

- Use COLS to specify columns whose data you want to synchronize. All other columns are ignored by Oracle GoldenGate.
- Use COLSEXCEPT to exclude columns from synchronization. All other columns are processed by Oracle GoldenGate. For tables with numerous columns, COLSEXCEPT may be more efficient than listing each column with COLS.

WARNING Do *not* exclude key columns, and do *not* use COLSEXCEPT to exclude columns that contain data types that are not supported by Oracle GoldenGate. COLSEXCEPT does not exclude unsupported data types.

To use COLS, the following is required:

- The table must have one or more key columns, or else a substitute key must be defined with the KEYCOLS option of TABLE.
- The key columns or the columns specified with KEYCOLS must be included in the column list specified with COLS. Otherwise, they will not be captured, and an error will be generated during processing. (Note: Without COLS, key columns are automatically captured.)

Without a primary or unique key or, in the absence of those, a KEYCOLS clause in the TABLE statement, Oracle GoldenGate uses all of the columns in the table, rendering COLS unnecessary.

Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

Syntax

```
TABLE <table spec>, {COLS | COLSEXCEPT} (<column> [, ...]) ;
```

Component	Description

`<column>`	The name of a column. To specify multiple columns, create a comma-delimited list as in the following examples.
	The following processes only columns 1 and 3.
	`TABLE hq.acct, COLS (col1, col3);`
	The following processes all columns except column 4.
	`TABLE hq.acct, COLSEXCEPT (col4);`

NOTE If the database uses compressed updates (where column values are not logged unless they changed), a column specified for extraction with COLS might not be available. To make these columns available, use the FETCHCOLS option in the TABLE statement or enable supplemental logging for the column.

Using DEF

Use DEF to specify a source-definitions template. The definitions template is created based on the definitions of a specific source table when DEFGEN is run for that table. Once the template is created, new source tables that have identical definitions to that table can be added without having to run DEFGEN for them, and without having to stop and start Extract. The definitions in the template specified with DEF will be used for definitions lookups. For more information about DEFGEN, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

```
TABLE <table spec>, DEF <definitions template>;
```

Argument	Description
`<definitions template>`	The name of a definitions template that was specified with the DEF option of TABLE in the DEFGEN parameter file. The definitions contained in the template must be identical to the definitions of the table in this TABLE statement.

Example

```
TABLE acct.cust*, DEF custdef;
```

Using EVENTACTIONS

Use EVENTACTIONS to cause the Extract process to take a defined action based on a record in the transaction log, known as the *event record*, that qualifies for a specific filter rule. You can use this system to customize Oracle GoldenGate processing based on database events.

WARNING EVENTACTIONS is not supported if the source database is Teradata and Extract is configured in maximum performance mode.

Examples of how to use this system would be to start, suspend, or stop a process, to perform a transformation, or to report statistics. The event marker system can be put to use for purposes such as:

- To establish a synchronization point at which SQLEXEC or user exit functions can be performed
- To execute a shell command that executes a data validation script or sends an email

- To activate tracing when a specific account number is detected
- To capture lag history
- To stop or suspend a process to run reports or batch processes at the end of the day

The event marker feature is supported for the replication of data changes, but not for initial loads.

Using 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.

   NOTE Many, but not all, of the EVENTACTIONS options apply to both TABLE (for Extract) and MAP (for Replicat), so all of the options for both processes are shown here. Exceptions are noted.

Combining 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
  - SUSPEND
  - ABORT
  - CHECKPOINT AFTER
  - FORCESTOP
  - STOP

Controlling 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 | SUSPEND | ABORT | FORCESTOP]
[IGNORE [RECORD | TRANSACTION [INCLUDEVENT]]
[DISCARD]
[LOG [INFO | WARNING]]
[REPORT]
[ROLLOVER]
[SHELL "<command>" |
   SHELL ("<command>", VAR <variable> = {<column name> | <expression>}
   [, ...][, ...]) ]
[TRACE[2] <trace file> [TRANSACTION] [DDL[INCLUDE] | DDLONLY]
   [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.

Action	Description
SUSPEND	Pauses the process so that it retains the active context of the current run and can still respond to SEND commands that are issued in GGSCI. When a process is suspended, the INFO command shows it as RUNNING, and the RBA field shows the last checkpoint position. To resume processing, issue the SEND <group> command with the RESUME option. To use the CHECKPOINT BEFORE option in conjunction with SUSPEND, the event record must be the start of a transaction for the SUSPEND to take place. That way, if the process is killed while in the suspended state, the event record with the SUSPEND action is the first record to be reprocessed upon restart. If both CHECKPOINT BERORE and SUSPEND are specified, but the event record is not the start of a transaction, the process abends before SUSPEND can take place. To use the CHECKPOINT AFTER option in conjunction with SUSPEND, the RESUME command must be issued before the checkpoint can take place, and the event record must be a COMMIT record. If the process is killed while in a SUSPEND state, the process reprocesses the transaction from the last checkpointed position upon restart. SUSPEND cannot be combined with ABORT but can be combined with all other options.
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.
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.

Action	Description

`IGNORE`
`[RECORD | TRANSACTION`
`[INCLUDEVENT]]`
	Ignores some or all of the transaction, depending on the selected action.
	- RECORD is the default. It forces the process to ignore only the specified event record, but not the rest of the transaction. 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.
	This action is not valid for DDL records. Because DDL operations are autonomous, ignoring a record is equivalent to ignoring the entire transaction.

`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.
`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.

Action	Description
`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.
`SHELL "<command>"`	Causes the process to execute the specified shell command when the event record is encountered. SHELL "<command>" executes a basic shell command. The command string is taken at its literal value and sent to the system that way. The command is case-sensitive and must be enclosed within double quote marks, for example: `EVENTACTIONS (SHELL "echo hello world! > output.txt")` 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. Note that the error message relates only to the execution of the SHELL command itself, and not the exit status of any subordinate commands. For example, SHELL can execute a script successfully, but commands in that script could fail. SHELL can be combined with all other EVENTACTIONS options.

Action	Description
`SHELL ("<command>", VAR <variable> = {<column name> | <expression>} [, ...][, ...])`	Causes the process to execute the specified shell command when the event record is encountered and supports parameter passing. The command and the parameters are case-sensitive. By default, any input value that is within double quotes is treated as literal text. However, you can use the USEANSISQLQUOTES parameter in the GLOBALS file to apply SQL-92 rules, where text enclosed in single quotes is treated as a literal, and text enclosed in double quotes is treated as a name. See also USEANSISQLQUOTES. When SHELL is used with arguments, the entire command and argument strings must be enclosed within parentheses, for example: `EVENTACTIONS (SHELL ("Current timestamp: $1  SQLEXEC result is $2 ",VAR $1 = @GETENV("JULIANTIMESTAMP"),VAR $2 = mytest.description));` The input is as follows: <command> is the command, which is passed literally to the system. VAR is a required keyword that starts the parameter input. <variable> is the user-defined name of the placeholder variable where the run-time variable value will be substituted. Extra variables that are not used in the command are ignored. Note that any literal in the SHELL command that matches a VAR variable name is replaced by the substituted VAR value. This may have unintended consequences, so test your code before putting it into production. <column name > can be the before or after (current) image of a column value. <expression> can be the following, depending on whether column data or DDL is being handled. **Column data valid expressions:** ◆ The value from a TOKENS clause in a TABLE statement. ◆ A return value from any Oracle GoldenGate column-conversion function. ◆ A return value from a SQLEXEC query or procedure.

Action	Description
	DDL valid expressions: ◆ Return value from @TOKEN function (Replicat only). ◆ Return value from @GETENV function. ◆ Return value from other functions that do not reference column data (for example, @DATENOW). ◆ Return value from @DDL function.

`TRACE[2] <trace file>`
`[TRANSACTION]`
`[DDL[INCLUDE] | DDLONLY]`
`[PURGE | APPEND]`
	Causes process trace information to be written to a trace file when the specified event record is encountered. TRACE provides step-by-step processing information. TRACE2 identifies the code segments on which the process is spending the most time.
	By default (without options), standard DML tracing without consideration of transaction boundaries is enabled until the process terminates.
	◆ <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. This option is valid only for DML operations.
	◆ DDL[INCLUDE] traces DDL and also DML transactional data processing. Either DDL or DDLINCLUDE is valid.
	◆ DDLONLY traces DDL but does not trace DML transactional data.
	These options are valid only for Replicat. By default DDL tracing is disabled.
	◆ 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.

Action	Description
	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.
`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. When using EVENTACTIONS for a DDL record, note that since each DDL record is autonomous, the DDL record is guaranteed to be the start of a transaction; therefore the CHECKPOINT BEFORE event action is implied for a DDL record. 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. When using EVENTACTIONS for a DDL record, note that since each DDL record is autonomous, the DDL record is guaranteed to be the end (boundary) of a transaction; therefore the CHECKPOINT AFTER event action is implied for a DDL record. CHECKPOINT AFTER can be combined with all EVENTACTIONS options except ABORT.

Action	Description
	◆ 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.

Example 1 The following enables tracing for a transaction that contains an insert operation for a specific order number.

```
TABLE source.order, FILTER (@GETENV ("GGHEADER", "OPTYPE") = "INSERT" AND
order_no = 1), EVENTACTIONS (TRACE order_1.trc TRANSACTION);
```

Example 2 This example shows how to configure Extract to roll over the trail to the next file in the sequence at the end of a defined processing period. A set of trail files then can be bound as a unit based on that period. Here is how it works:

1. When Extract encounters a record that satisfies the FILTER clause, a ROLLOVER event action is logged to the source database.
2. Upon capturing the record (the *event record*) in the transaction log, the Extract process closes the current trail file and opens a new trail file.
3. The ROLLOVER event action is combined with an IGNORE action to prevent the event record itself from being written to the trail file.

```
TABLE source.event_table, FILTER (@GETENV ("GGHEADER", "OPTYPE") =
"INSERT" AND order_no = 10,000), EVENTACTIONS (ROLLOVER, IGNORE);
```

For additional use cases and more information about the event marker system, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Using EXITPARAM

Use EXITPARAM to pass a parameter to a user exit routine whenever a record from the TABLE statement is encountered. Do not use this option for tables being processed in pass-through mode by a data-pump Extract group. For more information about user exits, see Chapter 5.

Syntax

```
TABLE <table spec>, EXITPARAM “<parameter string>”;
```

Component	Description
`“<parameter string>”`	A parameter that is a literal string. Enclose the parameter within double quotes. You can specify up to 100 characters for the parameter string.

Using FETCHCOLS and FETCHCOLSEXCEPT

Use FETCHCOLS and FETCHCOLSEXCEPT to fetch column values from the database when the values are not present in the transaction log record. Use this option if the database uses compressed updates (where column values are not logged unless they changed), but you need to ensure that other column values required for FILTER operations are available.

- FETCHCOLS fetches the specified column(s).
- FETCHCOLSEXCEPT fetches all columns except those specified. For tables with numerous columns, FETCHCOLSEXCEPT may be more efficient than listing each column with FETCHCOLS.

FETCHCOLS and FETCHCOLSEXCEPT are valid for all databases that are supported by Oracle GoldenGate.

For an Oracle database, Oracle GoldenGate fetches the values from the undo tablespace through Oracle's Flashback Query mechanism. The query provides a read-consistent image of the columns as of a specific time or SCN. For more information about how Oracle GoldenGate uses Flashback Query, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Instead of using FETCHCOLS or FETCHCOLSEXCEPT, it may be more efficient to enable supplemental logging for the desired columns.

For Sybase, encrypted column data is not supported by these parameters because Oracle GoldenGate does not support Sybase encrypted data.

To control fetching and enable a response when a column specified for fetching cannot be located, use the FETCHOPTIONS parameter. To include fetch results in statistical displays generated by the STATS EXTRACT command, use the STATOPTIONS parameter.

If values for columns specified with FETCHCOLS or FETCHCOLSEXCEPT are present in the transaction log, no database fetch is performed. This reduces database overhead.

Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

Syntax

```
TABLE <table spec>, {FETCHCOLS | FETCHCOLSEXCEPT} (<column> [, ...]) ;
```

Component	Description
`<column>`	Can be one of the following: ◆ A column name or a comma-delimited list of column names, as in `(COL1, COL2)`. ◆ An asterisk wildcard, as in (*).

Using FETCHMODCOLS and FETCHMODCOLSEXCEPT

Use FETCHMODCOLS and FETCHMODCOLSEXCEPT to force column values to be fetched from the database even if the columns are present in the transaction log. Depending on the database type, a log record can contain all of the columns of a table or only the columns that changed in the given transaction operation.

- FETCHMODCOLS fetches the specified column(s).
- FETCHMODCOLSEXCEPT fetches all columns present in the transaction log, except those specified. For tables with numerous columns, FETCHMODCOLSEXCEPT might be more efficient than listing each column with FETCHMODCOLS.

This option is valid for all databases that are supported by Oracle GoldenGate.

Limitations of use

- Do not use FETCHMODCOLS and FETCHMODCOLSEXCEPT for key columns.
- Do not use FETCHMODCOLS and FETCHMODCOLSEXCEPT for tables that are being processed in pass-through mode by a data-pump Extract group (PASSTHRU parameter in the parameter file). A database login is not supported by that processing mode.
- (Sybase) Do not use FETCHMODCOLS and FETCHMODCOLSEXCEPT for encrypted column data. Oracle GoldenGate does not support Sybase encrypted data.

Default None

Syntax

```
TABLE <table spec>, {FETCHMODCOLS | FETCHMODCOLSEXCEPT} (<column spec>);
```

Argument	Description
`(<column spec>)`	Can be one of the following: ◆ A column name or a comma-delimited list of column names, as in `(COL1, COL2)`. ◆ An asterisk wildcard, as in (*).

Using FETCHBEFOREFILTER

Use FETCHBEFOREFILTER to fetch columns specified with FETCHCOLS or FETCHCOLSEXCEPT before a FILTER operation is executed. Fetching beforehand ensures that values required for the filter are available. Without FETCHBEFOREFILTER, fetches specified with FETCHCOLS or FETCHCOLSEXCEPT are not performed until after filters are executed.

Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

Syntax

```
TABLE <table spec>, FETCHCOLS (<column> [, ...]),
FETCHBEFOREFILTER,
FILTER <filter clause>
;
```

Using FILTER

Use FILTER to select or exclude records based on a numeric value. A filter expression can use conditional operators, Oracle GoldenGate column-conversion functions, or both.

NOTE To filter based on a string, use one of the Oracle GoldenGate string functions (see Chapter 4) or use the WHERE option.

Separate all FILTER components with commas. A FILTER clause can include the following:

- 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

Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

Syntax

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

Component	Description

`<filter clause>`	Selects records based on an expression, such as:
	`FILTER ((PRODUCT_PRICE*PRODUCT_AMOUNT)>10000))`
	You can use the column-conversion functions of Oracle GoldenGate in a filter clause, as in:
	`FILTER (@COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`
	By default, Oracle GoldenGate treats any input string within double quotes as a literal, as in `FILTER (@STRFIND(NAME, "JOE") > 0)`. To use SQL-92 rules for identifiers and literals, use the USEANSISQLQUOTES parameter in the GLOBALS file.
	Oracle GoldenGate does not support FILTER for columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.
	The maximum size of the filter clause is 5,000 bytes.

`ON INSERT |`
`ON UPDATE|`
`ON DELETE`
	Restricts record filtering to the specified operation(s). Separate operations with commas, for example:
	`FILTER (ON UPDATE, ON DELETE, @COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`
	This example executes the filter for updates and deletes, but not inserts.

Component	Description

`IGNORE INSERT |`
`IGNORE UPDATE |`
`IGNORE DELETE`
	Does not apply the filter for the specified operation(s). Separate operations with commas, for example:
	`FILTER (IGNORE INSERT, @COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`
	This example executes the filter on updates and deletes, but ignores inserts.

Using GETBEFORECOLS

Use GETBEFORECOLS to specify columns for which you want before image to be captured and written to the trail upon an update or delete operation. Use GETBEFORECOLS when using the Oracle GoldenGate Conflict Detection and Resolution (CDR) feature in a bi-directional or multi-master configuration. For updates, the before image of the specified columns is included in the trail whether or not any given column is modified. To use this parameter, supplemental logging must be enabled for any database that does not log before values by default.

GETBEFORECOLS overrides COMPRESSUPDATES and COMPRESSDELETES if used in the same parameter file.

This parameter is valid for all databases except DB2. For DB2 on all platforms that are supported by Oracle GoldenGate, use the GETUPDATEBEFORES parameter instead of GETBEFORECOLS.

Syntax

```
TABLE <table spec> , GETBEFORECOLS(
{ON UPDATE | ON DELETE}
{ALL |
KEY |
KEYINCLUDING (<col>[,...]) |
ALLEXCLUDING (<col>[,...]) }
[,...]
)
```

Argument	Description
`{ON UPDATE | ON DELETE}`	Specifies whether the before image of the specified columns should be captured for updates or deletes. You can use ON UPDATE only, ON DELETE only, or both. If using both, specify them within the same GETBEFORECOLS clause. See the example for how to use both.

Argument	Description

`{ALL | KEY |`
`KEYINCLUDING (<col>[,...]) |`
`ALLEXCLUDING (<col>[,...])}`
	Specifies the columns for which a before image is captured.
	ALL: capture before image of all columns, including the primary key. This imposes the highest processing load for Extract, but allows conflict-detection comparisons to be performed using all columns for maximum accuracy.
	KEY: capture before image only for the primary key. This is the fastest option, but does not permit the most accurate conflict detection, because keys can match but non-key columns could be different. KEY is the default.
	KEYINCLUDING: capture before image of the primary key and also the specified column or columns. This is a reasonable compromise between speed and detection accuracy.
	ALLEXCLUDING: capture before image of all columns except the specified columns. For tables with numerous columns, ALLEXCLUDING may be more efficient than KEYINCLUDING. Do *not* exclude key columns.

Example In the following example, the before images for the key column(s) plus the name, address, and salary are always written to the trail file on update and delete operations.

```
TABLE src,
GETBEFORECOLS (
ON UPDATE KEYINCLUDING (name, address,  salary),
ON DELETE KEYINCLUDING (name, address, salary));
```

Using KEYCOLS

Use KEYCOLS to define one or more columns of the target table as unique. The primary use for KEYCOLS is to define a substitute primary key when a primary key or unique index is not available for the table.

Source and target key or unique-index columns must match, whether they are defined in the database or substitutes rendered by KEYCOLS. The source table must contain at least as many key or index columns as the target table. Otherwise, in the event of an update to the source key or index columns, Replicat will not have the before images for the extra target columns.

When defining keys, observe the following guidelines:

- If both the source and target tables lack keys or unique indexes, use KEYCOLS in both the TABLE and MAP statements, and specify matching sets of columns.
- If just one of the tables lacks a key or unique index, use KEYCOLS for that table, and specify columns that match the actual key or index columns of the other table. If a matching set cannot be defined, then use KEYCOLS in both the TABLE and MAP statements, and specify matching sets of columns that contain unique values. The KEYCOLS specification will override the existing key or index.

- If the target table has a larger key than the source table does (or more unique-index columns), KEYCOLS should be used in the TABLE statement to specify the actual source key or index columns, plus the source columns that match the extra target columns. Do not just specify the extra columns, because when a table has a primary key or unique index, the KEYCOLS specification will override them. Using KEYCOLS in this way ensures that before images are available for updates to the key or index columns.

When using KEYCOLS, make certain that the specified columns are logged to the transaction log so that they are available to Replicat in the trails. You can do so by using the database interface or by using the COLS option of the ADD TRANDATA command (Oracle only).

On the target tables, create a unique index on the KEYCOLS-defined key columns. An index improves the speed with which Oracle GoldenGate locates the target rows that it needs to process.

Do not use KEYCOLS for tables being processed in pass-through mode by a data-pump Extract group.

Syntax

```
TABLE <table spec>, KEYCOLS (<column> [, ... ]);
```

Component	Description
`(<column>)`	Defines a column to be used as a substitute primary key. To specify multiple columns, create a comma-delimited list as in: `KEYCOLS (id, name)` If a primary or unique key exists, those columns must be included in the KEYCOLS specification. The following column-types are not supported in KEYCOLS: **Oracle column types not supported by KEYCOLS**: Virtual columns, UDTs, function-based columns, and any columns that are explicitly excluded from the Oracle GoldenGate configuration **SQL Server, DB2 LUW, DB2 z/OS, MySQL, SQL/MX, Teradata, TimesTen column types not supported by KEYCOLS**: Columns that contain a timestamp or non-materialized computed column, and any columns excluded from the Oracle GoldenGate configuration. For SQL Server Oracle GoldenGate enforces the total length of data in rows for target tables without a primary key to be below 8000 bytes. **Sybase column types not supported by KEYCOLS**: Computed columns, function-based columns, and any columns that are explicitly excluded from the GoldenGate configuration

Using SQLEXEC

Use SQLEXEC to execute a SQL stored procedure or query from within a TABLE statement during Oracle GoldenGate processing. SQLEXEC enables Oracle GoldenGate to communicate directly with the database to perform any function supported by the database. The database function can be part of the synchronization process, such as retrieving values for column conversion, or it can be independent of extracting or replicating data.

When used within a TABLE statement, the procedure or query that is executed can accept input parameters from source or target rows and pass output parameters.

For additional instructions for using stored procedures and queries with Oracle GoldenGate, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

SQLEXEC dependencies and restrictions

- The SQL is executed by the user under which the Oracle GoldenGate process is running. This user must have the privilege to execute stored procedures and call database-supplied procedures.
- A query or procedure must be structured correctly when executing a SQLEXEC statement, with legal SQL syntax for the database; otherwise Oracle GoldenGate will abend, regardless of any error-handling rules that are in place. Refer to the SQL reference guide provided by the database vendor for permissible SQL syntax.
- Do not use SQLEXEC to change a value in a primary key column. The primary key value is passed from Extract to Replicat. Without it, Replicat operations cannot be completed. If primary key values must be changed with SQLEXEC, you may be able to avoid errors by mapping the original key value to another column and then defining that column as a substitute key with the KEYCOLS option. See "Using KEYCOLS" on page 359.
- For DB2 on z/OS, 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 for z/OS documentation for more information.
- Do not use SQLEXEC for tables being processed in pass-through mode by a data-pump Extract group.
- When using Oracle GoldenGate DDL support, all objects that are affected by a 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.

Supported data types

The following are the data types that are supported by SQLEXEC for input and output parameters.

- Numeric data types
- Date data types
- Character data types

For additional instructions for using stored procedures and queries with Oracle GoldenGate, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Using SQLEXEC with stored procedures

To execute a stored procedure from within a TABLE statement, use the SPNAME clause.

Syntax

```
SQLEXEC (
SPNAME <sp name>
[, ID <logical name>]
{, PARAMS <param spec> | NOPARAMS}
[, <option>] [, ...]
)
```

Component	Description
`<sp name>`	Specifies the name of the stored procedure.
`ID <logical name>`	Defines a logical name for the procedure. Use this option to execute the procedure multiple times within a TABLE statement. Up to 20 stored procedures can be executed per TABLE statement. ID is not required when executing a procedure once.
`PARAMS <param spec> | NOPARAMS`	Defines whether or not the procedure accepts parameters. Either PARAMS <param spec> or NOPARAMS must be used. <param spec> defines input parameters and the source of the input.

`<option>`	Represents one of the following options that can be used alone or in conjunction with other options to control the effects of the stored procedure.
	`AFTERFILTER | BEFOREFILTER`
	`ALLPARAMS`
	`DBOP`
	`ERROR`
	`EXEC`
	`MAXVARCHARLEN`
	`PARAMBUFSIZE`
	`TRACE`

Descriptions of SQLEXEC components begin alphabetically on page 365.

Using SQLEXEC with queries

To execute a query from within a TABLE statement, use the ID and QUERY clauses.

Syntax

```
SQLEXEC (
ID <logical name>
, QUERY “<sql query>”
{, PARAMS <param spec>| NOPARAMS}
[, <option>] [, ...]
)
```

Component	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.

Component	Description
`QUERY "<sql query>"`	Specifies the SQL query syntax to execute against the database. The query must be valid, standard query language for the database against which it is being executed. The query can either return results with a SELECT statement or execute an INSERT, UPDATE, or DELETE statement. For any query that produces output with a SELECT statement, only the first row returned by the SELECT is processed. Do not specify an "INTO ..." clause for any SELECT statements. The query must be all on one line and within quotes. To use quoted object names within a SQLEXEC query, the SQL query must be enclosed within single quotes, rather than double quotes, and the USEANSISQLQUOTES parameter must be used in the GLOBALS file to enforce SQL-92 rules for object and literal identifiers. The following is an example of using quoted object names in a query: `SQLEXEC 'SELECT "col1" from "schema"."table"'` For best results, type a space after each begin quote and before each end quote.

`PARAMS`
`<param spec> |`
`NOPARAMS`
	Defines whether or not the query accepts input parameters. One of these options must be used. <param spec> defines input parameters and the source of the input.

`<option>`	Represents one of the following options that you can use alone or in conjunction with other options to control the effects of the query.
	`AFTERFILTER | BEFOREFILTER`
	`ALLPARAMS`
	`DBOP`
	`ERROR`
	`EXEC`
	`MAXVARCHARLEN`
	`PARAMBUFSIZE`
	`TRACE`

Descriptions of SQLEXEC components begin alphabetically on page 365.

Using placeholders for input parameters

Most queries require placeholders for input parameters. How parameters are specified within the query depends on the database type.

- For Oracle, input parameters are specified by using a colon (:) followed by the parameter name, as in the following example.

  ```
  "SELECT NAME FROM ACCOUNT WHERE SSN = :SSN AND ACCOUNT = :ACCT"
  ```

- For other databases, input parameters are specified by using a question mark, as in the following example.

  ```
  "SELECT NAME FROM ACCOUNT WHERE SSN = ? AND ACCOUNT = ?"
  ```

Note that quotation marks are not required around the parameter name for any database.

Passing parameter values

Oracle GoldenGate provides options for passing input and output values to and from a procedure or query.

- To pass data values to input parameters within a stored procedure or query, use the PARAMS option of SQLEXEC (see page 370).
- To pass values from a stored procedure or query as input to a FILTER or COLMAP clause, use the following syntax:

  ```
  {<procedure name> | <logical name>}.<parameter>
  ```

  Where:

  - <procedure name> is the actual name of a stored procedure, which must match the value given for SPNAME in the SQLEXEC statement. Use this argument only if executing a procedure one time during the course of an Oracle GoldenGate run.
  - <logical name> is the logical name specified with the ID option of SQLEXEC. Use this argument to pass values from either a query or an instance of a stored procedure when the procedure executes multiple times within a TABLE statement.
  - <parameter> is either the name of the parameter, such as a column in a lookup table, or `RETURN_VALUE` if extracting returned values.

As an alternative to the preceding syntax, you can use the @GETVAL function. For more information, see page 456.

There are different constructs for naming input parameters, as follows:

- Oracle permits naming an input parameter any logical name, for example:

  ```
  SQLEXEC (ID appphone, QUERY " select per_type from ps_personal_data "
     " where emplid = :vemplid "
     " and per_status = 'N' and per_type = 'A' ",
     PARAMS (vemplid = emplid)),
  TOKENS (applid = @GETVAL(appphone.per_type));
  ```

- Other databases require the input parameters to be named p1, p2, and so forth, increasing the number for each input parameter. Consider whether the database requires the “p” to be upper or lower case. The following is an example of this type of input parameter:

  ```
  SQLEXEC (ID appphone, QUERY " select per_type from ps_personal_data "
     " where emplid = ? "
     " and per_status = 'N' and per_type = 'A' ",
     PARAMS (p1 = emplid)),
  TOKENS (applid = @GETVAL(appphone.per_type));
  ```

The following shows a set of Oracle source and target tables, a lookup table, and examples of how parameters for these tables are passed for a single instance of a stored procedure and multiple instances of a stored procedure.

Source table "cust":

```
custid                     Number
current_residence_state    Char(2)
birth_state                Char(2)
```

Target table "cust_extended":

```
custid                          Number
current_residence_state_long    Varchar(30)
birth_state_long                Varchar(30)
```

Lookup table "state_lookup":

```
abbreviation       Char(2)
long_name          Varchar(30)
```

Example 1 The following example shows the use of a stored procedure that executes once to get a value from the lookup table. The value is mapped to the target column in the COLMAP statement.

```
TABLE sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, &
PARAMS (long_name = birth_state)), &
COLMAP (custid = custid, &
birth_state_long = lookup.long_name);
```

Example 2 The following example shows multiple executions of a stored procedure that gets values from a lookup table. The values are mapped to target columns.

```
TABLE sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, ID lookup1, &
PARAMS (long_name = current_residence_state)), &
SQLEXEC (SPNAME lookup, ID lookup2, &
PARAMS (long_name = birth_state)), &
COLMAP (custid = custid, &
current_residence_state_long = lookup1.long_name, &
birth_state_long = lookup2.long_name);
```

Using AFTERFILTER and BEFOREFILTER

Use AFTERFILTER and BEFOREFILTER to specify when to execute the stored procedure or query in relation to the FILTER clause of a TABLE statement.

Syntax `AFTERFILTER | BEFOREFILTER`

Rule	Description
AFTERFILTER	Causes the SQL to execute after the FILTER statement. This enables you to skip the overhead of executing the SQL unless the filter is successful. This is the default.

Rule	Description
BEFOREFILTER	Causes the SQL to execute before the FILTER statement. This enables you to use the results of the procedure or query in the filter.

Example

```
SQLEXEC (SPNAME check, NOPARAMS, BEFOREFILTER)
```

Using ALLPARAMS

Use ALLPARAMS as a global rule that determines whether or not all of the specified parameters must be present for the stored procedure or query to execute. Rules for individual parameters established within the PARAMS clause override the global rule set with ALLPARAMS.

Syntax

```
ALLPARAMS {OPTIONAL | REQUIRED}
```

Rule	Description
OPTIONAL	Permits the SQL to execute whether or not all of the parameters are present. This is the default.
REQUIRED	Requires all of the parameters to be present for the SQL to execute.

Example

```
SQLEXEC (SPNAME lookup,
PARAMS (long_name = birth_state, short_name = state),
ALLPARAMS OPTIONAL)
```

Using DBOP

Use DBOP to commit INSERT, UPDATE, DELETE, and SELECT statements executed within the stored procedure or query. Otherwise, they could potentially be rolled back. Oracle GoldenGate issues the commit within the same transaction boundaries as the source transaction.

WARNING Use caution when executing SQLEXEC procedures against the database, especially against the production database. Any changes that are committed by the procedure can result in overwriting existing data.

Syntax

```
DBOP
```

Example

```
SQLEXEC (SPNAME check, NOPARAMS, DBOP)
```

Using ERROR

Use ERROR to define a response to errors associated with the stored procedure or query. Without explicit error handling, the Oracle GoldenGate process abends on errors. Make certain your procedures return errors to the process and specify the responses with ERROR.

Syntax

```
ERROR <action>
```

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 “column missing” conditions. This is the default.
REPORT	Ensures that all errors associated with the stored procedure or query are reported to the discard file. A discard file must be specified with the DISCARDFILE parameter. 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. Oracle GoldenGate continues processing other stored procedures or queries associated with the current TABLE statement before processing the error.
FINAL	Is similar to RAISE except that when an error associated with a procedure or query is encountered, 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.

Example

```
SQLEXEC (SPNAME check, NOPARAMS, ERROR REPORT)
```

Using EXEC

Use EXEC to control the frequency with which a stored procedure or query in a TABLE statement executes and how long the results are considered valid, if extracting output parameters.

Syntax

```
EXEC <frequency>
```

Frequency	Description
MAP	Executes the procedure or query once for each source-target table map for which it is specified. Using MAP renders the results invalid for any subsequent maps that have the same source table. For example, if a source table is being synchronized with more than one target table, the results would be valid only for the first source-target map. MAP is the default.
ONCE	Executes the procedure or query once during the course of an Oracle GoldenGate run, upon the first invocation of the associated TABLE statement. The results remain valid for as long as the process remains running.
TRANSACTION	Executes the procedure or query once per source transaction. The results remain valid for all operations of the transaction.

Frequency	Description
SOURCEROW	Executes the procedure or query once per source row operation. Use this option when you are synchronizing a source table with more than one target table, so that the results of the procedure or query are invoked for each source-target mapping.

Example 1 The following is an example of using ONCE.

```
TABLE sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, &
PARAMS (long_name = birth_state), &
EXEC ONCE), &
COLMAP (custid = custid,
birth_state_long = lookup.long_name);
```

Example 2 The following is an example of using TRANSACTION.

```
TABLE sales.cust, TARGET sales.cust_extended, &
SQLEXEC (SPNAME lookup, PARAMS (long_name = birth_state), EXEC TRANSACTION), &
COLMAP (custid = custid, &
birth_state_long = lookup.long_name);
```

Example 3 The following is an example of using the default (MAP) incorrectly. The two TABLE statements synchronize the same source table with two different target tables. However, the results of the procedure lookup will be expired by the time the second map executes, so the second map will result in a “column missing” condition. To implement this correctly, SOURCEROW should be used.

```
TABLE sales.srctab, TARGET sales.targtab, &
SQLEXEC (SPNAME lookup, PARAMS (param1 = srccol)), &
COLMAP (targcol = lookup.param2); &
TABLE sales.srctab, TARGET sales.targtab2, &
COLMAP (targcol2 = lookup.param2);
```

Example 4 The following is an example of using SOURCEROW. The second map returns a valid value because the procedure executes on every source row operation.

```
TABLE sales.srctab, TARGET sales.targtab, &
SQLEXEC (SPNAME lookup, &
PARAMS (param1 = srccol), EXEC SOURCEROW ), &
COLMAP (targcol = lookup.param2);

TABLE sales.srctab, TARGET sales.targtab2, &
COLMAP (targcol2 = lookup.param2);
```

Using ID

Use ID for queries and stored procedures within a TABLE statement as follows.

- For a query, use ID <logical name> so that a name can be used by Oracle GoldenGate to reference the column values returned by the query.
- For a stored procedure, use ID <logical name> to invoke the procedure multiple times within a TABLE statement, for example for two different column maps. Otherwise, it is not required. Up to 20 stored procedures can be executed per TABLE statement. They execute in the order listed in the parameter file.

Syntax `ID <logical name>`

Component	Description
`<logical name>`	A logical name for the procedure or query. For example, logical names for a procedure named "lookup" might be "lookup1," "lookup2," and so forth.

Example 1 The following example illustrates the use of ID <logical name>. It enables each column map to call a stored procedure named lookup separately and refer to its own results by means of lookup1 and lookup2.

```
TABLE sales.srctab, TARGET sales.targtab, &
SQLEXEC (SPNAME lookup, ID lookup1, PARAMS (param1 = srccol)), &
COLMAP (targcol1 = lookup1.param2), &
SQLEXEC (SPNAME lookup, ID lookup2, PARAMS (param1 = srccol)), &
COLMAP, (targcol2 = lookup2.param2);
```

Example 2 The following example shows a single execution of a stored procedure named lookup. In this case, the actual name of the procedure is used. A logical name is not needed.

```
TABLE sales.tab1, TARGET sales.tab2, &
SQLEXEC (SPNAME lookup), PARAMS (param1 = srccol)), &
COLMAP (targcol = lookup.param1);
```

Example 3 The following examples illustrate the use of ID <logical name> for Oracle and SQL Server queries, respectively. Note that in this illustration, the SQLEXEC statement spans multiple lines due to space constraints in this documentation. An actual SQLEXEC statement must be contained on one line only.

```
TABLE sales.account, TARGET sales.newacct,
SQLEXEC (ID lookup,
QUERY "select desc_col into desc_param from lookup_table
where code_col = :code_param",
PARAMS (code_param = account_code)),
COLMAP (newacct_id = account_id,
newacct_val = lookup.desc_param);

TABLE sales.account, TARGET sales.newacct,
SQLEXEC (ID lookup,
QUERY "select desc_col into desc_param from lookup_table
where code_col = ?",
PARAMS (p1 = account_code)),
COLMAP (newacct_id = account_id,
newacct_val = lookup.desc_param);
```

Using MAXVARCHARLEN

Use MAXVARCHARLEN to specify the maximum length allocated for any output parameter in a procedure or query. Beyond that, output values are truncated.

Syntax `MAXVARCHARLEN <num bytes>`

Component	Description
`(<num bytes>)`	Defines the maximum number of bytes allowed for an output parameter. The default is 255 bytes without an explicit MAXVARCHARLEN clause.

Example

```
MAXVARCHARLEN 100
```

Using NOPARAMS

Use NOPARAMS instead of PARAMS if a stored procedure or query does not require parameters. Either a PARAMS clause or NOPARAMS is required.

Syntax

```
NOPARAMS
```

Example

```
SQLEXEC (SPNAME check, NOPARAMS)
```

Using PARAMBUFSIZE

Use PARAMBUFSIZE to specify the maximum size of the memory buffer that stores parameter information, including both input and output parameters. Oracle GoldenGate issues a warning whenever the memory allocated for parameters is within 500 bytes of the maximum.

Syntax

```
PARAMBUFSIZE <num bytes>
```

Component	Description
`<num bytes>`	Defines the maximum number of bytes allowed for the memory buffer. The default is 10,000 bytes without an explicit PARAMBUFSIZE clause.

Example

```
PARAMBUFSIZE 15000
```

Using PARAMS

Use PARAMS to supply the names of parameters in a stored procedure or query that accept input and the name of a source column or Oracle GoldenGate column-conversion function that is supplying the input. Either a PARAMS clause or NOPARAMS is required.

By default, Oracle GoldenGate treats a string that is enclosed within double quotes as a literal. To use double quotes for column names and single quotes for literals (SQL-92 rules), use the USEANSISQLQUOTES parameter in the GLOBALS parameter file.

The following are the databases that are supported by SQLEXEC and the data types that are supported for input and output parameters.

- Numeric data types
- Date data types
- Character data types

By default, output parameters are truncated at 255 bytes per parameter. If a procedure requires longer parameters, use the MAXVARCHARLEN option.

Syntax

```
PARAMS (
[OPTIONAL | REQUIRED]
<param name> = {<source column> | <GG function>}
[, ...]
)
```

Component	Description
`OPTIONAL | REQUIRED`	Determines whether or not the procedure or query executes when parameter values are missing. ◆ 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. OPTIONAL is the default for all databases except Oracle. For Oracle, whether or not a parameter is optional is automatically determined when retrieving the stored procedure definition. ◆ REQUIRED indicates that a parameter value must be present. If the parameter value is not present, the SQL will not be executed.

`<param name> = {`
`<source column> |`
`<GG function`
`}`
	Maps the parameter name to the column or function that provides the input.
	Where:
	◆ <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 input parameters, the <param name> entries are P1 and P2.
	◆ <source column> is the name of a source column. By default, if the specified column is not present in the log (because it is a compressed update) the parameter assumes any default value specified by the procedure or query for the parameter.
	◆ <GG function> is the name of an Oracle GoldenGate column-conversion function. For more information about column-conversion functions, see Chapter 4.

Example The following example maps data from the account table to the target table newacct. When processing records from the account table, Oracle GoldenGate executes the lookup stored procedure before executing the column map. The code_param parameter in the procedure accepts input from the account_code source column.

```
TABLE sales.account, TARGET sales.newacct, &
SQLEXEC (SPNAME lookup, PARAMS (code_param = account_code)), &
COLMAP (newacct_id = account_id, &
newacct_val = lookup.desc_param);
```

Using TRACE

Use TRACE to log input and output parameters to the report file.

Sample discard file with SQLEXEC tracing enabled:

```
Input parameter values...

LMS_TABLE: INTERACTION_ATTR_VALUES
  KEY1: 2818249
  KEY2: 1
Report File:

From Table MASTER.INTERACTION_ATTR_VALUES to
MASTER.INTERACTION_ATTR_VALUES:
       #  inserts:      0
       #  updates:      0
       #  deletes:      0
       #  discards:     1

  Stored procedure GGS_INTERACTION_ATTR_VALUES:
       attempts:        2
       successful:      0
```

Syntax

```
TRACE {ALL | ERROR}
```

Action	Description
ALL	Writes the input and output parameters for each invocation of the procedure or query to the report file. This is the default.
ERROR	Writes the input and output parameters for each invocation of the procedure or query to the report file only after a SQL error occurs.

Example

```
SQLEXEC (SPNAME lookup, PARAMS (long_name = birth_state, short_name = state),
TRACE ERROR)
```

Using SQLPREDICATE

Use SQLPREDICATE to include a conventional SQL WHERE clause in the SELECT statement that Extract uses when selecting data from a table in preparation for an initial load. SQLPREDICATE forces the records returned by the selection to be ordered by the key values.

SQLPREDICATE is a better selection method for initial loads than the WHERE or FILTER options. It is much faster because it affects the SQL statement directly and does not require Oracle GoldenGate to fetch all records before filtering them, like those other options do.

For Oracle tables, using SQLPREDICATE reduces the amount of data that is stored in the undo segment, which can reduce the incidence of snapshot-too-old errors. This is useful when loading very large tables.

By using a SQLPREDICATE clause, you can partition the rows of a large table among two or more parallel Extract processes. This configuration enables you to take advantage of parallel Delivery load processing as well.

Another use for SQLPREDICATE would be to select data based on a timestamp or some other criteria simply to restrict which rows are extracted and loaded to the target table. SQLPREDICATE can also be used for ORDER BY clauses or any other type of selection clause.

Columns specified as part of the WHERE clause should be part of a key or index for best performance. Otherwise, a full table scan will be required, which will reduce the efficiency of the SELECT statement.

This parameter is valid for Oracle, DB2 LUW and z/OS, SQL Server, and Teradata databases. It should not be used with change data synchronization, but only with an initial load process, because it assumes the use of a SELECT statement that selects records directly from tables. Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

Syntax

```
TABLE <table spec>, SQLPREDICATE “WHERE <where clause>”;
```

Component	Description
`WHERE`	This is a required keyword.
`<where clause>`	A valid SQL WHERE clause.

Example

```
SQLPREDICATE "where state = 'CO' and city = 'DENVER’”
```

Using TARGETDEF

Use TARGETDEF to specify a target-definitions template. The definitions template is created based on the definitions of a specific target table when DEFGEN is run for that table. Once the template is created, new target tables that have identical definitions to that table can be added without having to run DEFGEN for them, and without having to stop and start Extract. The definitions in the template specified with TARGETDEF will be used for definitions lookups. For more information about DEFGEN, see the Oracle GoldenGate *Windows and UNIX Administrator’s Guide.*

Syntax

```
TABLE <table spec>, TARGETDEF <definitions template>;
```

Argument	Description
`<definitions template>`	The name of a definitions template that was specified with the DEF option of TABLE in the DEFGEN parameter file. The definitions contained in the template must be identical to the definitions of the table in this TABLE statement.

Example

```
TABLE acct.cust*, TARGET acc.cust*, DEF custdef, TARGETDEF tcustdef;
```

Using TOKENS

Use TOKENS to define a user token and associate it with data. Tokens enable you to 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 data. For example, you can use token data in column maps, stored procedures called by SQLEXEC, or macros.

To use the defined token data in target tables, use the @TOKEN column-conversion function in the COLMAP clause of a Replicat MAP statement. The @TOKEN function maps the name of a token to a target column.

Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

The character set of token data is not converted. The token must be in the character set of the source database for Extract and in the character set of the target database for Replicat.

Do not use this option for source tables that are encoded as EBCDIC on a z/OS system if the target tables are not EBCDIC.

For more information about using tokens, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

```
TABLE <table spec>, TOKENS (<token name> = <token data> [, ...]) ;
```

Component	Description
`<token name>`	A name of your choice for the token. It can be any number of valid characters and is not case-sensitive. Multi-byte names are not supported.
`<token data>`	Any valid character string of up to 2000 bytes. The data can be either a literal that is enclosed within double quotes (or single quotes if USEANSISQLQUOTES is in use) or the result of an Oracle GoldenGate column-conversion function. For more information about USEANSISQLQUOTES, see page 415.

Example The following creates tokens named TK-OSUSER, TK-GROUP, and TK-HOST and maps them to token data obtained with the @GETENV function.

```
TABLE ora.oratest, TOKENS (
TK-OSUSER = @GETENV ("GGENVIRONMENT" , "OSUSERNAME"),
TK-GROUP = @GETENV ("GGENVIRONMENT" , "GROUPNAME")
TK-HOST =  @GETENV ("GGENVIRONMENT" , "HOSTNAME"));
```

Using TRIMSPACES and NOTRIMSPACES

Use TRIMSPACES and NOTRIMSPACES to control whether or not trailing spaces in a source CHAR column are truncated when applied to a target CHAR or VARCHAR column. The default is TRIMSPACES.

NOTE Sybase treats all CHAR types as VARCHAR types, and therefore TRIMSPACES will have no effect. For Sybase, use the TRIMVARSPACES parameter.

TRIMSPACES is applied only to single-byte white spaces (U+0020). Ideographic spaces (U+3000) are not supported.

TRIMSPACES and NOTRIMSPACES also can be used at the root level of a parameter file to turn the trim feature on or off for different TABLE statements or groups of statements.

For Extract, TRIMSPACES only has an effect if Extract is performing mapping within the TABLE statement (by means of a TARGET statement).

Syntax

```
TABLE <table spec>, {TRIMSPACES | NOTRIMSPACES};
```

Example The following keeps the default of trimming trailing spaces for the first two tables, but prevents Extract from trimming spaces for the last two.

```
TABLE fin.src1;
TABLE fin.src2;
TABLE fin.src3, NOTRIMSPACES;
TABLE fin.src4, NOTRIMSPACES;
```

Using TRIMVARSPACES and NOTRIMVARSPACES

Use TRIMVARSPACES and NOTRIMVARSPACES to control whether or not trailing spaces in a source VARCHAR column are truncated when applied to a target CHAR or VARCHAR column.

The default is NOTRIMVARSPACES because spaces in a VARCHAR column could actually be part of the data. Before using TRIMVARSPACES, make certain that trailing spaces are not integral to the target data.

TRIMSPACES is applied only to single-byte white spaces (U+0020). Ideographic spaces (U+3000) are not supported.

TRIMVARSPACES and NOTRIMVARSPACES also can be used at the root level of a parameter file to turn the trim feature on or off for different TABLE statements or groups of statements.

For Extract, TRIMVARSPACES only has an effect if Extract is performing mapping within the TABLE statement (by means of a TARGET statement).

Syntax

```
TABLE <table spec>, {TRIMVARSPACES | NOTRIMVARSPACES};
```

Example The following keeps the default of not trimming trailing spaces for the first two tables, but trims spaces for the last two.

```
TABLE fin.src1;
TABLE fin.src2;
TABLE fin.src3, TRIMVARSPACES;
TABLE fin.src4, TRIMVARSPACES;
```

Using WHERE

Use WHERE to select records based on a conditional statement. Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

Oracle GoldenGate does not support WHERE for columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.

Literal strings used in WHERE must be enclosed within double quotes unless the USEANSISQLQUOTES parameter is used in the GLOBALS file. This parameter enforces SQL-92 rules for identifiers and literals.

For more information about using a WHERE clause and the column data that can be used, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

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

Component	Description
`<where clause>`	Selects records based on a condition, such as: `WHERE (branch = "NY")` The following table shows permissible WHERE operators. WHERE does not support evaluating the before image of a primary key column in the conditional statement as part of a primary key update operation.

Table 41 Permissible WHERE operators

Operator	Example
Column names	`PRODUCT_AMT`
Numeric values	`-123, 5500.123`
Literal strings enclosed in quotes	`"AUTO", "Ca"`
Column tests	`@NULL`, `@PRESENT`, `@ABSENT` (column is null, present or absent in the record). These tests are built into Oracle GoldenGate.
Comparison operators	`=, <>, >, <, >=, <=`
Conjunctive operators	`AND, OR`
Grouping parentheses	Use open and close parentheses for logical grouping of multiple elements.

Example The following WHERE 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)
```

TABLE for Replicat

Use the TABLE parameter in a Replicat parameter file to specify filtering rules that qualify a data record from the trail to be eligible for an event action that is specified with EVENTACTIONS.

WARNING EVENTACTIONS is not supported if the source database is Teradata and Extract is configured in maximum performance mode.

This form of TABLE statement is similar to that of the Replicat MAP statement, except that there is no mapping of the source table in the data record to a target table by means of a

TARGET clause. TABLE for Replicat is solely a means of triggering a non-data action to be taken by Replicat when it encounters an event record.

Because a target table is not supplied, the following apply:

- No options are available to enable Replicat to map table names or columns to a target table, nor are there options to enable Replicat to manipulate data.
- The ASSUMETARGETDEFS parameter cannot be used in the same parameter file as a Replicat TABLE statement, because ASSUMETARGETDEFS requires the names of target tables in order for Replicat to query for table definitions. You must create a source definitions file to provide the definitions of the source tables to Replicat. Transfer this file to the target system and use the SOURCEDEFS parameter in the Replicat parameter file to specify the path name of the file.
- The event record itself is not applied to the target database by Replicat. You must specify either IGNORE or DISCARD as one of the EVENTACTIONS options.

Terminate the TABLE statement with a semi-colon.

See MAP for Replicat for:

- Supported characters in table names
- Descriptions of EVENTACTIONS syntax options

Syntax

```
TABLE <table spec>,
[, SQLEXEC (<SQL specification>), BEFOREFILTER]
[, FILTER (<filter specification>)]
[, WHERE (<where clause>)]
{, EVENTACTIONS ({IGNORE | DISCARD} [<action>])}
;
```

Example The following example enables Replicat tracing for an order transaction that contains an insert operation for a specific order number (order_no = 1). The trace information is written to the order_1.trc trace file. The MAP parameter specifies the mapping of the source table to the target table.

```
MAP sales.order, TARGET rpt.order;

TABLE sales.order,
FILTER (@GETENV ("GGHEADER", "OPTYPE") = "INSERT" AND order_no = 1), &
EVENTACTIONS (TRACE order_1.trc TRANSACTION);
```

For additional use cases and more information about the event marker system, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

TABLEEXCLUDE

Valid for Extract

Use the TABLEEXCLUDE parameter with the TABLE and SEQUENCE parameters to explicitly exclude tables and sequences from a wildcard specification. TABLEEXCLUDE must precede all TABLE and SEQUENCE statements that contain the objects that you want to exclude.

Default None

Syntax `TABLEEXCLUDE <exclude specification>`

Argument	Description
`<exclude specification>`	The name or wildcard specification of the object to exclude. The same wildcard rules apply for TABLEEXCLUDE as for specifying objects with wildcards in a TABLE or SEQUENCE statement.

Example In the following example, the TABLE statement captures all tables except for the table named TEST.

```
TABLEEXCLUDE fin.TEST
TABLE fin.*;
```

TARGETDB

Valid for Replicat

Use the TARGETDB parameter for databases that require a data source name as part of the connection information. Target tables in MAP statements subsequent to a TARGETDB entry are assumed to be from the specified database.

This parameter may need to be used with the USERID parameter, depending on the authentication that is required to log into the database, as follows:

- For databases that require a database login, TARGETDB (if required) must be used with the USERID parameter within the same parameter statement.
- For databases that allow authentication at the operating-system level, you can specify TARGETDB without USERID.

Default None

Syntax `TARGETDB <data source>[, SESSIONCHARSET <character set>]`

Argument	Description
`<data source>`	The name of the data source.
`SESSIONCHARSET <character set>`	Supports Sybase, Teradata and MySQL. Sets the database session character set for the process login session. This parameter overrides any SESSIONCHARSET that is specified in the GLOBALS file.

Example 1 This example shows TARGETDB with and without the USERID parameter.

```
TARGETDB mydb
```

```
TARGETDB mydb, USERID ggs, &
   PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1
```

Example 2 This example sets a character set for the user session.

```
TARGETDB mydb, USERID ggs, &
   PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
   AES128, ENCRYPTKEY securekey1, SESSIONCHARSET ISO-8859-11
```

TARGETDEFS

Valid for Extract (primary and data pump)

Use the TARGETDEFS parameter when the target is an Enscribe file. TARGETDEFS names a file on the source system or on an intermediary system that contains data definitions of tables and files that exist on the target system. Specify at least one TARGETDEFS entry before TABLE statements for which the targets are Enscribe files.

To generate the target-definitions file, use the DEFGEN utility. Transfer the file to the source or intermediary system before starting Extract.

You can have multiple TARGETDEFS statements in the parameter file if more than one target-definitions file is needed for different definitions, for example if each TARGETDEFS file holds the definitions for a distinct application.

Default None

Syntax `TARGETDEFS <file name>`

Argument	Description
`<file name>`	The relative or fully qualified path name of the file from which to obtain the data definitions.

Example 1 `TARGETDEFS C:\repodbc\sales.def`

Example 2 `TARGETDEFS /ggs/dirdef/ODBC/tandem_defs`

TCPSOURCETIMER | NOTCPSOURCETIMER

Valid for Extract

Use the TCPSOURCETIMER and NOTCPSOURCETIMER parameters to manage the timestamps of replicated operations for reporting purposes within the Oracle GoldenGate environment.

TCPSOURCETIMER is the default. It adjusts the timestamp of data records when they are sent to other systems, making it easier to interpret synchronization lag.

NOTCPSOURCETIMER retains the original timestamp value. Use NOTCPSOURCETIMER when using timestamp-based conflict resolution in a bidirectional configuration. Use NOTCPSOURCETIMER when using a user token that refers to “GGHEADER”, “COMMITTIMESTAMP” of the @GETENV column-conversion function.

TCPSOURCETIMER and NOTCPSOURCETIMER are global parameters and apply to all TABLE statements in the Extract parameter file.

Default `TCPSOURCETIMER`

Syntax `TCPSOURCETIMER | NOTCPSOURCETIMER`

THREADOPTIONS

Valid for Extract

Use the THREADOPTIONS parameter to control certain aspects of the way that a threaded Extract operates.

Using performance options

Oracle GoldenGate queues data in memory before sending it to the trail. The INQUEUESIZE and OUTQUEUESIZE options of the THREADOPTIONS parameter determine how much data to queue. The higher the values, the better the performance for large amounts of data. Lower values will move data to the target more quickly in environments with very little activity. Start out with the default values. Typical values range from 100 to 1500. In most environments 1000 for each option should be sufficient.

In addition to these two parameters, AIX users might obtain better performance by setting the environment variable AIXTHREAD_SCOPE to S (system scope) which specifies the use of multiple CPUs so that processes can run concurrently. To use system scope, add the following to the .profile file of the user who starts the Manager process or else export the variable manually before starting GGSCI.

```
AIXTHREAD_SCOPE=S
export AIXTHREAD_SCOPE
```

Stop and restart GGSCI, Manager, and Extract for the change to take effect.

Default None

Syntax

```
THREADOPTIONS
[EOFDELAYMS <milliseconds>]
[IOLATENCY <milliseconds>]
[INQUEUESIZE <n>]
[MAXCOMMITPROPAGATIONDELAY <milliseconds>]
[OUTQUEUESIZE <n>]
```

Argument	Description
EOFDELAYMS <milliseconds>	Specifies how long the Extract process delays once it reaches the logical end of a redo log before searching for more data to process. The default is 250 milliseconds. Increasing the value could increase the lag between the time that changes are made to source tables and when they are applied to the target tables.
INQUEUESIZE <n>	Specifies the number of queue entries in the input queue of each producer Extract thread in an Oracle RAC cluster. Valid values are 16 to 65535. The default is 128. The default should be adequate in most cases.

Argument	Description
`IOLATENCY <milliseconds>`	Specifies the amount of time between the database-configured max commit propagation delay and the internal value used by Oracle GoldenGate. Valid values are between 0 and 180000 milliseconds (3 minutes). The default is 1500 (1.5 seconds) to account for internal I/O latency. Not valid when Extract is in Archived Log Only mode (ALO).
`MAXCOMMITPROPAGATIONDELAY <milliseconds>`	This option is valid for Oracle versions earlier than Oracle 11.2. The corresponding Oracle MAX_COMMIT_PROPAGATION_DELAY parameter was made obsolete in Oracle 11.2. MAX_COMMIT_PROPAGATION_DELAY specifies the amount of time to delay between the time a transaction was committed and the time an idle redo log was read. The specified delay allows for the difference in time between when Oracle writes data to the redo log and when Oracle GoldenGate reads that data from the log. This time difference can be significant if there is contention for access to the shared RAC database drive(s). If the database is on SAN and NFS devices that have caching and serialization features, which tend to minimize or eliminate these delays, lower values may be appropriate. The value must be greater than 0 and less than 90000 milliseconds (90 seconds). The default is three seconds. The value should always be a minimum of 2000 milliseconds above the value for the Oracle parameter of the same name, and should never be set lower than the Oracle value. To check Oracle's value, connect as a user with DBA privileges and issue the following command in SQL*Plus: `show parameter max_commit` Not valid when Extract is in Archived Log Only mode (ALO).
`OUTQUEUESIZE <n>`	Specifies the number of queue entries in the output queue of an Extract producer thread. Valid values are 8 to 65535. The default is 2048. The default should be adequate in most cases.

TRACE | TRACE2

Valid for Extract and Replicat

Use the TRACE and TRACE2 parameters to capture Extract or Replicat processing information to help reveal processing bottlenecks.

- TRACE provides step-by-step processing information.
- TRACE2 identifies the code segments on which Extract or Replicat is spending the most time.

Both support the tracing of DML and DDL.

Tracing also can be turned on and off by using the SEND EXTRACT or SEND REPLICAT command in GGSCI. For more information about GGSCI commands, see Chapter 1.

Contact Oracle Support for assistance if the trace reveals significant processing bottlenecks. For more information, go to http://support.oracle.com.

Default No tracing

Syntax

```
TRACE | TRACE2
[, DDL[INCLUDE] | DDLONLY]
[, [FILE] <file name>]
```

Argument	Description
`DDL[INCLUDE] | DDLONLY`	(Replicat only) Enables DDL tracing and specifies how DDL tracing is included in the trace report. ◆ DDL[INCLUDE] traces DDL and also traces transactional data processing. Either DDL or DDLINCLUDE is valid. ◆ DDLONLY traces DDL but does not trace transactional data.

`[FILE]`
`<file name>`
	The relative or fully qualified name of a file to which Oracle GoldenGate logs the trace information. The FILE keyword is optional, but must be used if other parameter options will follow the file name, for example:
	`TRACE FILE <file name> DDLINCLUDE`
	If no other options will follow the file name, the FILE keyword can be omitted, for example:
	`TRACE DDLINCLUDE <file name>`

Example

```
TRACE /home/ggs/dirrpt/trace.trc
```

TRACETABLE | NOTRACETABLE

Valid for Extract and Replicat

Use the TRACETABLE and NOTRACETABLE parameters with Oracle databases to identify a trace table that was created with the ADD TRACETABLE command. TRACETABLE is required only if the trace table was created with a name other than the default of GGS_TRACE. If a trace table named GGS_TRACE exists in the database, trace table functionality is enabled automatically, and TRACETABLE is not required.

The trace table is used for bidirectional synchronization to identify Replicat transactions to Extract.

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.

NOTRACETABLE prevents Replicat from writing an operation to the trace table, thus preventing Extract from recognizing Replicat transactions.

To control whether Replicat transactions are extracted by Extract or ignored, use the GETREPLICATES and IGNOREREPLICATES parameters. See page 212.

For instructions on configuring bidirectional synchronization, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Default GGS_TRACE

Syntax `TRACETABLE [<owner>.]<table name> | NOTRACETABLE`

Argument	Description
`[<owner>.]<table name>`	The owner (optional) and name of the trace table. If an owner is not specified, it is assumed to be the schema of the user specified with the USERID parameter.

Example `TRACETABLE ggs.excl_trans`

TRAILCHARSET

Valid for Replicat

Use the TRAILCHARSET parameter for the following purposes:

- To override the source character set that is recorded in the trail header. This applies to trails that are written by Extract version 11.2.1.0.0 or later, where the source character set is stored in the trail header.
- To supply a character set for the source data if the trail is written by an Extract version that is earlier than 11.2.1.0.0. In the earlier versions, the source character set is not stored in the trail.

When TRAILCHARSET is used, Replicat uses the specified character set as the source character set when converting character-type columns to the target character set. Replicat issues a warning message when it uses the TRAILCHARSET character set.

By default, Replicat performs character set conversion. This feature is controlled by the CHARSETCONVERSION (default) and NOCHARSETCONVERSION parameters. To use TRAILCHARSET, NOCHARSETCONVERSION cannot be used.

Default Character set of the operating system

Syntax `TRAILCHARSET <source_charset> [, REPLACEBADCHAR];`

Argument	Description
`<source_charset>`	The ICU character-set identifier or an Oracle character-set identifier of the source database. For Oracle databases, Oracle GoldenGate converts an Oracle identifier to the corresponding ICU identifier for conversion to the character set that is specified with the NLS_LANG specification in the SETENV parameter in the Replicat parameter file.
`REPLACEBADCHAR`	Prevents Replicat from abending when a conversion attempt fails. The failed character is replaced with a replacement character for each target character set. The replacement character is predefined in each character set.

Example 1 `TRAILCHARSET ISO-8859-9;`

Example 2 `TRAILCHARSET windows-932, REPLACEBADCHAR;`

Example 3 `TRAILCAHRSET EUC-CN;`

TRAILCHARSETASCII

Valid for Extract for DB2 on z/OS

Use TRAILCHARSETASCII to cause character data to be written to the trail file in the local ASCII codepage of the job in which Extract is running.

- Specification of this parameter on a single-byte DB2 z/OS subsystem causes character data from non-Unicode tables to be written to the trail file in the installed ASCII single-byte CCSID. Data from EBCDIC tables is converted to this ASCII CCSID.
- Specification of this parameter on a multi-byte DB2 z/OS subsystem causes Extract to process only ASCII and Unicode tables. Extract abends with an error if it encounters EBCDIC tables. Data from ASCII tables is written to the trail file in the installed ASCII mixed CCSID.

Either TRAILCHARSETASCII or TRAILCHARSETEBCDIC is required if the target is a multi-byte system. To replicate both ASCII and EBCDIC tables to a multi-byte DB2 z/OS target, process each character set with a different Oracle GoldenGate processing stream: one Extract, data pump, and Replicat for the ASCII tables and another Extract, data pump, and Replicat for the EBCDIC tables.

Default Character data is written in the character set of the host table.

Syntax `TRAILCHARSETASCII`

TRAILCHARSETEBCDIC

Valid for Extract for DB2 on z/OS and DB2 for i

Use TRAILCHARSETEBCDIC to cause character data to be written to the trail file in the local EBCDIC codepage of the job in which Extract is running.

- Specification of this parameter causes all character data to be written to the trail file in the EBCDIC codepage of the job in which Extract is running.
- Specification of this parameter on a single-byte DB2 z/OS subsystem causes character data from non-Unicode tables to be written to the trail file in the installed EBCDIC single-byte CCSID. Data from ASCII tables is converted to this EBCDIC CCSID.
- Specification of this parameter on a multi-byte DB2 z/OS subsystem causes Extract to process only EBCDIC and Unicode tables. Extract abends with an error if it encounters ASCII tables. Data from EBCDIC tables is written to the trail file in the installed EBCDIC mixed CCSID.

Either TRAILCHARSETASCII or TRAILCHARSETEBCDIC is required if the target is a multi-byte system. To replicate both ASCII and EBCDIC tables to a multi-byte DB2 z/OS target, process each character set with a different Oracle GoldenGate processing stream: one Extract, data pump, and Replicat for the ASCII tables and another Extract, data pump, and Replicat for the EBCDIC tables.

Default DB2 for i: Character data is written in UTF-8. DB2 on z/OS: Character data is written in the character set of the host table.

Syntax `TRAILCHARSETEBCDIC`

TRAILCHARSETUTF8

Valid for Extract for DB2 for i

Use TRAILCHARSETUTF8 to cause Extract to write all character (non graphic) data to the trail in UTF-8. Extract performs conversion if needed. Graphic data is written in UTF-16.

Default Enabled

Syntax `TRAILCHARSETUTF8`

TRANLOGOPTIONS

Valid for Extract

Use the TRANLOGOPTIONS parameter to control aspects of the way that Extract interacts with the transaction log or API that passes transaction data, depending on the database or capture mode. You can use multiple TRANLOGOPTIONS statements in the same parameter file, or you can specify multiple options within the same TRANLOGOPTIONS statement, if permissible for those options.

Use a given TRANLOGOPTIONS option only for the database or databases for which it is intended.

Default None

Syntax `TRANLOGOPTIONS {`

```
[ALTARCHIVEDLOGFORMAT <string>] [INSTANCE <instance_name>] [THREADID <id>]
[ALTARCHIVELOGDEST [PRIMARY] [INSTANCE <instance_name>] <path name>]
[ALTARCHIVELOGDEST ("<Backup Path>" [FILESPEC "<File Pattern>"]
   [[NOT] RECURSIVE] [PRIMARY])]
[ALTLOGDEST <path>]
[ARCHIVEDLOGONLY]
{[ASMBUFSIZE <size>] | [DBLOGREADERBUFSIZE <buffer size>]}
[ASMUSER SYS@<ASM_instance>, ASMPASSWORD <password> [<algorithm>
   ENCRYPTKEY {<keyname> | DEFAULT}]]
[ASYNCTRANSPROCESSING <transaction-buffer-size> | NOASYNCTRANSPROCESSING]
[BUFSIZE <size>]
[COMPLETEARCHIVEDLOGONLY]
[DBLOGREADER]
[EXCLUDETRANS <trans name>]
[EXCLUDEUSER <user name>]
[EXCLUDEUSERID <Oracle uid>]
[FILTERTABLE <table_name>]
[FORCEFETCHLOB]
[FETCHLOBIFERROR]
[FETCHPARTIALLOB]
[FETCHPARTIALXML]
[IGNOREDATACAPTURECHANGES | NOIGNOREDATACAPTURECHANGES]
[IGNOREDIRECTLOADINSERTS]
[INCLUDEREGIONID | INCLUDEREGIONIDWITHOFFSET]
[INTEGRATEDPARAMS (<parameter> <value> [, ...])]
[LEGACYLOBREADING]
[LOGRETENTION [ENABLED | SR | DISABLED][LOGSOURCE <platform>, [PATHMAP <path
to logs]]
[MANAGESECONDARYTRUNCATIONPOINT | NOMANAGESECONDARYTRUNCATIONPOINT]
[MAXREADSIZE <records>]
[MAXWARNEOF <seconds>]
[MININGUSER {/ | <user id>}[, MININGPASSWORD <password>]
   [<algorithm> ENCRYPTKEY {<keyname> | DEFAULT}] [SYSDBA]]
[NODDLCHANGEWARNING]
[NOFLUSH]
[PATHMAP <NFS mount point> <log path>]
[PURGEORPHANEDTRANSACTIONS | NOPURGEORPHANEDTRANSACTIONS]
[QUERYRETRYCOUNT <number of retries>] |
[READBUFFER <byte length>]
[READQUEUESIZE <size>]
[READTIMEOUT <milliseconds>]
[REQUIRELONGDATACAPTURECHANGES | NOREQUIRELONGDATACAPTURECHANGES]
[TRANSCLEANUPFREQUENCY <minutes>]
[VAMCOMPATIBILITY {1 | 2}]
}
[, ...]
```

Valid for Extract

Option	Description

`ALTARCHIVEDLOGFORMAT`
`<string>`
`[INSTANCE <instance_name>]`
`[THREADID <id>]`
	(Oracle) Valid for Extract in classic capture mode.
	Specifies a string that overrides the archive log format of the source database. <string> accepts the same specifier as Oracle's parameter LOG_ARCHIVE_FORMAT. Extract uses the supplied format specifier to derive the log file name. Example:
	`arch_%T.arc`
	When using ALTARCHIVEDLOGFORMAT on RAC, use the ALTARCHIVEDLOGFORMAT parameter on each node.
	The TRANLOGOPTIONS statement that includes ALTARCHIVEDLOGFORMAT cannot contain any other TRANLOGOPTIONS options. Use a separate TRANLOGOPTIONS statement to specify other options.
	One of the following conditions must be met to ensure that Extract can differentiate between the log streams:
	◆ You can use the INSTANCE or THREADID option to specify a log name format that contains a thread specifier (either "%t" or "%T").
	◆ Alternatively, you can use a unique log directory for each thread. Use ALTARCHIVELOGDEST for this purpose. (Using a unique directory is recommended, even with a log name format that uses a thread specifier.)
	Note: The default log format that is queried from the database for one thread will still be applied to another thread if Extract cannot find a user-defined log format or no default format is specified for that thread.
	The following options are for use with RAC. Extract will verify the supplied input against the database catalog.
	◆ INSTANCE <instance_name> applies ALTARCHIVEDLOGFORMAT to a specific Oracle instance.
	Example:
	`TRANLOGOPTIONS ALTARCHIVEDLOGFORMAT &`
	`INSTANCE rac1 log_%t_%s_%r.arc`
	◆ THREADID <id> specifies the thread number of the instance that has the specified log format.
	Example:
	`TRANLOGOPTIONS ALTARCHIVEDLOGFORMAT &`
	`THREADID 2 log_%t_%s_%r.arc`

Option	Description

`ALTARCHIVELOGDEST`
`[PRIMARY]`
`[INSTANCE <instance_name>]`
`[THREADID <id>]`
`<path name>`
	(Oracle) Valid for Extract in classic capture mode.
	Points Extract to the archived or backup Oracle transaction logs when they reside somewhere other than the default location. Extract first checks the specified location and then checks the default location.
	◆ `<path name>` specifies the fully qualified path to the archived logs in the alternate directory. This directory must be NFS mounted to the node where Oracle GoldenGate is running. Use that mount point for ALTARCHIVELOGDEST.
	Options:
	◆ PRIMARY prevents Extract from checking the default log location if it does not find the log in the alternate location. Only the ALTARCHIVELOGDEST path is checked. PRIMARY is the default for an Extract that is running in Archived Log Only (ALO) mode; otherwise, it is optional.
	◆ INSTANCE <instance_name> applies the specified ALTARCHIVELOGDEST behavior to a specific Oracle instance. On RAC, if this option is used, you must specify the ALTARCHIVELOGDEST parameter on each node.
	◆ THREADID <id> applies the specified ALTARCHIVELOGDEST behavior to a specific thread number.
	You can specify more than one ALTARCHIVELOGDEST parameter for any given Oracle instance. In that case, Extract searches each one in the order specified with ALTARCHIVELOGDEST.
	For example:
	`TRANLOGOPTIONS ALTARCHIVELOGDEST PRIMARY INSTANCE`
	`rac1 /disk1/node1/arch,`
	`ALTARCHIVELOGDEST INSTANCE rac1 /disk2/node1/arch,`
	`ALTARCHIVELOGDEST`
	`INSTANCE rac2 /disk1/node2/arch`
	In this example, Extract searches under /disk1/node1/arch for logs related to instance "rac1" then under /disk2/node1/arch if the first search fails. Extract does not check the default location for "rac1." For "rac2", it checks /disk1/node2/arch, then the default location.

Option	Description
`ALTARCHIVELOGDEST ("<backup path>" [FILESPEC "<file pattern>"] [[NOT] RECURSIVE] [PRIMARY])`	(SQL Server) Points Extract to the archived or backup logs when they reside somewhere other than the default location. Extract first checks the specified location and then checks the default location. ◆ Enclose the parameter arguments within parentheses. ◆ "<backup path>" specifies the path name, in double quotes, to the backup logs. You can use wildcard symbols after the last backslash (\ ) delimiter. An asterisk (*) matches zero or more characters. A question mark (?) matches exactly one character. Do not use this option if using NOT RECURSIVE. There can be only one TRANLOGOPTIONS ALTARCHIVELOGDEST entry in a SQL Server Extract parameter file. If there are multiple entries, only the last one will be used. Options: ◆ FILESPEC "<file pattern>" specifies a file pattern within "<backup path>" . Enclose the file pattern within double quotes. An asterisk (*) matches zero or more characters. A question mark (?) matches exactly one character. Do not use a backslash (\ ) delimiter. A backslash allows another path to be specified, which is invalid. ◆ [[NOT] RECURSIVE] specifies whether or not the files specified by "<backup path>" are searched recursively (all sub-directories also searched). ◆ PRIMARY prevents Extract from checking the default log location if it does not find the log in the alternate location. Only the ALTARCHIVELOGDEST path is checked. This is the default.
`ALTLOGDEST <path>`	(MySQL) Specifies the location of the MySQL log index file. Extract looks for the log files in this location instead of the database default location. ALTLOGDEST can be used when the database configuration does not include the full path name to the logs or when there are multiple MySQL installations on the machine. Extract reads the log index file to find the binary log file that it needs to read. When ALTLOGDEST is used, Extract assumes that the logs and the index are in the same location. Supply the full path name to the directory, for example: `TRANLOGOPTIONS  ALTLOGDEST "C:\Program Files\MySQL\MySQL Server 5.1\log\test.index"` On Windows, enclose the path within double quotes if the path contains any spaces.

Option	Description
`ARCHIVEDLOGONLY`	(Oracle) Valid for Extract in classic capture mode. ARCHIVEDLOGONLY causes Extract to read from the archived logs exclusively, without querying or validating the logs from system views such as v$log and v$archived_log. This parameter puts Extract into Archived Log Only mode (ALO). For requirements and more information, see the Oracle GoldenGate *Oracle Installation and Setup Guide*. By default, Extract does not use archived log-only mode even if the database that it connects to is a physical standby database.
`ASMBUFSIZE <size>`	(Oracle) Valid for Extract in classic capture mode. Controls the maximum size, in bytes, of a read operation into the internal buffer that holds the results of each read of the transaction log. Use this option instead of the DBLOGREADERBUFSIZE option if the source Oracle version is one that is: ◆ earlier than Oracle 10.2.0.5 ◆ 11g that is earlier than 11.2.0.2 ◆ any Oracle 11*g* R1 version These versions do not support the newer API that is available in Oracle versions that are supported by the DBLOGREADER option. It is recommended that you use the DBLOGREADER option together with the DBLOGREADERBUFSIZE option if supported by your Oracle version. (See DBLOGREADER.) Higher values increase extraction speed but cause Extract to consume more memory. Low values reduce memory usage but increase I/O because Extract must store data that exceeds the cache size to disk. The following are the valid ranges and default sizes, in bytes: ◆ Minimum: size of one block in the redo log ◆ Maximum: 28672 ◆ Default: 28672 The value of the BUFSIZE option must always be at least equal to, or greater than, the value of ASMBUFSIZE.

Option	Description
ASMUSER SYS@<ASM_instance>, ASMPASSWORD <password> [<algorithm> ENCRYPTKEY {<keyname> | DEFAULT}]	(Oracle) Valid for Extract in classic capture mode. Specifies the user and password for logging into an ASM instance to read the transaction logs. ◆ <user> must be SYS. ◆ < password> is the encrypted password that is copied from the ENCRYPT PASSWORD command results. ◆ <algorithm> specifies the encryption algorithm that was used to encrypt the password: AES128, AES192, AES256, or BLOWFISH. ◆ ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option. ◆ ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option. **Note:** This parameter does *not* replace the standard USERID parameter. Both are required in an ASM environment. ASMUSER is not needed if using the DBLOGREADER option to read the logs. See the Oracle GoldenGate *Windows and UNIX Administrator's Guide* for more information about password security features.
ASYNCTRANSPROCESSING <transaction-buffer-size> | NOASYNCTRANSPROCESSING	(Oracle) Valid for Extract in integrated capture mode. Controls whether integrated capture runs in asynchronous or synchronous processing mode, and controls the buffer size when Extract is in asynchronous mode. ASYNCTRANSPROCESSING is the default. In asynchronous transaction processing mode, there are two threads of control: ◆ One thread groups logical change records (LCR) into transactions, does object-level filtering, and does partial rollback processing, ◆ The other thread formats committed transactions, performs any user-specified transformations, and writes to the trail file. The transaction buffer is the buffer between these two threads and is used to transfer work from one thread to the other. The default transaction buffer size is 300 committed transactions, but is adjusted downward by the Oracle GoldenGate memory manager if its cache memory is close to being exhausted. NOASYNCTRANSPROCESSING disables asynchronous processing and causes Extract to operate in synchronous mode. In this mode, one thread performs all capture work.

Option	Description
`BUFSIZE <size>`	(DB2 LUW, DB2 z/OS, Oracle) Controls the maximum size, in bytes, of the buffers that are allocated to contain the data that is read from the transaction log. - For an Oracle source where Extract is processing file-based redo, this parameter also controls the maximum size, in bytes, of a read operation into the buffer. - For an Oracle source where Extract is processing ASM redo, either ASMBUFSIZE or DBLOGREADERBUFSIZE controls the read size, and in both cases BUFSIZE controls the buffer size. Higher values increase extraction speed but cause Extract to consume more memory. Low values reduce memory usage but increase I/O because Extract must store data that exceeds the cache size to disk. This parameter must be equal to, or greater than, the value that is set for ASMBUFSIZE or DBLOGREADERBUFSIZE (depending on which is in use.) The following are the valid ranges and default sizes, in bytes: *Oracle*: Minimum: 8,192 Maximum: 10,000,000 The default buffer size is determined by the source of the redo data: - For file-based redo, the default is 1000KB (1024000). - For ASM redo, the default is 1000KB (1024000). - For DBLOGREADER redo, the default is 2MB (2097152). *DB2 LUW:* Minimum: 40,960 Maximum: 33,554,432 Default: 131,072 The preceding values must be in multiples of the 4096 page size. Extract will truncate to a multiple if a given value does not meet this requirement. Check with the Systems Administrator to make sure that there is enough ECSA space to support the new buffer size.

Option	**Description**
COMPLETEARCHIVEDLOGONLY | NOCOMPLETEARCHIVEDLOGONLY	(Oracle) Valid for Extract in classic capture mode. Overrides the default Extract processing of archived logs. Possible conditions for this parameter: ◆ **Default in regular mode**: NOCOMPLETEARCHIVEDLOGONLY. Extract starts processing redo data from an archived log immediately when it becomes available, without waiting for it to be written completely to disk. **Override in regular mode**: Use COMPLETEARCHIVEDLOGONLY to force Extract to wait until an archived log is written completely to disk before starting to process redo data. ◆ **Default in archived log only (ALO) mode**: COMPLETEARCHIVEDLOGONLY. Forces Extract to wait for the archived log to be written to disk completely before starting to process redo data. **Override in ALO mode:** Use NOCOMMPLETEARCHIVEDLOGONLY to force Extract to start processing redo data from an archived log immediately when it becomes available. This parameter applies when copying production (source) archive logs to a secondary database where they will serve as the data source. Some Oracle programs do not build the archive log from the first byte to the last byte in sequential order, but instead may copy the first 500MB, then the last 500MB, and finally the middle 1000MB, for example. If Extract begins reading at the first byte, it will abend when it reaches the break in the byte sequencing. Waiting for the whole file to be written prevents this problem. Note that Extract starts to read an archive file before it is completely written to disk, but whether or not it starts to capture data before the file is complete depends on the conditions stated previously.
COMPLETEARCHIVEDLOGTIMEOUT <seconds>	(Oracle) Valid for Extract in classic capture mode. Controls the number of seconds that Extract waits, when in COMPLETEARCHIVEDLOGONLY mode, to try again if it cannot validate that a redo log is being completely written to disk. Use this option in conjunction with the COMPLETEARCHIVEDLOGONLY option of TRANLOGOPTIONS. This option is disabled by default, and Extract will abend after ten seconds if it cannot validate that the file is being written to disk. This check is performed by reading the block header from the last block and verifying against the expected sequence number to determine if the last block has been written out. For <seconds> use any value greater than 0.

Option	Description
DBLOGREADER	(Oracle) Valid for Extract in classic capture mode. Causes Extract to use a newer ASM API that is available as of Oracle 10.2.0.5 and later 10*g* R2 versions, and Oracle 11.2.0.2 and later 11*g* R2 versions (but not in Oracle 11*g* R1 versions). This API uses the database server to access the redo and archive logs, instead of connecting directly to the Oracle ASM instance. The database must contain the libraries that contain the API modules and must be running. To use this feature, the Extract database user must have SELECT ANY TRANSACTION privilege. When used, DBLOGREADER enables Extract to use a read size of up to 4 MB in size. This is controlled with the DBLOGREADERBUFSIZE option The maximum read size when using the default OCI buffer is 28672 bytes. This is controlled by the ASMBUFSIZE option. A larger buffer may improve the performance of Extract when redo rate is high. When using DBLOGREADER, do not use the ASMUSER and ASMPASSWORD options of TRANLOGOPTIONS. The API uses the user and password specified with the USERID parameter.
DBLOGREADERBUFSIZE <buffer size>	(Oracle) Valid for Extract in classic capture mode. Controls the maximum size, in bytes, of a read operation into the internal buffer that holds the results of each read of the transaction log in ASM. Higher values increase extraction speed but cause Extract to consume more memory. Low values reduce memory usage but increase I/O because Extract must store data that exceeds the cache size to disk. Use DBLOGREADERBUFSIZE together with the DBLOGREADER option if the source ASM instance is Oracle 10.2.0.5 or later10*g* R2 versions, or Oracle 11.2.0.2 and later 11*g* R2 versions (but not Oracle 11*g* R1 versions). The newer ASM API in those versions provides better performance than the older one. If the Oracle version is not one of those versions, then ASMBUFSIZE must be used.

Option	Description
	The following are the valid ranges and default sizes, in bytes: ◆ Minimum: size of one block in the redo log ◆ Maximum: 4 MB ◆ Default: 2 MB (2097152) The default should be sufficient in most cases. The value of the BUFSIZE option must always be at least equal to, or greater than, the value of DBLOGREADERBUFSIZE.
`EXCLUDETRANS <trans name>`	(Sybase, SQL Server) Specifies the transaction name of the Replicat database user or any other user so that those transactions are not captured by Extract. Use for bi-directional processing to prevent data looping between the databases. The default transaction name used by Replicat is ggs_repl, but any transaction can be specified with EXCLUDETRANS. For more information about bidirectional synchronization, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*
`EXCLUDEUSER <user name>`	(DB2 LUW, DB2 z/OS, Oracle, Sybase) Specifies the name of the Replicat database user, or of any other user, to be used as a filter that identifies transactions that will be subject to the rules of the GETREPLICATES or IGNOREREPLICATES parameter. Typically, this option is used to identify Replicat transactions in a bi-directional or cascading processing configuration, for the purpose of excluding or capturing them. However, it can be used to identify transactions by any other user, such as those of a specific business application. **DB2 z/OS considerations:** In DB2 for z/OS, the user is always the primary authorization ID of the transaction, which is typically that of the original RACF user who logged on, but also could be a different authorization ID if changed by a transaction processor or by DB2 exits.

Option	Description
	Oracle considerations: For an Oracle database, multiple EXCLUDEUSER statements can be used. All specified users are considered the same as the Replicat user, in the sense that they are subject to the rules of GETREPLICATES or IGNOREREPLICATES. You cannot use wildcards to specify a range of users in one statement. You can use EXCLUDEUSER and EXCLUDEUSERID in the same parameter file. The user name must be valid. Oracle GoldenGate queries the database to get the associated user-id and maps the numeric identifier back to the user name. For this reason, if the specified user is dropped and recreated while name resolution is set to the default of DYNAMICRESOLUTION (page 190), EXCLUDEUSER remains valid. If the same DDL is performed when name resolution is set to NODYNAMICRESOLUTION, EXCLUDEUSER becomes invalid, and Extract must be stopped and then started to make EXCLUDEUSER take effect.
`EXCLUDEUSERID <Oracle uid>`	Specifies the Oracle user-id (uid) of the Replicat database user, or of any other user, to be used as a filter that identifies transactions that will be subject to the rules of the GETREPLICATES or IGNOREREPLICATES parameter. Usage is the same as that of EXCLUDEUSER. The user-id is a non-negative integer with a maximum value of 2147483638. There are several system views that can be queried to get the user-id. The simplest one is the ALL_USERS view. Oracle GoldenGate does not validate the user-id. Note: If the user that is associated with the specified user-id is dropped and recreated, a new user-id will be assigned; therefore, EXCLUDEUSERID will become invalid for that user.

Option	Description
`FETCHLOBIFERROR`	(Oracle) Valid for Extract in classic capture mode. Overrides the Extract default of abending if LOB capture from the redo log results in an error, such as incomplete data. It forces Extract to fetch the LOB from the database if there is an error when reading it from the redo log. **Caution**: If a value gets deleted before the fetch occurs, Extract writes a null to the trail. If a value gets updated before a fetch, Extract writes the updated value. To prevent these inaccuracies, try to keep Extract latency low. The ?racle GoldenGate documentation provides guidelines for tuning process performance. Also, see the ?racle Installation and Setup Guide for instructions on setting fetch options. See also FORCEFETCHLOB.
`FETCHPARTIALLOB`	(Oracle) Valid for Extract in integrated capture mode. Use this option when replicating to a non-Oracle target or in other conditions where the full LOB image is required. It causes Extract to fetch the full LOB object, instead of using the partial change object from the redo record. By default, the database logmining server sends Extract a whole or partial LOB, depending on whether all or part of the source LOB was updated. To ensure the correct snapshot of the LOB, the Oracle Flashback feature must be enabled for the table and Extract must be configured to use it. The Extract FETCHOPTIONS parameter controls fetching and must be set to USESNAPSHOT (the default in the absence of NOUSESNAPSHOT). Without a Flashback snapshot, Extract fetches the LOB from the table, which may be a different image from the point in time when the redo record was generated.
`FETCHPARTIALXML`	(Oracle) Valid for Extract in integrated capture mode. Use this option when replicating to a non-Oracle target or in other conditions where the full LOB image is required. It causes Extract to fetch the full XML document, instead of using the partial change image from the redo record. By default, the database logmining server sends Extract a whole or partial XML document, depending on whether all or part of the source XML was updated. To ensure the correct snapshot of the XML, the Oracle Flashback feature must be enabled for the table and Extract must be configured to use it. The Extract FETCHOPTIONS parameter controls fetching and must be set to USESNAPSHOT (the default in the absence of NOUSESNAPSHOT). Without a Flashback snapshot, Extract fetches the XML document from the table, which may be a different image from the point in time when the redo record was generated.

Option	Description
FILTERTABLE <table_name>	(Extract for SQL/MX) Use this option to specify the name of the checkpoint table being used by Replicat. Operations on the checkpoint table will be ignored by the local Extract as a means of preventing data from looping back to the source. For information about creating a checkpoint table, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*
FORCEFETCHLOB	(Oracle) Valid for Extract in classic capture mode. Overrides the default behavior of capturing LOB data from the redo log. Causes LOBs to be fetched from the database by default. **Caution**: If a value gets deleted before the fetch occurs, Extract writes a null to the trail. If a value gets updated before a fetch, Extract writes the updated value. To prevent these inaccuracies, try to keep Extract latency low. The ?racle GoldenGate documentation provides guidelines for tuning process performance. Also, see the ?racle Installation and Setup Guide for instructions on setting fetch options.
IGNOREDATACAPTURECHANGES | NOIGNOREDATACAPTURECHANGES	(DB2 LUW) IGNOREDATACAPTURECHANGES ignores tables for which DATA CAPTURE CHANGES is not set. Use if tables were specified with a wildcard to ensure that processing continues for tables that do have change capture set. A warning is issued to the error log for tables that were skipped. The default is NOIGNOREDATACAPTURECHANGES.
IGNOREDIRECTLOADINSERTS	(Oracle) Valid for Extract in classic capture mode. Causes Extract to ignore all Oracle direct-load INSERTs. The default behavior (without this parameter) is to capture Oracle direct-load INSERTs. This option applies to Oracle logs with log compatibility of Oracle 10g or later.
INCLUDEREGIONID | INCLUDEREGIONIDWITHOFFSET	(Oracle) Valid for Extract in either capture mode. These options support the Oracle data type TIMESTAMP WITH TIME ZONE specified as TZR (which represents the time zone region, such as "US/Pacific"). By default, Extract abends on TIMESTAMP WITH TIME ZONE if it includes a time zone region. These options enable you to handle this timestamp based on the target database type. ◆ Use INCLUDEREGIONID when replicating from an Oracle source to an Oracle target of the same version or later. When INCLUDEREGIONID is specified, Extract adds a column index and the two-byte TMZ value as a time-zone mapping token and outputs it to the trail in the UTC format of YYYY-MM-DD HH:MI.SS.FFFFFF +00:00.

Option	Description
	◆ Use INCLUDEREGIONIDWITHOFFSET when replicating TIMESTAMP WITH TIME ZONE as TZR from an Oracle source that is v10*g* or later to an Oracle target that is earlier than 10*g*, or from an Oracle source to a target that is not an Oracle database. When INCLUDEREGIONIDWITHOFFSET is specified, Extract converts the time zone region value to a time offset that takes Daylight Saving Time into account based on the date and time. The timestamp data is written to the trail in local time in the format of YYYY-MM-DD HH:MI.SS.FFFFFF TZH:TZM, where TZH:TZM is the region ID converted time offset. When it detects that the source data type is TIMESTAMP and there is a region ID mapping token, Replicat applies the timestamp as follows: ◆ A TIMESTAMP WITH TIME ZONE with TZR is applied if the target Oracle version supports it. ◆ A timestamp with a UTC offset is applied to a non-Oracle database, or to an earlier version of Oracle that does not support TIMESTAMP WITH TIME ZONE with TZR.
`INTEGRATEDPARAMS (<parameter> <value> [, ...])`	(Oracle) Valid for Extract in integrated capture mode (Oracle Standard or Enterprise Edition 11.2.0.3 or later) Passes parameters to the Oracle database logmining server when Extract is in integrated capture mode. The input must be in the form of <parameter> <value>, as in: `TRANLOGOPTIONS INTEGRATEDPARAMS (downsream_real_time_mine Y)` Valid values are: ◆ max_sga_size Specifies the amount of SGA memory that is used by the database logmining server. Can be a positive integer in megabytes. The default is 1 GB if streams_pool_size is greater than 1 GB; otherwise, it is 75% of streams_pool_size. ◆ parallelism Specifies the number of processes supporting the database logmining server. Can be a positive integer. The default is 0.

Option	Description
	◆ downstream_real_time_mine Specifies whether or not integrated capture mines a downstream mining database in real-time mode. A value of Y specifies real-time capture and requires standby redo logs to be configured at the downstream mining database. A value of N specifies capture from archived logs shipped to the downstream mining database. The default is N.
`LEGACYLOBREADING`	Valid for SQL Server and Sybase. Causes the VAM module to use the LOB storage mechanism that was used in Extract versions 11.1.x or earlier. A different LOB storage mechanism is used starting with version 11.2.1.

`LOGRETENTION`
`[ENABLED | SR | DISABLED]`
	(Oracle) Valid for Extract in classic capture mode.
	(Oracle Enterprise Edition 10.2 and later) Use when Extract operates in classic capture mode to specify whether or not Oracle Recovery Manager (RMAN) retains the log files that Extract needs for recovery. When you use the REGISTER EXTRACT command, the logs are retained from the time that the command is issued, based on the current database SCN. The logs are retained until manually deleted. *Note*: This parameter does not enable or disable RMAN within the database itself.
	ENABLED
	Enables the log-retention feature. This is the default, except when Extract for an Oracle database is in Archived Log Only (ALO) mode. Extract must be registered with the database by using the REGISTER EXTRACT command with the LOGRETENTION option.
	Important: To support RMAN log retention on Oracle RAC, you must download and install the database patch that is provided in BUGFIX 11879974, before you add the Extract groups.
	ENABLED honors the SCN of the Bounded Recovery checkpoint and retains the logs up to and including that point. This checkpoint represents the log file of the oldest open *non-persisted* transaction. In the unlikely event that a problem with Bounded Recovery affects the persisted data, the logs that are required to reprocess the oldest open transaction must be available. To be more conservative, you can use the SR option instead. (To understand Bounded Recovery, see “BR” on page 134.)

Option	Description
	SR Enables the log-retention feature, but retains logs up to and including the SCN of the log that is required for Extract to revert to standard (normal) recovery mode. In normal mode, Extract needs access to the log that contains the oldest open transaction that it had in memory. Using SR is a conservative measure that retains more logs than would be retained in Bounded Recovery mode (the default), but it ensures data availability in case Bounded Recovery fails. Extract must be registered with the database by using the REGISTER EXTRACT command with the LOGRETENTION option. DISABLED Disables the log-retention feature, even if REGISTER EXTRACT is used. This is the default setting when Extract for an Oracle source is operating in Archived Log Only (ALO) mode, but you can override this if needed. Use the UNREGISTER EXTRACT command to unregister the associated Extract group from the database after disabling log retention. See page 52. Other information about LOGRETENTION: ◆ If the Oracle flash recovery storage area is full, RMAN will purge the archive logs even when needed by Extract. This limitation exists so that the requirements of Extract (and other Oracle replication components) do not interfere with the availability of redo to the database. ◆ LOGRETENTION makes use of an underlying (but non-functioning) Oracle Streams Capture process; thus, it requires the database to be the Enterprise Edition of Oracle version 10.2 or higher. Oracle Standard Edition and Express Edition do not support this feature. The LOGRETENTION feature can operate concurrently with other Streams installations. ◆ The database user that is assigned to Extract and specified with the USERID parameter must have certain privileges, which are the same as those required for DBLOGIN (see page 77).

Option	Description
`LOGSOURCE <platform>, [PATHMAP <path to logs]`	(Oracle) Valid for Extract in classic capture mode. (Oracle) Specifies the operating system and (optionally) the path name when redo and/or archived logs are stored on a platform other than the one which is hosting the database. Valid values: ◆ AIX ◆ HPUX ◆ LINUX ◆ MVS ◆ SOLARIS ◆ VMS ◆ WINDOWS ◆ S390 Use the optional PATHMAP option to specify the path to the logs. To maintain correct data alignment, the LOGSOURCE platform and the platform that Extract is running on must have the same: ◆ endian order ◆ bit width (as in 32-bit, 64-bit) The following are compatible endian platforms: ◆ Big endian: AIX, HPUX, MVS, SOLARIS, S290 ◆ Little endian: LINUX, VMS, WINDOWS For example when running Extract on HPUX a LOGSOURCE setting of AIX is valid but LINUX is not. When LOGSOURCE is used, put the entire TRANLOGOPTIONS statement on one line. Do not use ampersand (&) line terminators to split it into multiple lines.
`MANAGESECONDARYTRUNCATIONPOINT | NOMANAGESECONDARYTRUNCATIONPOINT`	(SQL Server 2005 and Sybase) Controls whether or not Oracle GoldenGate maintains the secondary truncation point. **SQL Server 2005:** Use MANAGESECONDARYTRUNCATIONPOINT if Oracle GoldenGate will *not* be running concurrently with SQL Server replication, so that Oracle GoldenGate will maintain the secondary truncation point. Use NOMANAGESECONDARYTRUNCATIONPOINT if Oracle GoldenGate will be running concurrently with SQL Server replication. Allows SQL Server replication to manage the secondary truncation point.

Option	Description
	Sybase: Use MANAGESECONDARYTRUNCATIONPOINT if Oracle GoldenGate will not be running concurrently with Sybase Replication Server. Extract will manage the secondary truncation point. Use NOMANAGESECONDARYTRUNCATIONPOINT when you do not want to truncate the Sybase transaction log. Extract will not manage the secondary truncation point. You can use this option when Extract must re-read the Sybase transaction log from a previous log position for debugging purposes.
`MAXREADSIZE <records>`	Valid for Sybase. Specifies how many records Extract will read from the transaction log at one time. Can be used to improve performance. Valid values are integers from 1 through 50000. The default is 256 records. Be careful when adjusting this parameter to very high values. It will reduce the frequency at which Extract adjusts the secondary truncation point, and log data can accumulate. Start with 10000 and evaluate performance before adjusting upward.
`MAXWARNEOF <seconds>`	(Oracle) Valid for Extract in classic capture mode. Specifies the number of seconds that Extract waits for a new log file to become available before generating a warning message. Extract generates only one warning message for a given sequence number. If MAXWARNEOF is not specified, Extract waits for one hour by default. A value of 0 omits the warning no matter how long Extract waits.

`MININGUSER {/ | <user id>}`
`[, MININGPASSWORD <password>]`
`[<algorithm> ENCRYPTKEY`
`{<keyname> | DEFAULT}]`
`[SYSDBA]]`
	(Oracle) Valid for Extract in integrated capture mode.
	Specifies login credentials for Extract in integrated capture mode to log in to a downstream Oracle mining database to interact with the logmining server. This user must:
	◆ Have the privileges granted in dbms_goldengate_auth.grant_admin_privilege.
	◆ Be the user that issues MININGDBLOGIN and REGISTER EXTRACT or UNREGISTER EXTRACT for the Extract group that is associated with this MININGUSER.
	◆ Not be changed while Extract is in integrated capture mode.

Option	Description

	MININGUSER options:
	◆ / directs Oracle GoldenGate to use an operating-system login for Oracle, not a database user login. Use this argument only if the database allows authentication at the operating-system level. Bypassing database-level authentication eliminates the need to update Oracle GoldenGate parameter files if application passwords frequently change.
	To use this option, the correct user name must exist in the database, in relation to the value of the Oracle OS_AUTHENT_PREFIX initialization parameter. The value set with OS_AUTHENT_PREFIX is concatenated to the beginning of a user's operating system account name and then compared to the database name. Those two names must match.
	When OS_AUTHENT_PREFIX is set to " " (a null string), the user name must be created with "identified externally." For example, if the OS user name is “ogg,” you would use the following to create the database user:
	`CREATE USER ogg IDENTIFIED EXTERNALLY;`
	When OS_AUTHENT_PREFIX is set to OPS$ or another string, the user name must be created in the format of:
	`<OS_AUTHENT_PREFIX_value><OS_user_name>`
	For example, if the OS user name is “ogg,” you would use the following to create the database user:
	`CREATE USER ops$ogg IDENTIFIED BY oggpassword;`
	◆ <user id> specifies the name of the mining database user or a SQL*Net connect string.
	◆ <password> is the password. Use when database authentication is required to specify the password for the database user.
	If the password was encrypted by means of the ENCRYPT PASSWORD command, supply the encrypted password; otherwise, use the clear-text password. If the password is case-sensitive, type it that way.
	If either the user ID or password changes, the change must be made in the Oracle GoldenGate parameter files, including the re-encryption of the password if necessary.

Option	Description

	◆ <algorithm> specifies the encryption algorithm that was used to encrypt the password with ENCRYPT PASSWORD. Can be one of:
	`AES128`
	`AES192`
	`AES256`
	`BLOWFISH`
	◆ ENCRYPTKEY {<keyname> | DEFAULT} specifies the encryption key that was specified with ENCRYPT PASSWORD.
	ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option.
	ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option.
	◆ SYSDBA specifies that the user logs in as sysdba.

`NODDLCHANGEWARNING`	(SQL Server) Forces Extract not to log a warning when a DDL operation is made to a source object for which Extract is capturing data. The default is to report a warning, so that the problem can be corrected. Oracle GoldenGate does not support DDL capture and replication for SQL Server, so it expects source and target metadata to remain constant. Some DDL changes do not cause Extract to abend, but the warning still will be logged whenever such changes occur. NODDLCHANGEWARNING will prevent those messages from accumulating in the Oracle GoldenGate log.
`NOFLUSH`	(DB2 z/OS) Inhibits the flushing of log buffers.
`PATHMAP <NFS mount point> <log path>`	(Oracle) Valid for Extract in classic capture mode. Use to specify the location of the redo and/or archived logs when they are stored on a system other than the one which is hosting the database. Specify the NFS mount point followed by the path to the Oracle log structure(s). More than one PATHMAP statement can be used. Can be used with the LOGSOURCE option if the system is a different platform from the one that hosts the database. When PATHMAP is used, put the entire TRANLOGOPTIONS statement on one line. Do not use ampersand (&) line terminators to split it into multiple lines.

Option	Description

`PURGEORPHANEDTRANSACTIONS |`
`NOPURGEORPHANEDTRANSACTIONS`
	(Oracle) Valid for Extract in classic capture mode.
	Enables or disables purging of orphaned transactions that occur when an Oracle RAC node fails and Extract cannot capture the rollback. A transaction is verified as orphaned before purging by comparing its startup time with the node's startup time; if the transaction started earlier, it is purged. Default is PURGEORPHANEDTRANSACTIONS.

`QUERYRETRYCOUNT <number of retries>`	(Extract for SQL Server) Specifies how many times to retry a query to obtain table metadata after timeouts. The default is one retry after a 30-second wait, after which the process abends if the retry fails. QUERYRETRYCOUNT can be specified to retry multiple times at 30-second intervals, according to the input value that is supplied. If all of the retries fail, Extract abends with the normal connection error message. Timeouts can occur for a long-running transaction that has created any table. The system tables become locked and prevent Extract's query from completing. Example: `TRANLOGOPTIONS QUERYRETRYCOUNT 4` This example causes Extract to attempt its query four times at 30-second intervals.
`READQUEUESIZE <size>`	Valid for Sybase. Specifies the internal queue size, in bytes, for transaction data. It can be increased to improve performance. Valid values are integers from 10 through 50000. The default is 256 bytes. Start with 10000 and evaluate performance before adjusting upward.

`REQUIRELONGDATACAPTURECHANGES |`
`NOREQUIRELONGDATACAPTURECHANGES`
	(DB2 LUW) Controls the response of Extract when:
	◆ DATA CAPTURE is set to NONE or to CHANGES without INCLUDE LONGVAR COLUMNS
	and...
	◆ the parameter file includes Oracle GoldenGate parameters that require the presence of before images for some or all column values: GETBEFOREUPATES, NOCOMPRESSUPDATES, and NOCOMPRESSDELETES.
	Both of those DATA CAPTURE settings prevent the logging of before values for LONGVAR columns. If those columns are not available to Extract, it can affect the integrity of the target data.

Option	Description
	◆ REQUIRELONGDATACAPTURECHANGES Extract abends with an error. ◆ NOREQUIRELONGDATACAPTURECHANGES Extract issues a warning but continues processing the data record.
`TRANSCLEANUPFREQUENCY <minutes>`	(Oracle) Valid for Extract in classic capture mode. Specifies an interval, in minutes, after which Oracle GoldenGate scans for orphaned transactions, and then scans again to delete them. The initial scan marks transactions considered to be orphaned. The second scan confirms they are orphaned, and they are deleted. Valid values are from 1 to 43200 minutes. Default is 10 minutes.
`VAMCOMPATIBILITY {1 | 2}`	(MySQL, SQL M/X, SQL Server, Sybase, Teradata) Allows different metadata structures to be passed across the Oracle GoldenGate API known as the*Vendor Access Module* (VAM), depending on the needs of the individual VAM implementation: ◆ A value of 1 specifies that the original VAM API metadata structure is being used. This metadata structure was implemented for Oracle GoldenGate for Teradata. The Teradata modules are the only VAM modules that should require the use of TRANLOGOPTIONS VAMCOMPATIBILITY, because they are configured as shared libraries and are not set programmatically, as with newer versions. Use TRANLOGOPTIONS VAMCOMPATIBILITY if using a new Oracle GoldenGate for Teradata Extract with an older TAM module to support backward compatibility with the older module. If you set the VAM compatibility with VAMInitialize, it does not have to be set with TRANLOGOPTIONS. This parameter is not needed if the Extract and the TAM module are the same version. ◆ A value of 2 specifies the use of an enhanced VAM API metadata structure, based on the original but with additional fields. This is currently used by the Oracle GoldenGate for Sybase product. Because this value is set programmatically within the VAM, the use of TRANLOGOPTIONS VAMCOMPATIBILITY is not required.

Example 1 The following specifies the location of the archived logs.

```
TRANLOGOPTIONS ALTARCHIVELOGDEST /fs1/oradata/archive/log2
```

Example 2 The following Oracle example filters for two users (one by name and one by user-id), whose transactions will be handled according to the GETREPLICATES or IGNOREREPLICATES rules, and it sets a new transaction buffer size.

```
TRANLOGOPTIONS EXCLUDEUSER ggsrep, EXCLUDEUSERID 90, BUFSIZE 100000
```

Example 3 The following excludes the Replicat transaction name in a SQL Server or Sybase environment.

```
TRANLOGOPTIONS EXCLUDETRANS “ggs_repl”
```

Example 4 The following shows how to deal with transaction logs that are on a platform other than the one which hosts the database. Note: This statement spans multiple lines only because of space constraints in this documentation.

```
TRANLOGOPTIONS, LOGSOURCE VMS, PATHMAP
DKA200:[RDBMS.ORACLE.ORA9201I.64.ADMIN.GGS.ARCH]
/net/deltan/uservol1/RDBMS.DIR/ORACLE.DIR/ORA9201I.DIR/
64.DIR/admin.DIR/ggs.DIR/ARCH.dir PATHMAP
DKA200:[RDBMS.ORACLE.ORA9201I.64.ORADATA.GGS]
/net/deltan/uservol1/rdbms.dir/oracle.dir/ora9201I.DIR/
64.dir/oradata.dir/ggs.dir
```

Example 5 The following shows some examples of how to use ALTONLINELOGS. Note that the paths that have spaces in them are enclosed within quote marks.

```
TRANLOGOPTIONS ALTONLINELOGS ("third one/log1.txt")
TRANLOGOPTIONS ALTONLINELOGS( "first one/log1.txt", second_one/log2.txt )
TRANLOGOPTIONS ALTONLINELOGS ( sixth_one/log1.txt, seventh_one/log2.txt, &
   "eighth one/log2.txt")
```

Example 6 The following shows how to encrypt the ASM user’s password.

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

TRANSACTIONTIMEOUT

Valid for Replicat

Use the TRANSACTIONTIMEOUT parameter to prevent an uncommitted Replicat target transaction from holding locks on the target database and consuming its resources unnecessarily. You can change the value of this parameter so that Replicat can work within existing application timeouts and other database requirements on the target.

TRANSACTIONTIMEOUT limits the amount of time that Replicat will hold a target transaction open if it has not received the end-of-transaction record for the last source transaction in that transaction. By default, Replicat groups multiple source transactions into one target transaction to improve performance, but it will not commit a partial source transaction and will wait indefinitely for that last record. The Replicat parameter GROUPTRANSOPS controls the minimum size of a grouped target transaction.

The following events could last long enough to trigger TRANSACTIONTIMEOUT:

- Network problems prevent trail data from being delivered to the target system.
- Running out of disk space on any system, preventing trail data from being written.
- Collector abends (a rare event).

- Extract abends or is terminated in the middle of writing records for a transaction.
- An Extract data pump abends or is terminated.
- There is a source system failure, such as a power outage or system crash.

How TRANSACTIONTIMEOUT works

During normal operations, Replicat remembers the position in the trail of the beginning of the first source transaction in the current target transaction, in case the transaction must be abended and retried. When TRANSACTIONTIMEOUT is enabled, Replicat also saves the position of the first record of the current source transaction and will use that position as the logical end-of-file (EOF) if TRANSACTIONTIMEOUT is triggered.

When triggered, TRANSACTIONTIMEOUT does the following:

1. Aborts the current target transaction
2. Repositions to the beginning of the first source transaction in the aborted target transaction.
3. Processes all of the trail records up to the logical end-of-file position (the beginning of the last, incomplete source transaction).
4. Commits the transaction at logical EOF point.
5. Waits for new trail data before processing any more trail records.

TRANSACTIONTIMEOUT can be triggered multiple times for the same source transaction, depending on the nature of the problem that is causing the trail data to arrive slowly enough to trigger TRANSACTIONTIMEOUT.

Finding out if there is a TRANSACTIONTIMEOUT condition

To determine whether or not Replicat is waiting for the rest of a source transaction when TRANSACTIONTIMEOUT is enabled, issue the SEND REPLICAT command with the STATUS option. The following statuses indicate this condition:

```
Performing transaction timeout recovery
Waiting for data at logical EOF after transaction timeout recovery
```

Default Disabled

Syntax `TRANSACTIONTIMEOUT <n> <units>`

Option	Description
`<n>`	An integer that specifies the wait interval. Valid values are from one second to one week (seven days). This value should be greater than that set with the EOFDELAY parameter in both the primary Extract and any associated data pumps.
`<units>`	One of the following: S, SEC, SECS, SECOND, SECONDS, MIN, MINS, MINUTE, MINUTES, HOUR, HOURS, DAY, DAYS.

Example `TRANSACTIONTIMEOUT 5 S`

TRANSMEMORY

Valid for Extract for DB2 on z/OS and NonStop SQL/MX

Use the TRANSMEMORY parameter to control the amount of memory and temporary disk space available for caching uncommitted transaction data. Because Oracle GoldenGate sends only committed transactions to the target database, it requires sufficient system memory to store transaction data on the source system until either a commit or rollback indicator is received.

This parameter is for use with a DB2 database on z/OS and for a NonStop SQL/MX database. For all other databases, use the CACHEMGR parameter.

About memory management with TRANSMEMORY

TRANSMEMORY enables you to tune the Oracle GoldenGate transaction cache size and define a temporary location on disk for storing data that exceeds the size of the cache. Options are available for defining the total cache size, the per-transaction memory size, the initial and incremental memory allocation, and disk storage space.

Transactions are added to the memory pool specified by RAM, and each is flushed to disk when TRANSRAM is reached. An initial amount of memory is allocated to each transaction based on INITTRANSRAM and is increased by the amount specified by RAMINCREMENT as needed, up to the maximum set with TRANSRAM. Consequently, the value for TRANSRAM should be evenly divisible by the sum of (INITTRANSRAM + RAMINCREMENT).

To view current TRANSMEMORY settings, use the VIEW REPORT <group> command in GGSCI.

Special z/OS considerations

On a z/OS system, the RAM option not only controls the total virtual memory allocation for all cached transactions, but also controls the size of the heap memory that is allocated during startup. The large default value prevents fragmentation within the virtual memory pool, but in some installations it could cause virtual memory to be wasted, especially if the applications primarily generate small transactions. Allocating a large amount of heap memory also can cause Extract to be unresponsive at startup until z/OS completes the allocation.

On z/OS, set RAM just large enough to hold enough transaction activity without affecting the performance of Extract. If set too low, it can cause Extract to write transaction data to disk, causing Extract to run more slowly and to consume disk space. You might need to do some testing to determine the optimal value.

Default None

Syntax

```
TRANSMEMORY
[RAM <size>]
[TRANSRAM <size>]
[TRANSALLSOURCES <size>]
[INITTRANSRAM <size>]
[RAMINCREMENT <size>]
[DIRECTORY (<directory name>, <max dir size>, <max file size>)]
```

Option	Description
`RAM <size>`	Specifies the total amount of memory to use for all cached transactions. On z/OS this also is the initial amount of memory to allocate *per transaction*. The default is 200 megabytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k
`TRANSRAM <size>`	Specifies the total amount of memory to use for a single transaction. The default is 50 megabytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k TRANSRAM should be evenly divisible by both INITTRANSRAM and RAMINCREMENT for optimal results.
`TRANSALLSOURCES <size>`	Specifies the total amount of memory and disk space to use for a single transaction. The default is 50% of total available memory (memory and disk). The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k
`INITTRANSRAM <size>`	(NonStop system only) Specifies the initial amount of memory to allocate for a transaction. The default is 500 kilobytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k
`RAMINCREMENT <size>`	Specifies the amount of memory to increment when a transaction requires more memory. The default is 500 kilobytes. The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k

`DIRECTORY (<directory name>,`
`<max dir size>,`
`<max file size>)`
	Specifies temporary disk storage for transaction data when its size exceeds the maximum specified with TRANSRAM. You can specify DIRECTORY more than once.
	◆ <directory name> is the fully qualified name of a directory. The default is the dirtmp sub-directory of the Oracle GoldenGate directory.
	◆ <max dir size> is the maximum size of all files in the directory. The default is 2 gigabytes. If the space specified is not available, then 75% of available disk space is used.
	◆ <max file size> is the maximum size of each file in the directory. The default is 200 megabytes.

Option	Description
	Values can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms: GB | MB | KB | G | M | K | gb | mb | kb | g | m | k The directory size and file size must be greater than the size of memory specified with RAM. The file names use the following format. <group>_trans_00001.mem or ... <PID>_trans_00001.mem A group name is used for online processes. A system process ID number (PID) is used for one-time runs specified with the SPECIALRUN parameter. The format for a threaded Extract is similar to the following, depending on the database. <group>_<thread #>_00001.mem

Example 1 The following example allows per-transaction memory to be incremented ten times before data is flushed to disk, once for the initial allocation specified with INITTRANSRAM and then nine more times as permitted by RAMINCREMENT.

```
TRANSMEMORY DIRECTORY(c:\test\dirtmp, 3000000000,
300000000), RAM 8000K, TRANSRAM 1000K, INITTRANSRAM 100K,
RAMINCREMENT 100K
```

Example 2 The following is the same as the preceding example, but with the addition of a second directory.

```
TRANSMEMORY DIRECTORY(c:\test\dirtmp, 3000000000,
300000000), DIRECTORY (c:\test\dirtmp2, 1000000000,
5000000), RAM 8000K, TRANSRAM 1000K, INITTRANSRAM 100K,
RAMINCREMENT 100K
```

NOTE In the previous examples, the parameter specification spans multiple lines because of space constraints. In an actual parameter file, multi-line parameter specifications must contain an ampersand (&) at the end of each line.

TRIMSPACES | NOTRIMSPACES

Valid for Extract and Replicat

Use the TRIMSPACES and NOTRIMSPACES parameters to control whether or not trailing spaces in a source CHAR column are truncated when applied to a target CHAR or VARCHAR column.

NOTE Sybase treats all CHAR types as VARCHAR types, and therefore TRIMSPACES will have no effect. For Sybase, use the TRIMVARSPACES parameter.

TRIMSPACES is applied only to single-byte white spaces (U+0020). Ideographic spaces (U+3000) are not supported.

TRIMSPACES and NOTRIMSPACES can be used as on-off switches for different TABLE or MAP statements in a parameter file. They also can be used within an individual TABLE or MAP statement and will override any global settings for that particular MAP or TABLE statement.

For Extract, TRIMSPACES only has an effect if Extract is performing mapping within the TABLE statement (by means of a TARGET statement).

Default `TRIMSPACES`

Syntax `TRIMSPACES | NOTRIMSPACES`

Example The following keeps the default of trimming spaces, except for the last MAP statement.

```
MAP fin.src1, TARGET fin.tgt1;
MAP fin.src2, TARGET fin.tgt2;
MAP fin.src3, TARGET fin.tgt3;
NOTRIMSPACES
MAP fin.src4, TARGET fin.tgt4;
```

TRIMVARSPACES | NOTRIMVARSPACES

Valid for Extract and Replicat

Use the TRIMVARSPACES and NOTRIMVARSPACES parameters to control whether or not trailing spaces in a source VARCHAR column are truncated when applied to a target CHAR or VARCHAR column.

The default is NOTRIMVARSPACES because spaces in a VARCHAR column could actually be part of the data. Before using TRIMVARSPACES, make certain that trailing spaces are not integral to the target data.

TRIMVARSPACES and NOTRIMVARSPACES can be used as on-off switches for different TABLE or MAP statements in a parameter file. They also can be used within an individual TABLE or MAP statement and will override any global settings for that particular MAP or TABLE statement.

For Extract, TRIMVARSPACES only has an effect if Extract is performing mapping within the TABLE statement (by means of a TARGET statement).

Default `NOTRIMVARSPACES`

Syntax `TRIMVARSPACES | NOTRIMVARSPACES`

Example The following enables the trimming of trailing spaces for the last MAP statement.

```
MAP fin.src1, TARGET fin.tgt1;
MAP fin.src2, TARGET fin.tgt2;
MAP fin.src3, TARGET fin.tgt3;
TRIMVARSPACES
MAP fin.src4, TARGET fin.tgt4;
```

UPDATEDELETES | NOUPDATEDELETES

Valid for Replicat

Use the UPDATEDELETES parameter to convert delete operations to update operations for all MAP statements that are specified after it in the parameter file. Use NOUPDATEDELETES to turn off UPDATEDELETES.

When using UPDATEDELETES, use the NOCOMPRESSDELETES parameter. This parameter causes Extract to write all of the columns to the trail, so that they are available for updates.

Default NOUPDATEDELETES

Syntax

```
UPDATEDELETES | NOUPDATEDELETES
```

UPDATEINSERTS | NOUPDATEINSERTS

Valid for Replicat

Use the UPDATEINSERTS parameter to convert insert operations to update operations for all MAP statements that are specified after it in the parameter file. Use NOUPDATEINSERTS to turn off UPDATEINSERTS.

Default NOUPDATEINSERTS

Syntax

```
UPDATEINSERTS | NOUPDATEINSERTS
```

UPREPORT

Valid for Manager

Use the UPREPORTMINUTES or UPREPORTHOURS parameter to specify the frequency with which Manager reports Extract and Replicat processes that are running. Every time one of those processes starts or stops, events are generated. Those messages are easily overlooked in the error log because the log can be so large. UPREPORTMINUTES and UPREPORTHOURS report on a periodic basis to ensure that you are aware of the process status.

To report on stopped processes, use the DOWNREPORT parameter. For more information, see page 187.

Default Do not report running processes

Syntax

```
UPREPORTMINUTES <minutes> | UPREPORTHOURS <hours>
```

Argument	Description
`<minutes>`	The frequency, in minutes, to report processes that are running.
`<hours>`	The frequency, in hours, to report processes that are running.

Example The following generates a report every 30 minutes.

```
UPREPORTMINUTES 30
```

USEANSISQLQUOTES

Valid for GLOBALS

Use the USEANSISQLQUOTES parameter to enable Oracle GoldenGate to recognize column names that are case-sensitive or that contain special characters. USEANSISQLQUOTES enables Oracle GoldenGate to follow SQL-92 rules for using quotation marks to delimit identifiers and literal strings.

With USEANSISQLQUOTES enabled, Oracle GoldenGate treats a string within double quotes as a column name, and it treats a string within single quotes as a literal. By default, Oracle GoldenGate treats a string within double quotes as a literal when used in COLMAP, FILTER, WHERE, SQLEXEC, and other options that take object names. For object selection and mapping, Oracle GoldenGate recognizes double-quoted strings as object names without the need for USEANSISQLQUOTES.

For example, consider the behavior of the @STRLEN conversion function, which returns a string length:

- By default (without USEANSISQLQUOTES) @STRLEN returns a value of 3 because Oracle GoldenGate interprets the double-quoted “ABC” as a literal.

  ```
  COLMAP ( TGT1 = @STRLEN("ABC") )
  ```

- With USEANSISQLQUOTES, Oracle GoldenGate interprets the double-quoted “ABC” as a column name, and @STRLEN returns the length of whatever the *value* is for column “ABC.”

  ```
  COLMAP ( TGT1 = @STRLEN("ABC") )
  ```

- With USEANSISQLQUOTES, Oracle GoldenGate interprets the single-quoted ‘ABC’ as a literal, and @STRLEN returns 3.

  ```
  COLMAP ( TGT1 = @STRLEN(‘ABC’) )
  ```

USEANSISQLQUOTES supports the following options of the TABLE and MAP parameters. These options can take literal strings, column names, column-conversion functions, or some combination of the previous, as input.

- Column-conversion function that takes a literal string or column name as an argument
- COLMAP
- EVENTACTIONS
- FILTER
- SQLEXEC (in TABLE or MAP and also standalone SQLEXEC)
- TOKENS
- WHERE

When used, USEANSISQLQUOTES affects all TABLE and MAP statements in the local Oracle GoldenGate instance.

Default Disabled: Double quotes indicate a literal.

Syntax `USEANSISQLQUOTES`

USEIPV6

Valid for GLOBALS

Use the USEIPV6 parameter to force the use of Internet Protocol version 6 (IPv6) by Oracle GoldenGate for TCP/IP connections. By default, Oracle GoldenGate uses IPv6 in dual-stack mode but falls back to IPv4, and only then to IPv6. USEIPV6 eliminates the IPv4 fallback step. The order of socket selection becomes:

- IPv6 dual-stack
- IPv6

When USEIPV6 is used, the entire network in which Oracle GoldenGate operates must be IPv6 compatible.

Default Disabled

Syntax `USEIPV6`

USERID

Valid for Manager, Extract, Replicat, DEFGEN

Use the USERID parameter to specify the type of authentication for an Oracle GoldenGate process to use when logging into a database, and to specify password encryption information. For more information about Oracle GoldenGate encryption, see the security guidelines in the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Specify USERID before any TABLE entries in the parameter file.

When and how to use USERID

USERID is not always required, nor is PASSWORD always required when USERID is required. In some cases, it is sufficient just to use USERID or even just to use the SOURCEDB or TARGETDB parameter, depending on how authentication for the database is configured.

See also SOURCEDB and TARGETDB.

USERID requirements per database type

NOTE For privileges that are required for the user that is specified with USERID, see the Oracle GoldenGate installation guide for the database.

DB2 for LUW

Use USERID with PASSWORD with SOURCEDB or TARGETDB for all Oracle GoldenGate processes that connect to a DB2 LUW database using database authentication. You can omit USERID and PASSWORD (and only use SOURCEDB or TARGETDB) if the database is configured allow authentication at the operating-system level. In this case, the operating system user must have the appropriate privileges as outlined in the Oracle GoldenGate *DB2 LUW Installation and Setup Guide*.

DB2 for z/OS database

Use USERID with PASSWORD if the user that is assigned to the Oracle GoldenGate process does not have the DB2 privileges that are required for the process to function properly.

MySQL

Use USERID with PASSWORD for all Oracle GoldenGate processes that connect to a MySQL database.

Oracle

Use USERID for Oracle GoldenGate processes that connect to an Oracle database.

- To use an operating system login, use USERID with the / argument.
- To use a database user name and password, use USERID with PASSWORD.
- Optionally, you can specify the user to log in as sysdba.

(*Oracle Enterprise Edition version 10.2 and later*) Special database privileges are required for the USERID user when Extract is configured to use LOGRETENTION. These privileges might have been granted when Oracle GoldenGate was installed. See the system requirements in the Oracle GoldenGate *Oracle Installation and Setup Guide*. For more information about LOGRETENTION, see “TRANLOGOPTIONS” on page 385.

(*Oracle Standard or Enterprise Edition 11.2.0.2 or later*) To use USERID for an Extract group that is configured for integrated capture, the following must be true:

- The user must have the privileges granted in the dbms_goldengate_auth.grant_admin_privilege procedure.
- The user must be the user that issues DBLOGIN and REGISTER EXTRACT or UNREGISTER EXTRACT for the Extract group that is associated with this USERID.

SQL/MX

- For Oracle GoldenGate processes that connect to a source SQL/MX database, use USERID without PASSWORD to specify the default schema. Also use the SOURCEDB parameter to specify the catalog name.
- For Oracle GoldenGate processes that connect to a target SQL/MX database, use USERID with PASSWORD. Also use the TARGETDB parameter to specify the target ODBC data source.

NOTE Password encryption is not supported for SQL/MX.

SQL Server

Use USERID with PASSWORD if the ODBC datasource connection that will be used by the Oracle GoldenGate process is configured to supply database authentication. USERID can be a specific login that is assigned to the process or any member of an account in the System Administrators or Server Administrators fixed server role.

- On a source SQL Server system, also use the SOURCEDB parameter to specify the source ODBC data source.
- On a target SQL Server system, also use the TARGETDB parameter to specify the target ODBC data source.

Sybase

Use USERID and PASSWORD for Oracle GoldenGate processes that connect to a Sybase database.

Teradata

Use USERID with PASSWORD for Oracle GoldenGate processes that connect to a Teradata database.

- On a source Teradata system, also use the SOURCEDB parameter to specify the source ODBC data source.
- On a target Teradata system, also use the TARGETDB parameter to specify the target ODBC data source.

TimesTen

Use USERID with PASSWORD if the ODBC datasource connection that will be used by Replicat is configured to supply database authentication. Also use the TARGETDB parameter to specify the target ODBC data source.

Default None

Syntax

```
USERID {/ | <user id>}[, PASSWORD <password>]
[<algorithm> ENCRYPTKEY {<keyname> | DEFAULT}] [SYSDBA]
```

Argument	Description

/	(Oracle) Directs Oracle GoldenGate to use an operating-system login for Oracle, not a database user login. Use this argument only if the database allows authentication at the operating-system level. Bypassing database-level authentication eliminates the need to update Oracle GoldenGate parameter files if application passwords frequently change.
	To use this option, the correct user name must exist in the database, in relation to the value of the Oracle OS_AUTHENT_PREFIX initialization parameter. The value set with OS_AUTHENT_PREFIX is concatenated to the beginning of a user's operating system account name and then compared to the database name. Those two names must match.
	When OS_AUTHENT_PREFIX is set to " " (a null string), the user name must be created with "identified externally."
	For example, if the OS user name is “ogg,” you would use the following to create the database user:
	`CREATE USER ogg IDENTIFIED EXTERNALLY;`
	When OS_AUTHENT_PREFIX is set to OPS$ or another string, the user name must be created in the format of:
	`<OS_AUTHENT_PREFIX_value><OS_user_name>`
	For example, if the OS user name is “ogg,” you would use the following to create the database user:
	`CREATE USER ops$ogg IDENTIFIED BY oggpassword;`

Argument	Description
`<user id>`	Specifies the name of a database user, a schema (SQL/MX) or a SQL*Net connect string (Oracle).
`<password>`	Use when database authentication is required to specify the password for the database user. If the password was encrypted by means of the ENCRYPT PASSWORD command, supply the encrypted password; otherwise, use the clear-text password. If the password is case-sensitive, type it that way. If either the user ID or password changes, the change must be made in the Oracle GoldenGate parameter files, including the re-encryption of the password if necessary.
`<algorithm>`	Specifies the encryption algorithm that was used to encrypt the password with ENCRYPT PASSWORD. Can be one of: AES128 AES192 AES256 BLOWFISH
`ENCRYPTKEY {<keyname> | DEFAULT}`	Specifies the encryption key that was specified with ENCRYPT PASSWORD. ◆ ENCRYPTKEY <keyname> specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME <keyname> option. ◆ ENCRYPTKEY DEFAULT directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option.
`SYSDBA`	(Oracle) Specifies that the user logs in as sysdba.

Example 1

```
USERID /
```

Example 2

```
USERID ogg
```

Example 3

```
USERID ogg@ora1.ora, &
PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC AES128, &
ENCRYPTKEY securekey1
```

Example 4

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

Example 5

```
USERID ogg, PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC &
BLOWFISH, ENCRYPTKEY DEFAULT
```

Example 6

```
USERID ogg, &
PASSWORD AACAAAAAAAAAAAJAUEUGODSCVGJEEIUGKJDJTFNDKEJFFFTC AES128, &
ENCRYPTKEY securekey1 SYSDBA
```

VAM

Valid for Extract

Use the VAM parameter to specify that a Vendor Access Module (VAM) is being used to perform data capture functions for the Extract process and send it to the Extract API. This parameter supplies required input for the VAM API.

Default None

Syntax `VAM <library name>, PARAMS (“<param>” [, “<param>”] [, ...])`

Argument	Description
`<library name>`	The name of the library that is supplied by the database vendor as a Windows DLL or a UNIX shared object. Use the full path name if the library is in a directory other than the Oracle GoldenGate directory. (Note: Teradata calls this library the *Teradata Access Module (TAM)*. This program or library communicates with the Oracle GoldenGate VAM API.

`PARAMS <param>`	◆ PARAMS is a required keyword.
	◆ <param> is any parameter, enclosed within quotes, that is passed to the Oracle GoldenGate API. See the following parameter options.
	PARAMS options for SQL/MX databases:
	ARLIBError <error>, <action>
	(Optional) Specifies how TMFARLIB errors are handled by the VAM.
	◆ <error> is an ARLIB error number.
	◆ <action> can be ABEND | WARN | IGNORE.
	The default is ABEND. Errors -1000 and -2000 will always result in ABEND, regardless of any other action that is specified.
	Examples:
	`Vam Params (arliberror (-16,-14), Warn)`
	`Vam Params (arliberror -2000, Abend)`
	`Vam Params (arliberror -1000, Abend)`
	ARErrorReportInterval <seconds>
	Sets the interval, in seconds, in which the same TMFARLIB error is reported back to Extract. This reduces the amount of messages for each type of error that accumulate.
	◆ <seconds> must be greater than, or equal to, zero.
	◆ The default is 60 seconds.

Argument	Description
	PARAMS options for Teradata databases inifile, <ini file>, callbackLib, extract.exe For the Teradata TAM, this is a required parameter. ◆ inifile indicates that the next parameter specifies the TAM initialization file. ◆ <ini file> is the name of the TAM initialization file. Unless the file resides in the same directory where the Extract program is installed, specify the fully qualified path name. ◆ callbackLib indicates that the next parameter specifies the program that interfaces with the TAM. This parameter is case-sensitive and must be entered exactly as shown here. ◆ extract.exe is the Extract program, which is the callback program for the TAM.

Example

```
VAM tam.dll, PARAMS (inifile, tam.ini, callbackLib, extract.exe)
```

VARWIDTHNCHAR | NOVARWIDTHNCHAR

Valid for Extract, Replicat, DEFGEN

Use the VARWIDTHNCHAR and NOVARWIDTHNCHAR parameters to control how NCHAR data is written to the trail and interpreted by Replicat.

- VARWIDTHNCHAR causes an NCHAR, NVARCHAR2, or NCLOB character set to be treated as a variable-length character set (UTF-8).
- NOVARWIDTHNCHAR causes an NCHAR, NVARCHAR2, or NCLOB character set to be treated as UTF-16.
- If neither option is specified, the NLS_NCHAR_CHARACTERSET property value from the database is used to determine how an NCHAR, NVARCHAR2, or NCLOB character set is treated.

Default Use NLS_NCHAR_CHARACTERSET property from database

Syntax

```
VARWIDTHNCHAR | NOVARWIDTHNCHAR
```

WARNLONGTRANS

Valid for Extract

Use the WARNLONGTRANS parameter to specify a length of time that a transaction can be open before Extract generates a warning message that the transaction is long-running. Also use WARNLONGTRANS to control the frequency with which Oracle GoldenGate checks for long-running transactions.

When WARNLONGTRANS is specified, Oracle GoldenGate checks for transactions that satisfy the specified threshold, and it reports the first one that it finds to the Oracle GoldenGate error log, the Extract report file, and the system log. By default, Oracle GoldenGate repeats this check every five minutes.

To view a list of open transactions on demand, to output transaction details to a file, or to either cancel those transactions or force them to the trail, see the SEND EXTRACT command (page 36).

This parameter is valid for MySQL, Oracle, SQL Server, and Sybase.

In the case of MySQL, WARNLONGTRANS displays either one or no open transactions, because the MySQL log contains only committed transactions. However, WARNLONGTRANS can be used to warn of a transaction that is still being processed by Extract and may be taking too long from the standpoint of log accumulation and archiving schedules.

Default One hour (and check every five minutes using a separate processing thread)

Syntax

```
WARNLONGTRANS <duration><unit>
[, CHECKINTERVAL <interval><unit>]
[, NOUSETHREADS]
[, USELASTREADTIME]
```

Argument	Description

`<duration><unit>`	Sets a length of time after which an open transaction is considered to be long-running. The default is one hour.
	◆ <duration> is the length of time expressed as a whole number.
	◆ <unit> is seconds, minutes, hours, or days in fully spelled out or abbreviated form:
	`S|SEC|SECS|SECOND|SECONDS`
	`M|MIN|MINS|MINUTE|MINUTES`
	`H|HOUR|HOURS`
	`D|DAY|DAYS`
	Do not put a space between <duration> and <unit>.

`CHECKINTERVAL <interval><unit>`	Sets the frequency at which Oracle GoldenGate checks for transactions that satisfy WARNLONGTRANS and reports the longest running one.
	◆ <interval> is the length of time between checks, expressed as a whole number.
	◆ <unit> is seconds, minutes, hours, or days in fully spelled out or abbreviated form:
	`S|SEC|SECS|SECOND|SECONDS`
	`M|MIN|MINS|MINUTE|MINUTES`
	`H|HOUR|HOURS`
	`D|DAY|DAYS`
	Do not put a space between <interval> and <unit>. The default, and the minimum value, is five minutes.

`NOUSETHREADS`	Specifies that the monitoring will be done by the main process thread. By default, it is done with a separate thread for performance reasons. NOUSETHREADS should only be used if the system does not support multi-threading.

Argument	Description
`USELASTREADTIME`	(Oracle only) Forces Extract to always use the time that it last read the redo log to determine whether a transaction is long-running or not. By default, Extract uses the timestamp of the last record that it read from the redo log. This applies to an Extract that is running in archive log only mode, as configured with TRANLOGOPTIONS using the ARCHIVEDLOGONLY option.

Example
```
WARNLONGTRANS 2h, CHECKINTERVAL 3m
```

WARNRATE

Valid for Replicat

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.

Default 100 errors

Syntax
```
WARNRATE <num errors>
```

Argument	Description
`<num errors>`	The number of SQL errors after which a warning is issued.

Example
```
WARNRATE 1000
```

WILDCARDRESOLVE

Valid for Extract and Replicat

Use the WILDCARDRESOLVE parameter to alter the rules for processing wildcarded table specifications in a TABLE, SEQUENCE, or MAP statement. WILDCARDRESOLVE must precede the associated TABLE, SEQUENCE, or MAP statements in the parameter file.

The target objects must already exist in the target database when wildcard resolution is attempted. If a target object does not exist, Replicat abends.

Default `DYNAMIC`

Syntax
```
WILDCARDRESOLVE {DYNAMIC | IMMEDIATE}
```

Argument	Description
DYNAMIC	Source objects that satisfy the wildcard definition are resolved each time the wildcard rule is satisfied. This is the default. Do not use this option when SOURCEISTABLE or GENLOADFILES is specified; WILDCARDRESOLVE will always be implicitly set to IMMEDIATE for these parameters. This is the required configuration for Teradata. DYNAMIC must be used when using wildcards to replicate Oracle sequences with the SEQUENCE parameter. When the default behavior is required, an explicit WILDCARDRESOLVE DYNAMIC parameter statement is optional. Using one might make it clear to someone who is reviewing the parameter file which method is being used.
IMMEDIATE	Source objects that satisfy the wildcard definition are processed at startup. This option is not supported for Teradata. This is the forced default for SOURCEISTABLE. This option does not support the Oracle interval partitioning feature. Dynamic resolution is required so that new partitions are found by Oracle GoldenGate.

Example The following example resolves wildcards at startup.

```
WILDCARDRESOLVE IMMEDIATE
TABLE hq.acct_*;
```

CHAPTER 3
Collector Parameters

This chapter describes the parameters for the Collector process. The Collector process operates on the target system to receive incoming data and write it to the trail.

Dynamic Collector

Typically, Oracle GoldenGate users do not interact with the Collector process. It is started dynamically by the Manager process. This is known as a *dynamic collector*.

Static Collector

You can run a *static* Collector manually by running the SERVER program at the command line with the following syntax and input parameters as shown:

Syntax

```
server <parameter> [<parameter>] [...]
```

Collector parameters are case-sensitive and must be preceded by a dash.

Table 42 Collector parameters

Parameter	Description
`-cp <checkpoint file>`	Specifies the name of the checkpoint file that Collector maintains for an alias Extract group. <checkpoint file> is the name of the passive Extract group that is associated with the alias Extract group. The checkpoint file is used to determine whether the passive Extract is running or not. It is running when the checkpoint file is locked by Collector (server program). Must be used with the -h and -p parameters. For more information about using passive and alias Extract groups, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

`-d`
`<definitions file>`
	A local file generated by the DEFGEN utility that contains exported definitions of source tables.

`-E`	Converts incoming header and data to EBCDIC format from ASCII. By default, Oracle GoldenGate does not convert the data.

`-e`
`<version error type>`
`<action>`
	Directs Collector to respond to specific formatting error conditions in custom ways. Default values are almost always sufficient. To specify more than one error type, use -e multiple times. For example:
	`-e OLD CONTINUE -e NEW DISCARD.`
	`<version error type>` can be one of the following:

Table 42 Collector parameters (continued)

Parameter	Description

	`NEW`
	Checks for records that contain more data than anticipated (more columns than the current definition). The Collector process may need an updated version of the source table (that is, DEFGEN must be run again). The default action is ABEND.
	`OLD`
	Checks for records that contain less data than anticipated. This usually indicates that a record has fewer columns than the table's current definition, which is considered a normal condition. The default action is CONTINUE.
	`OUTOFSYNC`
	Checks for records that cannot be converted according to the definition provided. The default action is ABEND.
	<action> can be one of the following:
	`ABEND`
	Discards the record and directs the Extract process to end immediately.
	`CONTINUE`
	Processes the record (if possible) regardless of the conversion error encountered.
	`DISCARD`
	Outputs the record to a discard file (if one is specified with –x). Collector sends a warning to the error file for the first discarded record and continues to process records.

`-ENCRYPT`
`<encrypt type>`
	The type of encryption being passed from the Extract process, as specified with the RMTHOST parameter in the Extract parameter file.
	Valid values:
	◆ `NONE`
	◆ `BLOWFISH`
	If using BLOWFISH, also specify the -KEYNAME option. For more information about Oracle GoldenGate security, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide.*

Table 42 Collector parameters (continued)

Parameter	Description
`-f`	Always forces file writes to be flushed to disk before returning a success status to the Extract process. By default, the file system buffers the I/O because it is more efficient than flushing to disk with every operation. Generally, the performance benefits outweigh the small risk that data could be lost if the system fails after an I/O is confirmed successful, but before the buffer actually is flushed to disk. Use -f if this risk is unacceptable, with the understanding that it can compromise the performance of Oracle GoldenGate.
`-g`	Supports files that are larger than 2GB (Solaris only).
`-h <host name or IP address>`	Specifies the name or IP address of the source system. Use this option when using an alias Extract on the target that is associated with an Extract running in PASSIVE mode on the source. It causes Collector to operate in *connection mode*. In this mode, it initiates a TCP/IP connection to the source Extract, instead of waiting for a connection request from Extract. Must be used with the -p Collector option. For more information about using passive and alias Extract groups, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.
`-k`	Directs Collector to terminate when the Extract process that it is serving disconnects. This option is used by the Manager process when starting the Collector process.
`-KEYNAME <name>`	Specifies a key name defined in the local ENCKEYS lookup file. Use if BLOWFISH is specified for -ENCRYPT.
`-l <file name>`	Logs output to the specified file. Use the fully qualified name.
`-m`	Specifies the maximum number of log files to allocate.
`-P <parameter file>`	A local file containing Collector parameters. Parameters in this file override parameters sent from the Extract process.
`-p <port number>`	A TCP/IP port number specified as follows: ◆ For a regular Extract or regular data pump: the port on which the Collector process listens for connection requests from Extract. ◆ For an Extract or data pump running in PASSIVE mode: the port on which Extract or the data pump listens for connection requests from Collector. Must be used with the -h <host> parameter in this case. The default is port 7819. For more information about using passive and alias Extract groups, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Table 42 Collector parameters (continued)

Parameter	Description

`-R`
`<alternate value>`
	Replaces invalid numeric ASCII data with an alternate value. The default is to replace with 0. The alternate value can be one of the following:
	`<number>`
	Specifies a substitute number.
	`NULL`
	Specifies NULL if the target column accepts NULL values; otherwise replaces with zero.
	`UNPRINTABLE`
	Rejects any column with unprintable data. The process stops and reports the bad value.
	`NONE`
	Does not replace numeric data. Oracle GoldenGate attempts to replicate the data as-is.

`-x <discard file>`	Specifies a discard file for outputting records that could not be formatted. Use the fully qualified name.

CHAPTER 4
Column Conversion Functions

Using the column conversion functions of Oracle GoldenGate, you can manipulate source values into the appropriate format for target columns. These functions enable you to manipulate numbers and characters, perform tests, extract parameter values, return environment information, and more.

Summary of column-conversion functions

This summary is organized according to the types of processing that can be performed with the Oracle GoldenGate functions. An alphabetical reference begins on page 431.

Table 43 Performing tests

Function	Description
CASE	Selects a value depending on a series of value tests.
EVAL	Selects a value based on a series of independent tests.
IF	Selects one of two values depending on whether a conditional statement returns TRUE or FALSE.

Table 44 Handling missing columns

Function	Description
COLSTAT	Returns an indicator that a column is MISSING, NULL, or INVALID.
COLTEST	Performs conditional calculations to test whether a column is PRESENT, MISSING, NULL, or INVALID.

Table 45 Working with dates

Function	Description
DATE	Returns a date and time based on the format passed into the source column.
DATEDIFF	Returns the difference between two dates or datetimes.
DATENOW	Returns the current date and time.

Table 46 Performing arithmetic calculations

Function	Description
COMPUTE	Returns the result of an arithmetic expression.

Table 47 Working with strings

Function	Description
NUMBIN	Converts a binary string into a number.
NUMSTR	Converts a string into a number.
STRCAT	Concatenates one or more strings.
STRCMP	Compares two strings.
STREXT	Extracts a portion of a string.
STREQ	Determines whether or not two strings are equal.
STRFIND	Finds the occurrence of a string within a string.
STRLEN	Returns the length of a string.
STRLTRIM	Trims leading spaces.
STRNCAT	Concatenates one or more strings to a maximum length.
STRNCMP	Compares two strings based on a specified number of characters.
STRNUM	Converts a number into a string.
STRRTRIM	Trims trailing spaces.
STRSUB	Substitutes one string for another.
STRTRIM	Trims leading and trailing spaces.
STRUP	Changes a string to uppercase.
VALONEOF	Compares a string or string column to a list of values.

Table 48 Other functions

Function	Description
BINARY	Maintains source binary data as binary data in the target column when the source column is defined as a character column.
BINTOHEX	Converts a binary string to a hexadecimal string.
GETENV	Returns environmental information.
GETVAL	Extracts parameters from a stored procedure as input to a FILTER or COLMAP clause.
HEXTOBIN	Converts a hexadecimal string to a binary string.
HIGHVAL | LOWVAL	Constrains a value to a high or low value.
RANGE	Divides rows into multiple groups of data for parallel processing.
TOKEN	Retrieves token data from a trail record header.

BINARY

Use the @BINARY function when a source column referenced by a column-conversion function is defined as a character column but contains binary data that must remain binary on the target. By default, binary data in a character column is converted (if necessary) to ASCII and assumed to be a null-terminated string. The @BINARY function copies arbitrary binary data to the target column.

Syntax

```
@BINARY(<column name>)
```

Argument	Description
`<column name>`	The name of the target column to which the data will be copied.

Example The following shows how @BINARY would be used to copy the data from the source column ACCT_CREATE_DATE to the target column ACCT_CHIEF_COMPLAINT.

```
ACCT_CHIEF_COMPLAINT =
@IF ( @NUMBIN (ACCT_CREATE_DATE ) < 48633, "xxxxxx",
@BINARY(ACCT_CHIEF_COMPLAINT))
```

BINTOHEX

Use the @BINTOHEX function to convert supplied binary data into its hexadecimal equivalent.

Syntax

```
@BINTOHEX(<data>)
```

Argument	Description
`<data>`	The name of the source column, an expression, or a literal string that is enclosed within quotes.

Example `@BINTOHEX("12345")` converts to "3132333435".

CASE

Use the @CASE function to select a value depending on a series of value tests. There is no limit to the number of cases you can test with @CASE. If the number of cases is large, list the most frequently encountered conditions first for the best performance.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@CASE (<value>, <test value1>, <test result1>
[, <test value2>, <test result2>] [, ...]
[, <default result>]
```

Argument	Description
`<value>`	A value to test, for example, a column name. Enclose literals within quotes.
`<test value>`	A valid result for <value>. Enclose literals within quotes.
`<test result>`	A value to return based on the value of <test value>. Enclose literals within quotes.
`<default result>`	A default value to return if <value> results in none of the <test value> values. Enclose literals within quotes.

Example 1 The following returns "A car" if PRODUCT_CODE is "CAR" and "A truck" if PRODUCT_CODE is "TRUCK". If PRODUCT_CODE fits neither of the first two cases, a FIELD_MISSING indication is returned because a default value was not specified.

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

Example 2 The following is similar to the previous example, except that it provides for a default value. If PRODUCT_CODE is neither "CAR" nor "TRUCK", the function returns "A vehicle."

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

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.

Syntax

```
@COLSTAT ({MISSING | NULL | INVALID})
```

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))
```

COLTEST

Use the @COLTEST function to enable conditional calculations by testing for one or more column conditions. If a condition is satisfied, @COLTEST returns TRUE. To perform the conditional calculation, use the @IF function.

Syntax

```
@COLTEST (<source column>, <test item> [, <test item>] [, ...])
```

Argument	Description	
<source column>	The name of a source column.	
<test item>	Valid values:	
	PRESENT	Indicates a column is present in the source record and not NULL. Column values can be missing if the database does not log values for columns that do not change, but that is not the same as NULL.
	NULL	Indicates a column is present in the source record and NULL.
	MISSING	Indicates a column is not present in the source record.
	INVALID	Indicates a column is present in the source record but contains invalid data.

Example 1 The following example uses @IF to map a value to the HIGH_SALARY column only if the BASE_SALARY column in the source record was both present (and not NULL) and greater than 250000. Otherwise, NULL is returned.

```
HIGH_SALARY =
@IF (@COLTEST (BASE_SALARY, PRESENT) AND
BASE_SALARY > 250000,
BASE_SALARY, @COLSTAT (NULL))
```

Example 2 In the following example, 0 is returned when the AMT column is missing or invalid; otherwise a value for AMT is returned.

```
AMOUNT = @IF (@COLTEST (AMT, MISSING, INVALID), 0, AMT)
```

COMPUTE

Use the @COMPUTE function to return the value of an arithmetic expression to a target column. The value returned from the function is in the form of a string.

You can omit the @COMPUTE phrase when returning the value of an arithmetic expression to another Oracle GoldenGate function, as in:

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

The preceding returns the same result as:

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

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.

Syntax `@COMPUTE(<expression>)`

Argument	Description
`<expression>`	A valid arithmetic expression.

Example 1 `AMOUNT_TOTAL = @COMPUTE (AMT + AMT2)`

Example 2 `AMOUNT_TOTAL = @IF (AMT >= 0, AMT * 100, 0)`

Example 3 `ANNUAL_SALARY = @COMPUTE (MONTHLY_SALARY * 12)`

DATE

Use the @DATE function to return dates and times in a variety of formats to the target column based on the format passed into the source column. @DATE converts virtually any type of input into a valid SQL date. @DATE also can be used to extract portions of a date column or to compute a numeric timestamp column based on a date.

Syntax

```
@DATE ("<output descriptor>", "<input descriptor>", <source col>
[, "<input descriptor>", <source col>] [, ...])
```

Argument	Description
`"<output descriptor>"`	A target string containing date descriptors and optional literal values, such as spaces or colons, required by the target column. Date descriptors can be strung together as needed. See Table 49 on page 435 for descriptions of date descriptors. The format descriptor must match the date/time/timestamp format for the target. Oracle GoldenGate will override the specified format, if necessary, to make it correct.
`"<input descriptor>"`	A source string containing date descriptors and optional literal values, such as spaces or colons. Date descriptors can be strung together as needed. For example: Descriptor string "YYYYMMDD" indicates that the following numeric or character column contains (in order) a four-digit year (YYYY), month (MM), and day (DD). Descriptor string "DD/MM/YY" indicates that the field contains the day, a slash, the month, a slash, and the two digit year. See Table 49 for date descriptions.
`<source col>`	The name of a source column supplying the preceding input.

Table 49 Date Descriptors

Descriptor	Description	Valid for...
CC	Century	Input/Output
YY	Two-digit year	Input/Output

Table 49 Date Descriptors (continued)

Descriptor	Description	Valid for...
YYYY	Four-digit year	Input/Output
MM	Numeric month	Input/Output
MMM	Alphanumeric month, such as APR, OCT	Input/Output
DD	Numeric day of month	Input/Output
DDD	Numeric day of the year, such as 001 or 365	Input/Output
DOW0	Numeric day of the week (Sunday = 0)	Input/Output
DOW1	Numeric day of the week (Sunday = 1)	Input/Output
DOWA	Alphanumeric day of the week, such as SUN, MON, TUE	Input/Output
HH	Hour	Input/Output
MI	Minute	Input/Output
SS	Seconds	Input/Output
JTSLCT	Use for a Julian timestamp that is already local time, or to keep local time when converting to a Julian timestamp.	Input/Output
JTSGMT	Julian timestamp, the same as JTS.	Input/Output
JTS	Julian timestamp. JUL and JTS produce numbers you can use in numeric expressions. The unit is microseconds. On a Windows machine, the value will be padded with zeros (0) because the granularity of the Windows timestamp is milliseconds.	Input/Output
JUL	Julian day. JUL and JTS produce numbers you can use in numeric expressions.	Input/Output
TTS	NonStop 48-bit timestamp	Input
PHAMIS	PHAMIS application date format	Input
FFFFFF	Fraction (up to microseconds)	Input/Output
STRATUS	STRATUS application timestamp	Input/Output
CDATE	C timestamp in seconds since the Epoch	Input/Output

Example 1 In an instance where a two-digit year is supplied, but a four-digit year is required in the output, several options exist to obtain the correct century.

- The century can be hard coded, as in:

  ```
  "CC", 19 or "CC", 20
  ```

- The @IF function can be used to set a condition, as in:

  ```
  "CC", @IF (YY > 70, 19, 20)
  ```

  This causes the century to be set to 19 when the year is greater than 70; otherwise the century is set to 20.

- The system can calculate the century automatically. If the year is less than 50, the system calculates a century of 20; otherwise, a century of 19 is calculated.

Example 2 The following converts year, month and day columns into a date.

```
date_col = @DATE ("YYYY-MM-DD", "YY", date1_yy, "MM", date1_mm, "DD",
date1_dd)
```

Example 3 The following converts a date and time, defaulting seconds to zero.

```
date_col = @DATE ("YYYY-MM-DD:HH:MI:00", "YYMMDD", date1, "HHMI", time1)
```

Example 4 The following converts a numeric column stored as YYYYMMDDHHMISS to a SQL date.

```
datetime_col = @DATE ("YYYY-MM-DD:HH:MI:SS", "YYYYMMDDHHMISS", numeric_date)
```

Example 5 The following converts a numeric column stored as YYYYMMDDHHMISS to a Julian timestamp.

```
julian_ts_col = @DATE ("JTS", "YYYYMMDDHHMISS", numeric_date)
```

Example 6 The following converts a Julian timestamp column to two separate columns: a datetime column in the format YYYY-MM-DD:HH:MI:SS and a fraction column that holds the microseconds portion of the timestamp.

```
datetime_col = @DATE ("YYYY-MM-DD:HH:MI:SS", "JTS", jts_field), fraction_col
= @DATE ("FFFFFF", "JTS", jts_field)
```

Example 7 The following produces the time at which an order is filled. The inner @DATE expression changes the order_taken column into a Julian timestamp, then adds the order_minutes column converted into microseconds to this timestamp. The expression is passed back as a new Julian timestamp to the outer @DATE expression, which converts it back to a more readable date and time.

```
order_filled = @DATE ("YYYY-MM-DD:HH:MI:SS", "JTS", @DATE ("JTS",
"YYMMDDHHMISS", order_taken) + order_minutes * 60 * 1000000)
```

Example 8 The following does a full calculation of times. It goes from a source date column (named "dt") to a target column (named "dt5") that is to be converted to the date + 5 hours. The calculation also goes from a source timestamp column (named "ts") to a target column (named "ts5") that is to be converted to the timestamp + 5 hours.

```
MAP scratch.t4, TARGET scratch.t4_copy,
COLMAP ( USEDEFAULTS,
dt5 = @DATE ("YYYY-MM-DD:HH:MI:SS", "JTS",
@COMPUTE (@DATE ("JTS", "YYYY-MM-DD:HH:MI:SS", dt) + 18000000000 ) ),
ts5 = @DATE ("YYYY-MM-DD:HH:MI:SS.FFFFFF", "JTS",
@COMPUTE ( @DATE ("JTS", "YYYY-MM-DD:HH:MI:SS.FFFFFF", ts) + 18000000000 ) )
) ;
```

DATEDIFF

Use the @DATEDIFF function to calculate the difference between two dates or datetimes, in days or seconds.

Syntax

```
@DATEDIFF ("difference", “<date>”, “<date>”)
```

Argument	Description
`<difference>`	The difference between the specified dates. Valid values can be: `DD` Computes the difference in days. `SS` Computes the difference in seconds.
`<date>`	A string within quotes, in the format of YYYY-MM-DD[*HH:MI[:SS]], where * can be a colon (:) or a blank space, or the @DATENOW function without quotes to return the current date.

Example 1 The following calculates the number of days since the beginning of the year 2011.

```
YTD = @DATEDIFF ("DD", "2011-01-01", @DATENOW ())
```

Example 2 The following calculates the numerical day of the year. (@DATEDIFF returns 0 for 2011-01-01):

```
todays_day = @COMPUTE (@DATEDIFF ("DD", "2011-01-01", @DATENOW ()) +1)
```

DATENOW

Use the @DATENOW function to return the current date and time in the format YYYY-MM-DD HH:MI:SS. The date and time are returned in local time, including adjustments for Daylight Saving Time. @DATENOW takes no arguments.

Syntax

```
@DATENOW ()
```

DDL

Use the @DDL function to return information about a DDL operation.

Syntax

```
@DDL ({TEXT | OPTYPE | OBJNAME | OBJTYPE | OBJOWNER})
```

Argument	Description
`OBJNAME`	Returns the name of the object that is affected by the DDL.
`OBJOWNER`	Returns the name of the owner of the object that is affected by the DDL.
`OBJTYPE`	Returns the type of object that is affected by the DDL, such as TABLE or INDEX)

Argument	Description
`OPTYPE`	Returns the operation type of the DDL, such as CREATE or ALTER.
`TEXT`	Returns the first 200 characters of the text of the DDL statement.

Example The following example uses the output from @DDL in an EVENTACTIONS shell command.

```
DDL INCLUDE OBJNAME src.t* &
EVENTACTIONS (SHELL ("echo The DDL text is var1> out.txt ", &
VAR var1 = DDL(TEXT));
```

The redirected output file might contain a string like this:

```
"The DDL text is CREATE TABLE src.test_tab (col1 int);"
```

EVAL

Use the @EVAL function to select a value based on a series of independent tests. There is no limit to the number of conditions you can test. If the number of cases is large, list the most frequently encountered conditions first for best performance.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@EVAL (<condition1>, <result1>
[<condition2>, <result2>] [, ....]
[, <default result>])
```

Argument	Description
`<condition>`	A conditional test using standard conditional operators.
`<result>`	A value or string to return based on the results of the conditional test. Enclose literals within double quotes.
`<default result>`	A default result to return if none of the conditions is satisfied. A default result is optional.

Example 1 In the following example, if the AMOUNT column is greater than 10000, “high amount” is returned. If AMOUNT is greater than 5000 (and less than or equal to 10000), “somewhat high” is returned (unless the prior condition was satisfied). If neither condition is satisfied, a COLUMN_MISSING indication is returned because a default result is not specified.

```
AMOUNT_DESC = @EVAL (AMOUNT > 10000, “high amount”, AMOUNT > 5000, “somewhat
high” )
```

Example 2 The following is a modification of the preceding example. It returns the same results, except that a default value is specified, and a result of “lower” is returned if AMOUNT is less than or equal to 5000.

```
@EVAL (AMOUNT > 10000, “high amount”, AMOUNT > 5000, “somewhat high”, “lower”)
```

GETENV

Use the @GETENV function to return information about the Oracle GoldenGate environment. You can use the information as input into the following:

- Stored procedures or queries (with SQLEXEC)
- Column maps (with the COLMAP option of TABLE or MAP)
- User tokens (defined with the TOKENS option of TABLE and mapped to target columns by means of the @TOKENS function)
- The GET_ENV_VALUE user exit function (see page 505)

Table 50. Overview of GETENV options

Information type	Description
General information types	
`(“LAG” , “<unit>”)`	Returns lag information.
`(“LASTERR” , ”<option>”)`	Returns information about the last replicated operation, including detailed error information.
`(“JULIANTIMESTAMP”)`	Returns the current system time in Julian format.
`(“RECSOUTPUT”)`	Returns a count of the number of records that Extract has written to the trail file since the process started.
Table-level statistics information types	
`("STATS", ["TABLE", "<table_name"], "<operation_type">)`	Returns statistics for DML and DDL operations for one or more specified tables.
`("DELTASTATS", ["TABLE", "<Table_Name"], "<operation_type">)`	Returns statistics for DML and DDL operations as a delta value for one or more specified tables.
Oracle GoldenGate information types	
`(“GGENVIRONMENT”, “<option>”)`	Returns Oracle GoldenGate environment information.
`(“GGFILEHEADER”, “<option>”)`	Returns the format and properties of an Oracle GoldenGate trail file, which is stored in the file header.
`(“GGHEADER”, “<option>”)`	Returns Oracle GoldenGate record header information.
`(“RECORD”, “<option>”)`	Returns the sequence number and RBA to indicate the location of a record in an Oracle GoldenGate trail file.
Database information types	
`(“DBENVIRONMENT”, “<option>”)`	Returns global database environment information.

Table 50. Overview of GETENV options (continued)

Information type	Description
`(“TRANSACTION”, “<option>”)`	Returns information about a source transaction.
Operating system information type	
`(“OSVARIABLE”, “<variable>”)`	Returns the string value of a specified operating-system environment variable.
Base 24 information types	

`("TLFKEY", "SYSKEY"`
`“<unique key>”)`
	Enables a unique key to be associated with TLF/PTLF records in ACI's Base24 application.

Example The following example uses the @GETENV function in a TOKENS clause of a TABLE statement to populate user tokens within the Oracle GoldenGate record header. It demonstrates how several of the function’s options can be combined to return specific information.

```
TABLE fin.product, TOKENS (
TKN-OSUSER = @GETENV ("GGENVIRONMENT", "OSUSERNAME"),
TKN-DOMAIN = @GETENV ("GGENVIRONMENT", "DOMAINNAME"),
TKN-COMMIT-TS = @GETENV ("GGHEADER", "COMMITTIMESTAMP"),
TKN-TABLE = @GETENV ("GGHEADER", "TABLENAME"),
TKN-OP-TYPE = @GETENV ("GGHEADER", "OPTYPE"),
TKN-LENGTH = @GETENV ("GGHEADER", "RECORDLENGTH"),
TKN-LAG-SEC = @GETENV ("LAG", "SECONDS"),
TKN-DB-USER = @GETENV ("DBENVIRONMENT", "DBUSER"),
TKN-DB-VER = @GETENV ("DBENVIRONMENT", "DBVERSION"),
TKN-ROWID = @GETENV ("RECORD", "GDVN"));
```

Using the LAG information type

Use the LAG option of @GETENV to return lag information. Lag is the difference between the time that a record was processed by Extract or Replicat and the timestamp of that record in the data source. Both LAG and <environment value> must be enclosed within double quotes.

Syntax

```
@GETENV (“LAG”, “<unit>”)
```

Environment value	Return value
`“SEC”`	Returns the lag in seconds. This is the default when a unit is not explicitly provided for LAG.
`“MSEC”`	Returns the lag in milliseconds.
`“MIN”`	Returns the lag in minutes.

Using the LASTERR information type

Use the LASTERR option of @GETENV to return information about the last failed operation processed by Replicat. The options provide error information. LASTERR is valid for use with the Replicat process only. Both LASTERR and <environment value> must be enclosed within double quotes.

Syntax

```
@GETENV (“LASTERR”, ”<return value>”)
```

Environment value	Return value
“DBERRNUM”	Returns the database error number associated with the failed operation.
“DBERRMSG”	Returns the database error message associated with the failed operation.
“OPTYPE”	Returns the operation type that was attempted. For a list of Oracle GoldenGate operation types, see the appendix section of the Oracle GoldenGate *Windows and UNIX Administrator’s Guide.*
“ERRTYPE”	Returns the type of error. Possible results are: ◆ DB (for database errors) ◆ MAP (for errors in mapping)

Using the JULIANTIMESTAMP information type

Use the JULIANTIMESTAMP option of @GETENV to return the current time in Julian format. The unit is microseconds (one millionth of a second). On a Windows machine, the value will be padded with zeros (0) because the granularity of the Windows timestamp is milliseconds (one thousandth of a second). For example, the following is a typical column mapping:

```
MAP dbo.tab8451, Target targ.tabjts, COLMAP (USEDEFAULTS, &
JTSS = @GETENV ("JULIANTIMESTAMP")
JTSFFFFFF = @date ("yyyy-mm-dd:hh:mi:ss.ffffff", "JTS", &
@getenv ("JULIANTIMESTAMP") ) )
;
```

Possible values that the JTSS and JTSFFFFFF columns can have are:

```
212096320960773000 2010-12-17:16:42:40.773000
212096321536540000 2010-12-17:16:52:16.540000
212096322856385000 2010-12-17:17:14:16.385000
212096323062919000 2010-12-17:17:17:42.919000
212096380852787000 2010-12-18:09:20:52.787000
```

The last three digits (the microseconds) of the number all contain the padding of 0s .

Syntax

```
@GETENV (“JULIANTIMESTAMP”)
```

Using the RECSOUTPUT information type

Use the RECSOUTPUT option of @GETENV to retrieve a current count of the number of records that Extract has written to the trail file since the process started. The returned value is not unique to a table or transaction, but instead for the Extract session itself. The count resets to 1 whenever Extract stops and then is started again.

Syntax

```
@GETENV (“RECSOUTPUT”)
```

Using the STATS and DELTASTATS information types

Use the STATS and DELTASTATS options of @GETENV to return the number of operations that were processed, per table, by Extract or Replicat for any or all of the following:

- INSERT operations
- UPDATE operations
- DELETE operations
- TRUNCATE operations
- Total DML operations
- Total DDL operations
- Number of conflicts that occurred, if the Conflict Detection and Resolution (CDR) feature is used.
- Number of CDR resolutions that succeeded
- Number of CDR resolutions that failed

STATS returns counts since process startup, whereas DELTASTATS returns counts since the last execution of a DELTASTATS.

The execution logic is as follows:

- When Extract processes a transaction record that satisfies @GETENV with STATS or DELTASTATS, the table name is matched against resolved source tables in the TABLE statement.
- When Replicat processes a trail record that satisfies @GETENV with STATS or DELTASTATS, the table name is matched against resolved target tables in the TARGET clause of the MAP statement.

All input elements in the @GETENV syntax must be enclosed within double quotes.

Any errors in the processing of this function, such as an unresolved table entry or incorrect syntax, returns a zero (0) for the requested statistics value.

Constraining the output with “TABLE”

The output of these functions can be constrained to a given table or tables; otherwise, the operation count is returned for all tables in all TABLE or MAP statements, depending on the process in which @GETENV executes.

For example, the following counts DML operations only for tables in the “hr” schema:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = @GETENV ("STATS",
"TABLE", "hr.*", "DML"));
```

Likewise, the following counts DML operations only for the “emp” table in the “hr” schema:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = @GETENV ("STATS",
"TABLE", "hr.emp", "DML"));
```

By contrast, because there are no specific tables specified for STATS in the following example, the function counts all INSERT, UPDATE, and DELETE operations for all tables in all schemas that are represented in the TARGET clauses of MAP statements:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = &
@GETENV ("STATS", "DML"));
```

Understanding how recurring table specifications affect operation counts

An Extract that is processing the same source table to multiple output trails returns statistics based on each localized output trail to which the table linked to @GETENV is written. For example, if Extract captures 100 inserts for table “ABC ” and writes table ABC to three trails, the result for the @GETENV is 300

```
EXTRACT ABC
...
EXTTRAIL c:\ogg\dirdat\aa;
TABLE TEST.ABC;
EXTTRAIL c:\ogg\dirdat\bb;
TABLE TEST.ABC;
TABLE EMI, TOKENS (TOKEN-CNT = @GETENV ("STATS", "TABLE", "ABC", "DML"));
EXTTRAIL c:\ogg\dirdat\cc;
TABLE TEST.ABC;
```

In the case of an Extract that writes a source table multiple times to a single output trail, or in the case of a Replicat that has multiple MAP statements for the same TARGET table, the statistics results are based on all matching TARGET entries. For example, if Replicat filters 20 rows for REGION “WEST,” 10 rows for REGION “EAST,” 5 rows for REGION “NORTH,” and 2 rows for REGION “SOUTH,” all for table “ABC,” the result of the @GETENV is 37.

```
REPLICAT ABC
...
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, "WEST"));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, "EAST"));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, "NORTH"));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, "SOUTH"));
MAP TEST.EMI, TARGET TEST.EMI, &
   COLMAP (CNT = @GETENV ("STATS", "TABLE", "ABC", "DML"));
```

Capturing multiple statistics

You can execute multiple instances of @GETENV to get counts for different operation types.

This example returns statistics only for INSERT and UPDATE operations:

```
REPLICAT TEST
..
..
MAP TEST.ABC, TARGET TEST.ABC, COLMAP (USEDEFAULTS, IU = @COMPUTE(@GETENV &
   ("STATS", "TABLE", "ABC", "DML") - (@GETENV ("STATS", "TABLE", &
   "ABC", "DELETE"));
```

This example returns statistics for DDL and TRUNCATE operations:

```
REPLICAT TEST2
..
..
MAP TEST.ABC, TARGET TEST.ABC, COLMAP (USEDEFAULTS, DDL = @COMPUTE &
(@GETENV ("STATS", "DDL") + (@GETENV ("STATS", "TRUNCATE"));
```

Example use case

In the following use case, if all DML from the source is applied successfully to the target, Replicat suspends by means of EVENTACTIONS with SUSPEND, until resumed from GGSCI with SEND REPLICAT with RESUME.

GETENV used in Extract parameter file:

```
TABLE HR1.HR*;
TABLE HR1.STAT, TOKENS ("env_stats" = @GETENV("STATS", "TABLE", &
   "HR1.HR*", "DML"));
```

GETENV used in Replicat parameter file:

```
MAP HR1.HR*, TARGET HR2.*;
MAP HR1.STAT, TARGET HR2.STAT, filter (
   @if (
   @token ("stats") =
   @getenv("STATS", "TABLE", "TSSCAT.TCUSTORD", "DML"), 1, 0 )
   ),
   eventactions (suspend);
```

Syntax

```
@GETENV ({"STATS" | “DELTASTATS”}, ["TABLE", "<table_name"],
"<statistic_type">)
```

Input value	Return value
"TABLE", "<table_name>"	Optional, executes the STATS or DELTASTATS only for the specified table or tables. Without this option, counts are returned for all tables that are specified in TABLE (Extract) or MAP (Replicat) parameters in the parameter file. Valid <table_name> values are: ◆ "<schema>.<table>" specifies a specific table. ◆ "<table>" specifies a specific table of the default schema. ◆ "<schema>.*" specifies all tables of a schema. ◆ "*" specifies all tables of the default schema.
"<statistic_type>" can be one of:	
"INSERT"	Returns the number of INSERT operations that were processed.
"UPDATE"	Returns the number of UPDATE operations that were processed.
"DELETE"	Returns the number of DELETE operations that were processed.
"DML"	Returns the total of INSERT, UPDATE, and DELETE operations that were processed.
"TRUNCATE"	Returns the number of TRUNCATE operations that were processed. This variable returns a count only if Oracle GoldenGate DDL replication is not being used. If DDL replication is being used, this variable returns a zero.
"DDL"	Returns the number of DDL operations that were processed, including TRUNCATEs and DDL specified in INCLUDE and EXCLUDE clauses of the DDL parameter, all scopes (MAPPED, UNMAPPED, OTHER). This variable returns a count only if Oracle GoldenGate DDL replication is being used. This variable is not valid for "DELTASTATS".
CDR_CONFLICTS	Returns the number of conflicts that Replicat detected when executing the Conflict Detection and Resolution (CDR) feature. Example for a specific table: @GETENV("STATS","TABLE","HR.EMP","CDR_CONFLICTS") Example for all tables processed by Replicat: @GETENV("STATS","CDR_CONFLICTS")

Input value	Return value

`CDR_RESOLUTIONS_SUCCEEDED`	Returns the number of conflicts that Replicat resolved when executing the Conflict Detection and Resolution (CDR) feature.
	Example for a specific table:
	`@GETENV("STATS","TABLE","HR.EMP", "CDR_RESOLUTIONS_SUCCEEDED")`
	Example for all tables processed by Replicat:
	`@GETENV("STATS","CDR_RESOLUTIONS_SUCCEEDED")`

`CDR_RESOLUTIONS_FAILED`	Returns the number of conflicts that Replicat could not resolve when executing the Conflict Detection and Resolution (CDR) feature.
	Example for a specific table:
	`@GETENV("STATS","TABLE","HR.EMP", "CDR_RESOLUTIONS_FAILED")`
	Example for all tables processed by Replicat:
	`@GETENV("STATS","CDR_RESOLUTIONS_FAILED")`

Using the GGENVIRONMENT information type

Use the GGENVIRONMENT option of @GETENV to return information about the Oracle GoldenGate environment. This option is valid for the Extract and Replicat processes. Both GGSENVIRONMENT and <environment value> must be enclosed within double quotes.

Syntax

```
@GETENV (“GGENVIRONMENT”, “<return value>”)
```

Environment value	Return value
`“DOMAINNAME”`	(Windows only) Returns the domain name associated with the user that started the process.
`“GROUPDESCRIPTION”`	The description of the group, taken from the checkpoint file if a description was provided with the DESCRIPTION parameter when creating the group with the ADD command in GGSCI.
`“GROUPNAME”`	Returns the name of the process group.
`“GROUPTYPE”`	Returns the type of process, either EXTRACT or REPLICAT.
`“HOSTNAME”`	Returns the name of the system running the Extract or Replicat process.
`“OSUSERNAME”`	Returns the operating system user name that started the process.
`“PROCESSID”`	The process ID that is assigned to the process by the operating system.

Using the GGHEADER information type

Use the GGHEADER option of @GETENV to return information from the header portion of an Oracle GoldenGate trail record. Every data record within the Oracle GoldenGate trail contains a header, which describes the transaction environment of the record. For more information on record headers, see the Oracle GoldenGate *Windows and UNIX Reference Guide*.

This option is valid for the Extract and Replicat processes. Both GGHEADER and <environment value> must be enclosed within double quotes.

Syntax

```
@GETENV (“GGHEADER”, “<return value>”)
```

Environment value	Return value
`“BEFOREAFTERINDICATOR”`	Returns the before or after indicator showing whether the record is a before image or an after image. Possible results are: ◆ BEFORE (before image) ◆ AFTER (after image)
`“COMMITTIMESTAMP”`	Returns the transaction timestamp (the time when the transaction committed) expressed in the format of YYYY-MM-DD HH:MI:SS.FFFFFF, for example: `2011-01-24 17:08:59.000000`
`“LOGPOSITION”`	Returns the position of the Extract process in the data source. (See the LOGRBA option.)
`“LOGRBA”`	LOGRBA and LOGPOSITION store details of the position in the data source of the record. For transactional log-based products, LOGRBA is the sequence number and LOGPOSITION is the relative byte address. However, these values will vary depending on the capture method and database type.

`“OBJECTNAME” |`
`“TABLENAME”`
	Returns the table name or object name (if a sequence).

`“OPTYPE”`	Returns the type of operation. Possible results are: INSERT UPDATE DELETE ENSCRIBE COMPUPDATE SQL COMPUPDATE PK UPDATE TRUNCATE If the operation is not one of the above types, then the function returns the word TYPE with the number assigned to the type. For more information about possible record types, see Appendix 1 in the Oracle GoldenGate *Windows and UNIX Administrator’s Guide*.

Environment value	Return value
"RECORDLENGTH"	Returns the record length in bytes.
"TRANSACTIONINDICATOR"	Returns the transaction indicator. The value corresponds to the TransInd field of the record header, which can be viewed with the Logdump utility (see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*). Possible results are: ◆ BEGIN (represents TransInD of 0, the first record of a transaction.) ◆ MIDDLE (represents TransInD of 1, a record in the middle of a transaction.) ◆ END (represents TransInD of 2, the last record of a transaction.) ◆ WHOLE (represents TransInD of 3, the only record in a transaction.)

Using the GGFILEHEADER information type

Use the GGFILEHEADER option of @GETENV to return attributes of an Oracle GoldenGate extract file or trail file that are stored in the file header. Every file in a trail contains this header. The header describes the file itself and the environment in which it is used.

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.

This option is valid for the Replicat process. Both GGFILEHEADER and <environment value> must be enclosed within double quotes.

NOTE If a given database, operating system, or Oracle GoldenGate version does not provide information that relates to a given token, a NULL value will be returned.

Syntax

```
@GETENV ("GGFILEHEADER", "<return_value>")
```

Environment value	Return value
"COMPATIBILITY"	Returns the Oracle GoldenGate compatibility level of the trail file. The compatibility level of the current Oracle GoldenGate version must be greater than, or equal to, the compatibility level of the trail file to be able to read the data records in that file. Current valid values are 0 or 1. ◆ 1 means that the trail file is of Oracle GoldenGate version 10.0 or later, which supports file headers that contain file versioning information. ◆ 0 means that the trail file is of an Oracle GoldenGate version that is older than 10.0. File headers are not supported in those releases. The 0 value is used for backward compatibility to those Oracle GoldenGate versions.
Information about the trail file	
"CHARSET"	Returns the global character set of the trail file. For example: WCP1252-1
"CREATETIMESTAMP"	Returns the time that the trail was created, in local GMT Julian time in INT64.
"FILENAME"	Returns the name of the trail file. Can be an absolute or relative path, with a forward or backward slash depending on the filesystem.
"FILEISTRAIL"	Returns a True/false flag indicating whether the trail file is a single file (such as one created for a batch run) or a sequentially numbered file that is part of a trail for online, continuous processing. If false, the SeqNum subtoken is not valid.
"FILESEQNO"	Returns the sequence number of the trail file, without any leading zeros. For example, if a file sequence number is aa000026, FILESEQNO returns 26.
"FILESIZE"	Returns the size of the trail file. It returns NULL on an active file and returns a size value when the file is full and the trail rolls over.
"FIRSTRECCSN"	Returns the commit sequence number (CSN) of the first record in the trail file.Value is NULL until the trail file is completed. For more information about the CSN, see Appendix 1 on page 559.
"LASTRECCSN"	Returns the commit sequence number (CSN) of the last record in the trail file.Value is NULL until the trail file is completed. For more information about the CSN, see Appendix 1 on page 559.
"FIRSTRECIOTIME"	Returns the time that the first record was written to the trail file. Value is NULL until the trail file is completed.

Environment value	Return value
“LASTRECIOTIME”	Returns the time that the last record was written to the trail file. Value is NULL until the trail file is completed.
“URI”	Returns the universal resource identifier of the process that created the trail file, in the format of: <host_name>:<dir>:[:<dir>][:<dir_n>]<group_name> **Where:** ◆ host_name is the name of the server that hosts the process ◆ dir is a subdirectory of the Oracle GoldenGate installation path. ◆ group_name is the name of the process group that is linked with the process. **Example**: sys1:home:oracle:v9.5:extora Shows where the trail was processed and by which process. This includes a history of previous runs.
“URIHISTORY”	Returns a list of the URIs of processes that wrote to the trail file before the current process. ◆ For a primary Extract, this field is empty. ◆ For a data pump, this field is URIHistory + URI of the input trail file.
Information about the Oracle GoldenGate process that created the trail file	
“GROUPNAME”	Returns the name of the group that is associated with the Extract process that created the trail. The group name is that which was given in the ADD EXTRACT command. For example, “ggext.”
“DATASOURCE”	Returns the data source that was read by the process. Can be one of: ◆ DS_EXTRACT_TRAILS (source was an Oracle GoldenGate extract file, populated with change data) ◆ DS_DATABASE (source was a direct select from database table written to a trail, used for SOURCEISTABLE-driven initial load) ◆ DS_TRAN_LOGS (source was the database transaction log) ◆ DS_INITIAL_DATA_LOAD (source was Extract; data taken directly from source tables) ◆ DS_VAM_EXTRACT (source was a vendor access module) ◆ DS_VAM_TWO_PHASE_COMMIT (source was a VAM trail)
“GGMAJORVERSION”	Returns the major version of the Extract process that created the trail, expressed as an integer. For example, if a version is 1.2.3, it returns 1.
“GGMINORVERSION”	Returns the minor version of the Extract process that created the trail, expressed as an integer. For example, if a version is 1.2.3, it returns 2.

Environment value	Return value
“GGVERSIONSTRING”	Returns the maintenance (or patch) level of the Extract process that created the trail, expressed as an integer. For example, if a version is 1.2.3, it returns 3.
“GGMAINTENANCELEVEL”	Returns the maintenance version of the process (xx.xx.xx).
“GGBUGFIXLEVEL”	Returns the patch version of the process (xx.xx.xx.xx).
“GGBUILDNUMBER”	Returns the build number of the process.
Information about the local host of the trail file	
“HOSTNAME”	Returns the DNS name of the machine where the Extract that wrote the trail is running. For example: ◆ `sysa` ◆ `sysb` ◆ `paris` ◆ `hq25`
“OSVERSION”	Returns the major version of the operating system of the machine where the Extract that wrote the trail is running. For example: ◆ `Versions10_69` ◆ `#1 SMP Fri Feb 24 16:56:28 EST 2006` ◆ `5.00.2195 Service Pack 4`
“OSRELEASE”	Returns the release version of the operating system of the machine where the Extract that wrote the trail is running. For example, release versions of the examples given for OSVERSION could be: ◆ `5.10` ◆ `2.6.9-34.ELsmp`
“OSTYPE”	Returns the type of operating system of the machine where the Extract that wrote the trail is running. For example: ◆ `SunOS` ◆ `Linux` ◆ `Microsoft Windows`
“HARDWARETYPE”	Returns the type of hardware of the machine where the Extract that wrote the trail is running. For example: ◆ `sun4u` ◆ `x86_64` ◆ `x86`
Information about the database that produced the data in the trail file.	
“DBNAME”	Returns the name of the database, for example findb.

Environment value	Return value
“DBINSTANCE”	Returns the name of the database instance, if applicable to the database type, for example ORA1022A.
“DBTYPE”	Returns the type of database that produced the data in the trail file. Can be one of: DB2 UDB DB2 ZOS CTREE MSSQL MYSQL ORACLE SQLMX SYBASE TERADATA NONSTOP ENSCRIBE ODBC
“DBCHARSET”	Returns the character set that is used by the database that produced the data in the trail file. (For some databases, this will be empty.)
“DBMAJORVERSION”	Returns the major version of the database that produced the data in the trail file.
“DBMINORVERSION”	Returns the minor version of the database that produced the data in the trail file.
“DBVERSIONSTRING”	Returns the maintenance (patch) level of the database that produced the data in the trail file.
“DBCLIENTCHARSET”	Returns the character set that is used by the database client.
“DBCLIENTVERSIONSTRING”	Returns the maintenance (patch) level of the database client. (For some databases, this will be empty.)
Recovery information carried over from the previous trail file.	
“RECOVERYMODE”	Returns recovery information for internal Oracle GoldenGate use.
“LASTCOMPLETECSN”	Returns recovery information for internal Oracle GoldenGate use.
“LASTCOMPLETEXIDS”	Returns recovery information for internal Oracle GoldenGate use.
“LASTCSN”	Returns recovery information for internal Oracle GoldenGate use.
“LASTXID”	Returns recovery information for internal Oracle GoldenGate use.
“LASTCSNTS”	Returns recovery information for internal Oracle GoldenGate use.

Using the RECORD information type

Use the RECORD option of @GETENV to return the location of a record in an Oracle GoldenGate trail file. This function can return the sequence number of the file and the relative byte address within that file. Together, these values provide a unique value that can be associated with a given record.

This option is valid for an Extract data pump or a Replicat process. Both RECORD and <environment value> must be enclosed within double quotes.

Syntax

```
@GETENV (“RECORD”, “<environment value>”)
```

Environment value	Return value
“FILESEQNO”	Returns the sequence number of the trail file without any leading zeros.
“FILERBA”	Returns the relative byte address of the record within the FILESEQNO file.
“RSN”	Returns the record sequence number within the transaction.
TIMESTAMP	Returns the timestamp of the record.

Using the DBENVIRONMENT information type

Use the DBENVIRONMENT option of @GETENV to return global environment information for a database. This option is valid for the Extract and Replicat processes. Both DBENVIRONMENT and <environment value> must be enclosed within double quotes.

Syntax

```
@GETENV (“DBENVIRONMENT”, “<return value>”)
```

Environment value	Return value
“DBNAME”	Returns the database name.
“DBVERSION”	Returns the database version.
"DBUSER"	Returns the database login user. *Note*: Microsoft SQL Server does not log the user ID.
“SERVERNAME”	Returns the name of the server.

Using the TRANSACTION information type

Use the TRANSACTION option of @GETENV to return information about a source transaction. This option is valid for the Extract process. Both TRANSACTION and <environment value> must be enclosed within double quotes.

Syntax

```
@GETENV (“TRANSACTION”, “<return value>”)
```

Environment value	Return value
`"TRANSACTIONID" | "XID"`	Returns the transaction ID number. The transaction ID and the CSN are associated with the first record of every transaction and are stored as tokens in the trail record. For each transaction ID, there is an associated CSN. Transaction ID tokens have no zero-padding on any platform, because they never get evaluated as relative values. They only get evaluated for whether they match or do not match. Note that in the trail, the transaction ID token is shown as TRANID.
`"CSN"`	Returns the commit sequence number (CSN). The CSN is not zero-padded when returned for these databases: Oracle, DB2 LUW, and DB2 z/OS. For all other supported databases, the CSN is zero-padded. In the case of the Sybase CSN, each substring that is delimited by a dot (.) will be padded to a length that does not change for that substring. Note that in the trail, the CSN token is shown as LOGCSN. See the TRANSACTIONID | XID environment value for additional information about the CSN token. For more information about the CSN itself, see Appendix 1 on page 559.
`"TIMESTAMP"`	Returns the commit timestamp of the transaction.
`"NAME"`	Returns the transaction name, if available.
`"USERID"`	(Oracle) Returns the Oracle user-id of the database user that committed the last transaction.
`"USERNAME"`	(Oracle) Returns the Oracle user-name of the database user that committed the last transaction.
`"PLANNAME"`	(DB2 on z/OS) Returns the plan name under which the current transaction was originally executed. The plan name is included in the begin unit of recovery log record.

Using the OSVARIABLE information type

Use the OSVARIABLE option of @GETENV to return the string value of a specified operating-system environment variable. This option is valid for Extract and Replicat. Both OSVARIABLE and <variable> must be within double quotes.

Syntax

```
@GETENV ("OSVARIABLE", "<variable>")
```

Environment value	Return value

`“<variable>”`	The name of the variable. The search is an exact match of the supplied variable name. For example, the UNIX grep command would return all of the following variables, but @GETENV ("OSVARIABLE", "HOME") would only return the value for HOME:
	`ANT_HOME=/usr/local/ant`
	`JAVA_HOME=/usr/java/j2sdk1.4.2_10`
	`HOME=/home/judyd`
	`ORACLE_HOME=/rdbms/oracle/ora1022i/64`
	The search is case-sensitive if the operating system supports case-sensitivity.

Using the TLFKEY information type

Use the TLFKEY option of @GETENV to associate a unique key with TLF/PTLF records in ACI's Base24 application. The 64-bit key is composed of the following concatenated items:

- The number of seconds since 2000.
- The block number of the record in the TLF/PTLF block multiplied by ten.
- The node specified by the user (must be between 0 and 255).

This option is valid for the Extract and Replicat processes. TLFKEY must be within double quotes.

Syntax

```
@GETENV ("TLFKEY", SYSKEY <unique key>)
```

Environment value	Return value
`<unique key>`	The NonStop node number of the source TLF/PTLF file. Example: `GETENV ("TLFKEY", SYSKEY, 27)`

GETVAL

Use the @GETVAL function to extract values from a stored procedure or query so that they can be used as input to a FILTER or COLMAP clause of a MAP or TABLE statement.

Whether or not a parameter value can be extracted with @GETVAL depends upon the following:

1. Whether or not the stored procedure or query executed successfully.
2. Whether or not the stored procedure or query results have expired.

Handling missing column values

When a value cannot be extracted, the @GETVAL function results in a “column missing” condition. Typically, this occurs for update operations if the database only logs values for columns that were changed.

Usually this means that the column cannot be mapped. To test for missing column values, use the @COLTEST function to test the result of @GETVAL, and then map an alternative value for the column to compensate for missing values, if desired. Or, to ensure that column values are available, you can use the FETCHCOLS or FETCHCOLSEXCEPT option of the TABLE or MAP parameter to fetch the values from the database if they are not present in the log. Enabling supplemental logging for the necessary columns also would work.

Syntax

```
@GETVAL (<name>.<parameter>)
```

Argument	Description
`<name>`	The name of the stored procedure or query. When using SQLEXEC to execute the procedure or query, valid values are as follows: Queries: Use the logical name specified with the ID option of the SQLEXEC clause. ID is a required SQLEXEC argument for queries. Stored procedures: Use one of the following, depending on how many times the procedure is to be executed within a TABLE or MAP statement: ◆ For multiple executions, use the logical name defined by the ID clause of the SQLEXEC statement. ID is required for multiple executions of a procedure. ◆ For a single execution, use the actual stored procedure name.
`<parameter>`	Valid values are one of the following. ◆ The name of the parameter in the stored procedure or query from which the data will be extracted and passed to the column map. ◆ RETURN_VALUE, if extracting values returned by a stored procedure or query.

Example 1 The following enables each map statement to call the stored procedure lookup by referencing the logical names lookup1 and lookup2 within the @GETVAL function and refer appropriately to each set of results.

```
MAP schema.srctab, TARGET schema.targtab,
SQLEXEC (SPNAME lookup, ID lookup1, PARAMS (param1 = srccol)),
COLMAP (targcol1 = @GETVAL (lookup1.param2));
MAP schema.srctab, TARGET schema.targtab2,
SQLEXEC (SPNAME lookup, ID lookup2, PARAMS (param1 = srccol)),
COLMAP (targcol2= @GETVAL (lookup2.param2));
```

Example 2 The following shows a single execution of the stored procedure lookup. In this case, the actual name of the procedure is used. A logical name is not needed.

```
MAP schema.tab1, TARGET schema.tab2,
SQLEXEC (SPNAME lookup, PARAMS (param1 = srccol)),
COLMAP (targcol = @GETVAL (lookup.param1));
```

Example 3 The following shows the execution of a query from which values are mapped with @GETVAL.

```
MAP sales.account, TARGET sales.newacct,
SQLEXEC (ID lookup,
QUERY " select desc_col into 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));
```

Alternate syntax

With SQLEXEC, you can capture parameter results without explicitly using the @GETVAL keyword. Simply refer to the procedure name (or logical name if using a query or multiple instances of a procedure) and parameter in the following format:

Syntax `{<procedure name> | <logical name>}.<parameter>`

Example 1 In the following example, @GETVAL is called implicitly for the phrase proc1.p2 without the @GETVAL keyword.

```
MAP test.tab1, TARGET test.tab2,
SQLEXEC (SPNAME proc1, ID myproc, PARAMS (p1 = sourcecol1)),
COLMAP (targcol1 = proc1.p2);
```

Example 2 In the following example, the @GETVAL function is called implicitly for the phrase lookup.desc_param without the @GETVAL keyword.

```
MAP sales.account, TARGET sales.newacct,
SQLEXEC (ID lookup,
QUERY " select desc_col into desc_param from lookup_table "
" where code_col = :code_param ",
PARAMS (code_param = account_code)),
COLMAP (newacct_id = account_id, newacct_val = lookup.desc_param);
```

HEXTOBIN

Use the @HEXTOBIN function to convert a supplied string of hexadecimal data into raw format.

Syntax `@HEXTOBIN(<data>)`

Argument	Description
`<data>`	The name of the source column, an expression, or a literal string that is enclosed within quotes.

Example `@HEXTOBIN(“414243”)` converts to three bytes: 0x41 0x42 0x43.

HIGHVAL | LOWVAL

Use the @HIGHVAL and @LOWVAL functions when you need to generate a value, but you want to constrain it within an upper or lower limit. These functions emulate the COBOL functions of the same names.

Use @HIGHVAL and @LOWVAL only with string and binary data types. When using them with strings, only @STRNCMP is valid. Using them with decimal or date data types or with SQLEXEC operations can cause errors. DOUBLE data types result in
-1 or 0 (Oracle NUMBER, no precision, no scale).

Syntax

```
@HIGHVAL ([<length>]) | @LOWVAL ([<length>])
```

Argument	Description
`<length>`	Optional. Specifies the binary output length in bytes. The maximum value of <length> is the length of the target column.

Example The following example assumes that the size of the group_level column is 5 bytes.

Function statement	Result
`group_level = @HIGHVAL()`	`{0xFF, 0xFF, 0xFF, 0xFF, 0xFF}`
`group_level = @LOWVAL()`	`{0x00, 0x00, 0x00, 0x00, 0x00}`
`group_level = @HIGHVAL(3)`	`{0xFF, 0xFF, 0xFF}`
`group_level = @LOWVAL(3)`	`{0x00, 0x00, 0x00}`

IF

Use the @IF function to return one of two values, based on a condition. You can use the @IF function with other Oracle GoldenGate functions to begin a conditional argument that tests for one or more exception conditions. You can direct processing based on the results of the test. You can nest @IF statements, if needed.

Syntax

```
@IF (<conditional expression>, <value if non-zero>, <value if zero>)
```

Argument	Description
`<conditional expression>`	A valid conditional expression or Oracle GoldenGate function. Use numeric operators (such as =, > or <) only for numeric comparisons. For character comparisons, use one of the character-comparison functions.
`<value if non-zero>`	Non-zero is considered true.
`<value if zero>`	Zero (0) is considered false .

Example 1 The following returns an amount only if the AMT column is greater than zero; otherwise zero is returned.

```
AMOUNT_COL = @IF (AMT > 0, AMT, 0)
```

Example 2 The following returns WEST if the STATE column is CA, AZ or NV; otherwise it returns EAST.

```
REGION = @IF (@VALONEOF (STATE, "CA", "AZ", "NV"), "WEST", "EAST")
```

Example 3 The following returns the result of the PRICE column multiplied by the QUANTITY column if both columns are greater than 0. Otherwise, the @COLSTAT (NULL) function creates a NULL value in the target column.

```
ORDER_TOTAL = @IF (PRICE > 0 AND QUANTITY > 0, PRICE * QUANTITY,
@COLSTAT (NULL))
```

NUMBIN

Use the @NUMBIN function to convert a binary string of eight or fewer bytes into a number. Use this function when the source column defines a byte stream that actually is a number represented as a string.

Syntax

```
@NUMBIN (<source column>)
```

Argument	Description
`<source column>`	The name of the source column containing the string to be converted.

Example The following combines @NUMBIN and @DATE to transform a 48-bit Tandem column to a 64-bit Julian value for local time.

```
DATE = @DATE ("JTSLCT", "TTS" @NUMBIN (DATE))
```

NUMSTR

Use the @NUMSTR function to convert a string (character) column or value into a number. Use @NUMSTR to do either of the following:

- Map a string (character) to a number.
- Use a string column that contains only numbers in an arithmetic expression.

Syntax

```
@NUMSTR (<input>)
```

Argument	Description
`<input>`	Can be either of the following: ◆ The name of a character column. ◆ A literal string that is enclosed within quotes.

Example

```
PAGE_NUM = @NUMSTR (ALPHA_PAGE_NO)
```

RANGE

Use the @RANGE function to divide the rows of any table across two or more Oracle GoldenGate processes. It can be used to increase the throughput of large and heavily accessed tables and also can be used to divide data into sets for distribution to different destinations. Specify each range in a FILTER clause in a TABLE or MAP statement.

@RANGE is safe and scalable. It preserves data integrity by guaranteeing that the same row will always be processed by the same process group.

@RANGE computes a hash value of the columns specified in the input. If no columns are specified, the KEYCOLS clause of the TABLE or MAP statement is used to determine the columns to hash, if a KEYCOLS clause exists. Otherwise, the primary key columns are used.

Oracle GoldenGate adjusts the total number of ranges to optimize the even distribution across the number of ranges specified.

Because any columns can be specified for this function, rows in tables with relational constraints to one another must be grouped together into the same process or trail to preserve referential integrity.

NOTE Using Extract to calculate the ranges is more efficient than using Replicat. Calculating ranges on the target side requires Replicat to read through the entire trail to find the data that meets each range specification.

Syntax

```
@RANGE (<range>, <total ranges> [, <column>] [, <column>] [, ...])
```

Argument	Description
`<range>`	The range assigned to the specified process or trail. Valid values are 1, 2, 3, and so forth, with the maximum value being the value defined by <total ranges>.
`<total ranges>`	The total number of ranges allocated. For example, to divide data into three groups, use the value 3.
`<column>`	The name of a column on which to base the range allocation. This argument is optional. If not used, Oracle GoldenGate allocates ranges based on the table's primary key.

Example 1 In the following example, the replication workload is split into three ranges (between three 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, 3, ID));
```

(Replicat group 2 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (2, 3, ID));
```

(Replicat group 3 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (3, 3, ID));
```

Example 2 In the following example, one Extract process splits the processing load into two trails. Since no columns were defined on which to base the range calculation, Oracle GoldenGate will use the primary key columns.

```
RMTTRAIL /ggs/dirdat/aa
TABLE fin.account, FILTER (@RANGE (1, 2));
RMTTRAIL /ggs/dirdat/bb
TABLE fin.account, FILTER (@RANGE (2, 2));
```

Example 3 In the following example, two tables have relative operations based on an order_ID column. The order_master table has a key of order_ID, and the order_detail table has a key of order_ID and item_number. Because the key order_ID establishes relativity, it is used in @RANGE filters for both tables to preserve referential integrity. The load is split into two ranges.

(Parameter file #1)

```
MAP sales.order_master, TARGET sales.order_master,
FILTER (@RANGE (1, 2, order_ID));
MAP sales.order_detail, TARGET sales.order_detail,
FILTER (@RANGE (1, 2, order_ID));
```

(Parameter file #2)

```
MAP sales.order_master, TARGET sales.order_master,
FILTER (@RANGE (2, 2, order_ID));
MAP sales.order_detail, TARGET sales.order_detail,
FILTER (@RANGE (2, 2, order_ID));
```

STRCAT

Use the @STRCAT function to concatenate one or more strings or string (character) columns. Enclose literal strings within quotes.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRCAT (<string1>, <string2> [, ...])
```

Argument	Description
`<string1>`	The first column or literal string to be concatenated.
`<string2>`	The next column or literal string to be concatenated.

Example The following creates a phone number from three columns and includes the literal formatting values.

```
PHONE_NO = @STRCAT (AREA_CODE, PREFIX, "-", PHONE)
```

STRCMP

Use the @STRCMP function to compare two character columns or literal strings. Enclose literals within quotes.

@STRCMP returns the following:

- –1 if the first string is less than the second.
- 0 if the strings are equal.
- 1 if the first string is greater than the second.

Trailing spaces are truncated before comparing the strings.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. This function can compare different character data types, such as CHAR and NCHAR.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@STRCMP (<string1>, <string2>)
```

Argument	Description
`<string1>`	The first column or literal string to be compared.
`<string2>`	The second column or literal string to be compared.

Example The following example compares two literal strings and returns 1 because the first string is greater than the second one.

```
@STRNCMP ("JOHNSON", "JONES")
```

STREQ

Use the @STREQ function to determine whether or not two string (character) columns or literal strings are equal. Enclose literals within quotes. @STREQ returns the following:

- 1 (true) if the strings are equal.
- 0 (false) if the strings are not equal.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. This function can compare different character data types, such as CHAR and NCHAR.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@STREQ (<string1>, <string2>)
```

Argument	Description
`<string1>`	The first column or literal string to be compared.
`<string2>`	The second column or literal string to be compared.

Example The following compares the value of the region column to the literal value “EAST”. If region = EAST, the record passes the filter.

```
FILTER (@STREQ (region, “EAST”))
```

You could use @STREQ in a comparison to determine a result, as shown in the following example. If the state is “NY”, the expression returns “East Coast”. Otherwise, it returns “Other”.

```
@IF (@STREQ (state, "NY"), "East Coast", "Other")
```

STREXT

Use the @STREXT function to extract a portion of a string.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STREXT (<string>, <begin position>, <end position>)
```

Argument	Description
`<string>`	The string from which to extract. The string can be either the name of a character column or a literal string. Enclose literals within quotes.
`<begin position>`	The character position at which to begin extracting.
`<end position>`	The character position at which to end extracting. The end position is included in the extraction.

```
The following example uses three @STREXT functions to extract a phone number
into three different columns.
AREA_CODE = @STREXT (PHONE, 1, 3),
PREFIX = @STREXT (PHONE, 4, 6),
PHONE_NO = @STREXT (PHONE, 7, 10)
```

STRFIND

Use the @STRFIND function to determine the position of a string within a string column or else return zero if the string is not found. Optionally, @STRFIND can accept a starting position within the string.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@STRFIND (<string>, “search string>” [, < begin position>])
```

Argument	Description
`<string>`	The string in which to search. This can be either the name of a character column or a literal string. Enclose literals within quotes.
`“<search string>”`	The string for which to search. Enclose the search string within quotes.
`<begin position>`	The character position at which to begin searching.

Example Assuming the string for the ACCT column is ABC123ABC, the following are possible results.

Function statement	Result
`@STRFIND (ACCT, "23")`	5
`@STRFIND (ACCT, "ZZ")`	0
`@STRFIND (ACCT, "ABC", 2)`	7 (because the search started at the second character)

STRLEN

Use the @STRLEN function to return the length of a string, expressed as the number of characters.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@STRLEN (<string>)
```

Argument	Description
`<string>`	The name of a string (character) column or a literal string. Enclose literals within quotes.

Example

```
@STRLEN (ID_NO)
```

STRLTRIM

Use the @STRLTRIM function to trim leading spaces.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRLTRIM (<string>)
```

Argument	Description
`<string>`	The name of a character column or a literal string. Enclose literals within quotes.

Example

```
birth_state = @strltrim (state)
```

STRNCAT

Use the @STRNCAT function to concatenate one or more strings to a maximum length.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRNCAT (<string>, <max length> [, <string>, <max length>] [, ...] )
```

Argument	Description
`<string>`	The name of a string (character) column or a literal string. Enclose literals within quotes.
`<max length>`	The maximum string length, in characters.

Example The following concatenates two strings and results in “ABC123.”

```
PHONE_NO = @STRNCAT ("ABCDEF", 3, "123456", 3)
```

STRNCMP

Use the @STRNCMP function to compare two strings based on a specific number of characters. The string can be either the name of a string (character) column or a literal string that is enclosed within quotes. The comparison starts at the first character in the string.

@STRNCMP returns the following:

- –1 if the first string is less than the second.
- 0 if the strings are equal.
- 1 if the first string is greater than the second.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@STRNCMP (<string1>, <string2>, <max length>)
```

Argument	Description
`<string1>`	The first string to be compared.
`<string2>`	The second string to be compared.
`<max length>`	The maximum number of characters in the string to compare.

Example The following example compares the first two characters of each string, as specified by a <max length> of 2, and it returns 0 because both sets are the same.

```
@STRNCMP ("JOHNSON", "JONES", 2)
```

STRNUM

Use the @STRNUM function to convert a number into a string and specify the output format and padding.

Syntax

```
@STRNUM (<column>, {LEFT | LEFTSPACE, | RIGHT | RIGHTZERO} [<length>] )
```

Argument	Description
`<column>`	The name of a source numeric column.
`LEFT`	Left justify, without padding.
`LEFTSPACE`	Left justify, fill the rest of the target column with spaces.
`RIGHT`	Right justify, fill the rest of the target column with spaces. If the value of a column is a negative value, the spaces are added before the minus sign. For example, strnum(Col1, right) used for a column value of -1.27 becomes ###-1.27, assuming the target column allows 7 digits. The minus sign is not counted as a digit, but the decimal is.

Argument	Description
RIGHTZERO	Right justify, fill the rest of the target column with zeros. If the value of a column is a negative value, the zeros are added after the minus sign and before the numbers. For example, strnum(Col1, rightzero) used for a column value of -1.27 becomes -0001.27, assuming the target column allows 7 digits. The minus sign is not counted as a digit, but the decimal is.
<length>	Specifies the output length, when any of the options are used that specify padding (all but LEFT). For example: ◆ strnum(Col1, right, 6) used for a column value of -1.27 becomes ##-1.27. The minus sign is not counted as a digit, but the decimal is. ◆ strnum(Col1, rightzero, 6) used for a column value of -1.27 becomes -001.27. The minus sign is not counted as a digit, but the decimal is.

Example Assuming a source column named NUM has a value of 15 and the target column's maximum length is 5 characters, the following examples show the different types of results obtained with formatting options.

Function statement	**Result** (# denotes a space)
`CHAR1 = @STRNUM (NUM, LEFT)`	15
`CHAR1 = @STRNUM (NUM, LEFTSPACE)`	15###
`CHAR1 = @STRNUM (NUM, RIGHTZERO)`	00015
`CHAR1 = @STRNUM (NUM, RIGHT)`	###15

If an output <length> of 4 is specified in the preceding example, the following shows the different types of results.

Function statement	**Result** (# denotes a space)
`CHAR1 = @STRNUM (NUM, LEFTSPACE, 4)`	15##
`CHAR1 = @STRNUM (NUM, RIGHTZERO, 4)`	0015
`CHAR1 = @STRNUM (NUM, RIGHT, 4)`	##15

STRRTRIM

Use the @STRRTRIM function to trim trailing spaces.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRRTRIM (<string>)
```

Argument	Description
`<string>`	The name of a character column or a literal string. Enclose literals within quotes.

Example

```
street_address = @strrtrim (address)
```

STRSUB

Use the @STRSUB function to substitute strings within a string (character) column or constant.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@STRSUB
(<source string>, <search string>, <substitute string>
[, <search string>, <substitute string>] [, ...])
```

Argument	Description
`<source string>`	The source string or column containing the characters for which substitution is to occur.
`<search string>`	The string for which substitution is to occur.
`<substitute string>`	The string that will be substituted for the search string.

Example 1 The following returns xxABCxx.

```
@STRSUB ("123ABC123", "123", "xx")
```

Example 2 The following returns 023zBC023.

```
@STRSUB ("123ABC123", "A", "z", "1", "0")
```

STRTRIM

Use the @STRTRIM function to trim leading and trailing spaces.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRTRIM (<string>)
```

Argument	Description
`<string>`	The name of a character column or a literal string. Enclose literals within quotes.

Example

```
pin_no = @strtrim (custpin)
```

STRUP

Use the @STRUP function to change an alphanumeric string or string (character) column to upper case.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support NCHAR or NVARCHAR data types.

Syntax

```
@STRUP (<string>)
```

Argument	Description
`<string>`	The name of a character column or a literal string. Enclose literals within quotes.

Example The following returns “SALESPERSON.”

```
@STRUP (“salesperson”)
```

TOKEN

Use the @TOKEN function to retrieve token data that is stored in the user token area of the Oracle GoldenGate record header. You can map token data to a target column by using @TOKEN in the source expression of a COLMAP clause. As an alternative, you can use @TOKEN within a SQLEXEC statement, an Oracle GoldenGate macro, or a user exit.

To define token data, use the TOKENS clause of the TABLE parameter in the Extract parameter file. For more information about using tokens, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

```
@TOKEN (“<token name>”)
```

Argument	Description
`“<token name>”`	The name, in quotes, of the token for which data is to be retrieved.

Example In the following example, 10 tokens are mapped to target columns.

```
MAP ora.oratest, TARGET ora.rpt,
COLMAP (
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")
);
```

VALONEOF

Use the @VALONEOF function to compare a string or string (character) column to a list of values. If the value or column is in the list, 1 is returned; otherwise 0 is returned.

For this function, 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 Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support NCHAR or NVARCHAR data types.

Syntax `@VALONEOF (<expression>, <value> [, <value>] [, ...])`

Argument	Description
`<expression>`	The name of a character column or a literal enclosed within quotes.
`<value>`	A criteria value.

Example In the following example, if STATE is CA or NY, the expression returns “COAST,” which is the response returned by @IF when the value is non-zero (true). Otherwise, the expression returns “MIDDLE.”

```
@IF (@VALONEOF (STATE, "CA", "NY"), "COAST", "MIDDLE")
```

CHAPTER 5
User Exit Functions

This chapter describes the Oracle GoldenGate user exit functions and their syntax. For more information about using Oracle GoldenGate user exits, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Calling a user exit

Write the user exit routine in C programming code. Use the CUSEREXIT parameter to call the user exit from a Windows DLL or UNIX shared object at a defined exit point within Oracle GoldenGate processing. Your user exit routine must be able to accept different events and information from the Extract and Replicat processes, process the information as desired, and return a response and information to the caller (the Oracle GoldenGate process that called it). For more information and syntax for the CUSEREXIT parameter, see page 153.

User exit function summary

Parameter	Description
ERCALLBACK	Implements a callback routine. Callback routines retrieve record and Oracle GoldenGate context information, and they modify the contents of data records.
EXIT_CALL_RESULT	Provides a response to the routine.
EXIT_CALL_TYPE	Indicates when, during processing, the routine is called.
EXIT_PARAMS	Supplies information to the routine.

Using EXIT_CALL_TYPE

Use EXIT_CALL_TYPE to indicate when, during processing, the Extract or Replicat process (the caller) calls a user exit routine. A process can call a routine with the following calls.

Table 51 User exit calls

Call type	Processing point
EXIT_CALL_ABORT_TRANS	Valid when the RECOVERYOPTIONS mode is APPEND (the default). Called when a data pump or Replicat reads a RESTART ABEND record from the trail, placed there by a writer process that abended. (The writer process can be the primary Extract writing to a local trail read by a data pump, or a data pump writing to a remote trail read by Replicat.) This call type enables the user exit to abort or discard the transaction that was left incomplete when the writer process stopped, and then to recover and resume processing at the start of the previous completed transaction.
EXIT_CALL_BEGIN_TRANS	Called just before either of the following: ◆ a BEGIN record of a transaction that is read by a data pump ◆ the start of a Replicat transaction
EXIT_CALL_CHECKPOINT	Called just before an Extract or Replicat checkpoint is written.
EXIT_CALL_DISCARD_ASCII_RECORD	Called during Extract processing before an ASCII input record is written to the discard file. The associated ASCII buffer can be retrieved and manipulated by the user exit using callback routines. This call type is not applicable for use with the Replicat process.
EXIT_CALL_DISCARD_RECORD	Called during Replicat processing before a record is written to the discard file. Records can be discarded for several reasons, such as when a value in the Oracle GoldenGate change record is different from the current version in the target table.The associated discard buffer can be retrieved and manipulated by the user exit using callback routines. This call type is not applicable for use with the Extract process.
EXIT_CALL_END_TRANS	Called just after either of the following: ◆ an END record of a transaction that is read by a data pump ◆ the last record in a Replicat transaction
EXIT_CALL_FATAL_ERROR	Called during Extract or Replicat processing just before Oracle GoldenGate terminates after a fatal error.

Table 51 User exit calls (continued)

Call type	Processing point
EXIT_CALL_PROCESS_MARKER	Called during Replicat processing when a marker from a NonStop server is read from the trail, and before writing to the marker history file.
EXIT_CALL_PROCESS_RECORD	◆ For Extract, called before a record buffer is output to the trail. ◆ For Replicat, called just before a replicated operation is performed. This call is the basis of most user exit processing. When EXIT_CALL_PROCESS_RECORD is called, the record buffer and other record information are available through callback routines. If source-target mapping is specified in the parameter file, the mapping is performed before the EXIT_CALL_PROCESS_RECORD event takes place. The user exit can map, transform, clean, or perform virtually any other operation with the record. The user exit can return a status indicating whether the caller should process or ignore the record.
EXIT_CALL_START	Called at the start of processing. The user exit can perform initialization work, such as opening files and initializing variables.
EXIT_CALL_STOP	Called before the process stops gracefully or ends abnormally. The user exit can perform completion work, such as closing files or outputting totals.
EXIT_CALL_RESULT	Set by the user exit routines to instruct the caller how to respond when each exit call completes.

Using EXIT_CALL_RESULT

Use EXIT_CALL_RESULT to provide a response to the routine.

Table 52 User exit responses

Call result	Description
EXIT_ABEND_VAL	Instructs the caller to terminate immediately.
EXIT_IGNORE_VAL	Rejects records for further processing. EXIT_IGNORE_VAL is appropriate when the user exit performs all the required processing for a record and there is no need to output or replicate the data record.

Table 52 User exit responses

Call result	Description
EXIT_OK_VAL	If the routine does nothing to respond to an event, EXIT_OK_VAL is assumed. If the exit call type is any of the following... EXIT_CALL_PROCESS_RECORD EXIT_CALL_DISCARD_RECORD EXIT_CALL_DISCARD_ASCII_RECORD ... and EXIT_OK_VAL is returned, then Oracle GoldenGate processes the record buffer that was returned by the user exit.
EXIT_PROCESSED_REC_VAL	Instructs Extract or Replicat to skip the record, but update the statistics that are printed to the report file for that table and for that operation type.
EXIT_STOP_VAL	Instructs the caller to stop processing gracefully. EXIT_STOP_VAL or EXIT_ABEND_VAL may be appropriate when an error condition occurs in the user exit.

Using EXIT_PARAMS

Use EXIT_PARAMS to supply information to the user exit routine, such as the program name and user-defined parameters. You can process a single data record multiple times.

Table 53 User exit input

Exit parameter	Description
PROGRAM_NAME	Specifies the full path and name of the calling process, for example \ggs\extract or \ggs\replicat. Use this parameter when loading an Oracle GoldenGate callback routine using the Windows API or to identify the calling program when user exits are used with both Extract and Replicat processing.
FUNCTION_PARAM	◆ Allows you to pass a parameter that is a literal string to the user exit. Specify the parameter with the EXITPARAM option of the TABLE or MAP statement from which the parameter will be passed. See page 251. This is only valid during the exit call to process a specific record. ◆ The FUNCTION_PARAM can also be used at the exit call startup event to pass the parameters that are specified in the PARAMS option of the CUSEREXIT parameter. (See page 153.) This is only valid to supply a global parameter at exit startup.
MORE_RECS_IND	Set on return from an exit. For database records, determines whether Extract or Replicat processes the record again. This allows the user exit to output many records per record processed by Extract, a common function when converting Enscribe to SQL (data normalization). To request the same record again, set MORE_RECS_IND to CHAR_NO_VAL or CHAR_YES_VAL.

Using ERCALLBACK

Use ERCALLBACK to execute a callback routine. A user callback routine retrieves context information from the Extract or Replicat process and sets context values, including the record itself, when the call type is one of the following:

- EXIT_CALL_PROCESS_RECORD
- EXIT_CALL_DISCARD_RECORD
- EXIT_CALL_DISCARD_ASCII_RECORD

Syntax

```
ERCALLBACK (<function_code>, <buffer>, <result_code>);
```

Argument	Description
`<function_code>`	The function to be executed by the callback routine. The user callback routine behaves differently based on the function code passed to the callback routine. While some functions can be used for both Extract and Replicat, the validity of the function in one process or the other is dependent on the input parameters that are set for that function during the callback routine. See “Function Codes” on page 478 for a full description of available function codes.
`<buffer>`	A void pointer to a buffer containing a predefined structure associated with the specified function code.
`<result_code>`	The status of the function executed by the callback routine. The result code returned by the callback routine indicates whether or not the callback function was successful. A result code can be one of the values in Table 54.

Table 54 Result codes

Code	Description
EXIT_FN_RET_BAD_COLUMN_DATA	Invalid data was encountered when retrieving or setting column data.
EXIT_FN_RET_BAD_DATE_TIME	A date, timestamp, or interval type of column contains an invalid date or time value.
EXIT_FN_RET_BAD_NUMERIC_VALUE	A numeric type of column contains an invalid numeric value.
EXIT_FN_RET_COLUMN_NOT_FOUND	The column was not found in a compressed update record.
EXIT_FN_RET_ENV_NOT_FOUND	The specified environment value could not be found in the record.

Table 54 Result codes (continued) (continued)

Code	Description
EXIT_FN_RET_EXCEEDED_MAX_LENGTH	The metadata could not be retrieved because the name of the table or column did not fit in the allocated buffer.
EXIT_FN_RET_FETCH_ERROR	The record could not be fetched. View the error message to see the reason.
EXIT_FN_RET_INCOMPLETE_DDL_REC	An internal error occurred when processing the DDL record. The record is probably incomplete.
EXIT_FN_RET_INVALID_CALLBACK_FNC_CD	An invalid callback function code was passed to the callback routine.
EXIT_FN_RET_INVALID_COLUMN	A non-existent column was referred to in the function call.
EXIT_FN_RET_INVALID_COLUMN_TYPE	The routine is trying to manipulate a data type that is not supported by Oracle GoldenGate for that purpose.
EXIT_FN_RET_INVALID_CONTEXT	The callback function was called at an improper time.
EXIT_FN_RET_INVALID_PARAM	An invalid parameter was passed to the callback function.
EXIT_FN_RET_NO_SRCDB_INSTANCE	The source database instance could not be found.
EXIT_FN_RET_NO_TGTDB_INSTANCE	The target database instance could not be found.
EXIT_FN_RET_NOT_SUPPORTED	This function is not supported for this process.
EXIT_FN_RET_OK	The callback function succeeded.
EXIT_FN_RET_SESSION_CS_CNV_ERR	A ULIB_ERR_INVALID_CHAR_FOUND error was returned to the character-set conversion routine. The conversion failed.
EXIT_FN_RET_TABLE_NOT_FOUND	An invalid table name was specified.
EXIT_FN_RET_TOKEN_NOT_FOUND	The specified user token could not be found in the record.

Function Codes

Function codes determine the output of the callback routine. The callback routine expects the contents of the data buffer to match the structure of the specified function code. The callback routine function codes and their data buffers are described in the following sections. The following is a summary of available functions.

Table 55 Summary of Oracle GoldenGate function codes

Function code	Description
COMPRESS_RECORD	Use the COMPRESS_RECORD function when some, but not all, of a target table's columns are present after mapping and the entire record must be manipulated, rather than individual column values.
DECOMPRESS_RECORD	Use the DECOMPRESS_RECORD function when some, but not all, of a target table's columns are present after mapping and the entire record must be manipulated, rather than individual column values.
GET_BEFORE_AFTER_IND	Use the GET_BEFORE_AFTER_IND function to determine whether a record is a before image or an after image of the database operation.
GET_CATALOG_NAME_ONLY	Use the GET_CATALOG_NAME_ONLY function to return the name of the database catalog.
GET_COL_METADATA_FROM_INDEX	Use the GET_COL_METADATA_FROM_INDEX function to determine the column metadata that is associated with a specified column index.
GET_COL_METADATA_FROM_NAME	Use the GET_COL_METADATA_FROM_NAME function to determine the column metadata that is associated with a specified column name.
GET_COLUMN_INDEX_FROM_NAME	Use the GET_COLUMN_INDEX_FROM_NAME function to determine the column index associated with a specified column name.
GET_COLUMN_NAME_FROM_INDEX	Use the GET_COLUMN_NAME_FROM_INDEX function to determine the column name associated with a specified column index.
GET_COLUMN_VALUE_FROM_INDEX	Use the GET_COLUMN_VALUE_FROM_INDEX function to return the column value from the data record using the specified column index.
GET_COLUMN_VALUE_FROM_NAME	Use the GET_COLUMN_VALUE_FROM_NAME function to return the column value from the data record by using the specified column name.

Table 55 Summary of Oracle GoldenGate function codes (continued)

Function code	Description
GET_DATABASE_METADATA	Use the GET_DATABASE_METADATA function to return database metadata.
GET_DDL_RECORD_PROPERTIES	Use the GET_DDL_RECORD_PROPERTIES function to return information about a DDL operation.
GET_ENV_VALUE	Use the GET_ENV_VALUE function to return information about the Oracle GoldenGate environment.
GET_ERROR_INFO	Use the GET_ERROR_INFO function to return error information associated with a discard record.
GET_GMT_TIMESTAMP	Use the GET_GMT_TIMESTAMP function to return the operation commit timestamp in GMT format.
GET_MARKER_INFO	Use the GET_MARKER_INFO function to return marker information when posting data. Use markers to trigger custom processing within a user exit.
GET_OPERATION_TYPE	Use the GET_OPERATION_TYPE function to determine the operation type associated with a record.
GET_POSITION	Use the GET_POSITION function is obtain a read position of an Extract data pump or Replicat in the Oracle GoldenGate trail.
GET_RECORD_BUFFER	Use the GET_RECORD_BUFFER function to obtain information for custom column conversions.
GET_RECORD_LENGTH	Use the GET_RECORD_LENGTH function to return the length of the data record.
GET_RECORD_TYPE	Use the GET_RECORD_TYPE function to return the type of record being processed
GET_SCHEMA_NAME_ONLY	Use the GET_SCHEMA_NAME_ONLY function to return only the schema name of a table.
GET_SESSION_CHARSET	Use the GET_SESSION_CHARSET function to return the character set of the user exit session.
GET_STATISTICS	Use the GET_STATISTICS function to return the current processing statistics for the Extract or Replicat process.
GET_TABLE_COLUMN_COUNT	Use the GET_TABLE_COLUMN_COUNT function to return the total number of columns in a table.

Table 55 Summary of Oracle GoldenGate function codes (continued)

Function code	Description
GET_TABLE_METADATA	Use the GET_TABLE_METADATA function to return metadata for the table that associated with the record that is being processed.
GET_TABLE_NAME	Use the GET_TABLE_NAME function to return the name of the source or target table associated with the record being processed.
GET_TABLE_NAME_ONLY	Use the GET_TABLE_NAME_ONLY function to return only the name of the table.
GET_TIMESTAMP	Use the GET_TIMESTAMP function to return the I/O timestamp associated with a source data record.
GET_TRANSACTION_IND	Use the GET_TRANSACTION_IND function to determine whether a data record is the first, last or middle operation in a transaction,
GET_USER_TOKEN_VALUE	Use the GET_USER_TOKEN_VALUE function to obtain the value of a user token from a trail record.
OUTPUT_MESSAGE_TO_REPORT	Use the OUTPUT_MESSAGE_TO_REPORT function to output a message to the report file.
RESET_USEREXIT_STATS	Use the RESET_USEREXIT_STATS function to reset the statistics for the Oracle GoldenGate process.
SET_COLUMN_VALUE_BY_INDEX	Use the SET_COLUMN_VALUE_BY_INDEX function to modify a single column value without manipulating the entire data record.
SET_COLUMN_VALUE_BY_NAME	Use the SET_COLUMN_VALUE_BY_NAME function to modify a single column value without manipulating the entire data record.
SET_OPERATION_TYPE	Use the SET_OPERATION_TYPE function to change the operation type associated with a data record.
SET_RECORD_BUFFER	Use the SET_RECORD_BUFFER function for compatibility with HP NonStop user exits, and for complex data record manipulation.
SET_SESSION_CHARSET	Use the SET_SESSION_CHARSET function to set the character set of the user exit session.
SET_TABLE_NAME	Use the SET_TABLE_NAME function to change the table name associated with a data record.

COMPRESS_RECORD

Valid for Extract and Replicat

Use the COMPRESS_RECORD function to re-compress records that have been decompressed with the DECOMPRESS_RECORD function. Call COMPRESS_RECORD only *after* using DECOMPRESS_RECORD.

The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.

Syntax

```
#include “usrdecs.h”
short result_code;
compressed_rec_def compressed_rec;
ERCALLBACK (COMPRESS_RECORD, &compressed_rec, &result_code);
```

Buffer

```
typedef struct
{
char *compressed_rec;
long compressed_len;
char *decompressed_rec;
long decompressed_len;
short *columns_present;
short source_or_target;
char requesting_before_after_ind;
} compressed_rec_def;
```

Input Can be the following:

Input	Description
decompressed_rec	A pointer to the buffer containing the record before compression. The record is assumed to be in the default Oracle GoldenGate canonical format.
decompressed_len	The length of the decompressed record.
source_or_target	One of the following to indicate whether the source or target record is being compressed. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL
requesting_before_after_ind	Used as internal input. Does not need to be set. If set, it will be ignored.
columns_present	An array of values that indicates the columns present in the compressed record. For example, if the first, third and sixth columns exist in the compressed record, and the total number of columns in the table is seven, the array should contain: 1, 0, 1, 0, 0, 1, 0 Use the GET_TABLE_COLUMN_COUNT function to get the number of columns in the table (see page 521).

Output Can be the following:

Output	Description
`compressed_rec`	A pointer to the record returned in compressed format. Typically, compressed_rec is a pointer to a buffer of type exit_rec_buf_def. The exit_rec_buf_def buffer contains the actual record about to be processed by Extract or Replicat. The buffer is supplied when the call type is EXIT_CALL_DISCARD_RECORD. Exit routines may change the contents of this buffer, for example to perform custom mapping functions. The caller must ensure that the appropriate amount of memory is allocated to compressed_rec.
`compressed_len`	The returned length of the compressed record.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
EXIT_FN_RET_INVALID_PARAM
```

DECOMPRESS_RECORD

Valid for Extract and Replicat

Use the DECOMPRESS_RECORD function when you want to retrieve or manipulate an entire update record with the GET_RECORD_BUFFER (see page 512) or SET_RECORD_BUFFER function (see page 537) and the record is compressed. DECOMPRESS_RECORD makes compressed records easier to process and map by putting the record into its logical column layout. The columns that are present will be in the expected positions without the index and length indicators (see "Compressed record format"). The missing columns will be represented as zeroes. When used, DECOMPRESS_RECORD should be invoked before any manipulation occurs. After the user exit processing is completed, use the COMPRESS_RECORD function (see page 481) to re-compress the record before returning it to the Oracle GoldenGate process.

This function is valid for processing UPDATE operations only. Deletes, inserts and updates appear in the buffer as full record images.

The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.

Compressed record format

Compressed SQL updates have the following format:

```
<index><length><value>[<index><length><value>][...]
```

Where:

- <index> is a two-byte index into the list of columns of the table (first column is zero).
- <length> is the two-byte length of the table.
- <value> is the actual column value, including one of the following two-byte null indicators when applicable. `0` is not null. `-1` is null.

Syntax

```
#include “usrdecs.h”
short result_code;
compressed_rec_def compressed_rec;
ERCALLBACK (DECOMPRESS_RECORD, &compressed_rec, &result_code);
```

Buffer

```
typedef struct
{
char *compressed_rec;
long compressed_len;
char *decompressed_rec;
long decompressed_len;
short *columns_present;
short source_or_target;
char requesting_before_after_ind;
} compressed_rec_def;
```

Input Can be the following:

Input	Description
compressed_rec	A pointer to the record in compressed format. Use the GET_RECORD_BUFFER function to obtain this value (see page 512).
compressed_len	The length of the compressed record. Use the GET_RECORD_BUFFER (see page 512) or GET_RECORD_LENGTH (see page 515) function to get this value.
source_or_target	One of the following to indicate whether the source or target record is being decompressed. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL
requesting_before_ after_ind	Used as internal input. Does not need to be set. If set, it will be ignored.

Output Can be the following:

Output	Description
decompressed_rec	A pointer to the record returned in decompressed format. The record is assumed to be in the Oracle GoldenGate internal canonical format. The caller must ensure that the appropriate amount of memory is allocated to decompressed_rec.
decompressed_len	The returned length of the decompressed record.

Output	Description
`columns_present`	An array of values that indicate the columns present in the compressed record. For example, if the first, third and sixth columns exist in the compressed record, and the total number of columns in the table is seven, the array should contain: `1, 0, 1, 0, 0, 1, 0` This array helps mapping functions determine when and whether a compressed column should be mapped.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
EXIT_FN_RET_INVALID_PARAM
```

GET_BEFORE_AFTER_IND

Valid for Extract and Replicat

Use the GET_BEFORE_AFTER_IND function to determine whether a record is a before image or an after image of the database operation. Inserts are after images, deletes are before images, and updates can be either before or after images (see the Extract and Replicat parameters GETUPDATEBEFORES and GETUPDATEAFTERS). If update before images are being extracted, the before images precede the after images within the same update.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_BEFORE_AFTER_IND, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input None

Output Can be the following:

Output	Description

`before_after_ind`	One of the following to indicate whether the record is a before or after image.
	`BEFORE_IMAGE_VAL`
	`AFTER_IMAGE_VAL`

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_CATALOG_NAME_ONLY

Valid for Extract and Replicat

Use the GET_CATALOG_NAME_ONLY function to retrieve the catalog, but not the schema or name, of the source or target table associated with the record being processed. To return the fully qualified name, see:

GET_TABLE_NAME

To return other parts of the table name, see:

GET_TABLE_NAME_ONLY

GET_SCHEMA_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_TABLE_NAME, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input Can be the following:

Input	Description
buffer	A pointer to a buffer to accept the returned catalog name. The name is null-terminated. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the catalog name is interpreted in the session character set.
max_length	The maximum length of your allocated buffer to accept the name. This is returned as a NULL terminated string.
source_or_target	One of the following indicating whether to return the source or target table catalog. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL

Output Can be the following:

Output	Description
buffer	The fully qualified, null-terminated catalog name.
actual length	The string length of the returned name. The actual length does not include the null terminator.
value_truncated	A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the catalog name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_COL_METADATA_FROM_INDEX

Valid for Extract and Replicat

Use the GET_COL_METADATA_FROM_INDEX function to retrieve column metadata by specifying the index of the desired column.

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
col_metadata_def column_meta_rec;
ERCALLBACK (GET_COL_METADATA_FROM_INDEX, &column_meta_rec, &result_code);
```

Buffer

```
typedef struct
{
   short column_index;
   char *column_name;
   long max_name_length;
   short native_data_type;
   short gg_data_type;
   short gg_sub_data_type;
   short is_nullable;
   short is_part_of_key;
   short key_column_index;
   short length;
   short precision;
   short scale;
   short source_or_target;
} col_metadata_def;
```

Input Can be the following:

Input	Description
`column_index`	The column index of the column value to be returned.
`max_name_length`	The maximum length of the returned column name. Typically, the maximum length is the length of the name buffer. Since the returned name is null-terminated, the maximum length should equal the maximum length of the column name.

`source_or_target`	One of the following to indicate whether the source or target record is being compressed.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description
`column_name`	The column name of the column value to be returned.
`native_data_type`	The native (to the database) data type of the column. Either native_data_type or dd_data_type is returned, depending on the process, as follows: ◆ If Extract is making the callback request for a source column, native_data_type is returned. If Extract is requesting a mapped target column, gg_data_type is returned (assuming there is a target definitions file on the system).

<table>
<tr><th>Output</th><th>Description</th></tr>
<tr><td></td><td>◆ If an Extract data pump is making the callback request for a source column and there is a local database, native_data_type is returned. If there is no database, gg_data_type is returned (assuming there is a source definitions file on the system). If the pump is requesting the target column, gg_data_type is returned (assuming a target definitions file exists on the system).
◆ If Replicat is making the callback request for the source column, then gg_data_type is returned (assuming a source definitions file exists on the system). If Replicat is requesting the source column and ASSUMETARGETDEFS is being used in the parameter file, then native_data_type is returned. If Replicat is requesting the target column, native_data_type is returned.</td></tr>
<tr><td><code>gg_data_type</code></td><td>The Oracle GoldenGate data type of the column.</td></tr>
<tr><td><code>gg_sub_data_type</code></td><td>The Oracle GoldenGate sub-type of the column.</td></tr>
<tr><td><code>is_nullable</code></td><td>Flag indicating whether the column permits a null value (TRUE or FALSE).</td></tr>
<tr><td><code>is_part_of_key</code></td><td>Flag (TRUE or FALSE) indicating whether the column is part of the key that is being used by Oracle GoldenGate.</td></tr>
<tr><td><code>key_column_index</code></td><td>Indicates the order of the columns in the index. For example, the following table has two key columns that exist in a different order from the order in which they are declared in the primary key.
<pre>
CREATE TABLE ABC
(
cust_code   VARCHAR2(4),
name   VARCHAR2(30),
city   VARCHAR2(20),
state   CHAR(2),
PRIMARY KEY (city, cust_code)
USING INDEX
);
</pre>
Executing the callback function for each column in the logical column order returns the following:
<table>
<tr><th>Column name</th><th>Key index value returned</th></tr>
<tr><td>cust_code</td><td>1</td></tr>
<tr><td>name</td><td>-1</td></tr>
<tr><td>city</td><td>0</td></tr>
<tr><td>state</td><td>-1</td></tr>
</table>
◆ If the column is part of the key, the value returned is the order of the column within the key.
◆ If the column is not part of the key, a value of -1 is returned.</td></tr>
</table>

Output	Description
`length`	Returns the length of the column.
`precision`	If a numeric data type, returns the precision of the column.
`scale`	If a numeric data type, returns the scale.

Return Values

```
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_EXCEEDED_MAX_LENGTH
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_OK
```

GET_COL_METADATA_FROM_NAME

Valid for Extract and Replicat

Use the GET_COL_METADATA_FROM_NAME function to retrieve column metadata by specifying the name of the desired column. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include "usrdecs.h"
short result_code;
col_metadata_def column_meta_rec;
ERCALLBACK (GET_COL_METADATA_FROM_NAME, &column_meta_rec, &result_code);
```

Buffer

```
typedef struct
{
   short column_index;
   char *column_name;
   long max_name_length;
   short native_data_type;
   short gg_data_type;
   short gg_sub_data_type;
   short is_nullable;
   short is_part_of_key;
   short key_column_index;
   short length;
   short precision;
   short scale;
   short source_or_target;
} col_metadata_def;
```

Input Can be the following:

Input	Description
`column_name`	The column name of the column value to be returned.

`source_or_target`	One of the following to indicate whether the source or target record is being compressed.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description
`column_index`	The column index of the column value to be returned.

`source_or_target`	One of the following to indicate whether the source or target record is being compressed.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

`native_data_type`	The native (to the database) data type of the column.
`gg_data_type`	The Oracle GoldenGate data type of the column.
`gg_sub_data_type`	The Oracle GoldenGate sub-type of the column.
`is_nullable`	Flag indicating whether the column permits a null value (TRUE or FALSE).
`is_part_of_key`	Flag (TRUE or FALSE) indicating whether the column is part of the key that is being used by Oracle GoldenGate.

`key_column_index`	Indicates the order of the columns in the index. For example, the following table has two key columns that are defined in one order in the table and another in the index definition.
	`CREATE TABLE tcustmer`
	`(`
	`cust_code        VARCHAR2(4),`
	`name             VARCHAR2(30),`
	`city             VARCHAR2(20),`
	`state            CHAR(2),`
	`PRIMARY KEY (city, cust_code)`
	`USING INDEX`
	`);`

<table>
<tr><th>Output</th><th>Description</th></tr>
<tr><td></td><td>The return is as follows:
<table>
<tr><th>Column name</th><th>Key index value returned</th></tr>
<tr><td>cust_code</td><td>1</td></tr>
<tr><td>name</td><td>-1</td></tr>
<tr><td>city</td><td>0</td></tr>
<tr><td>state</td><td>-1</td></tr>
</table>
◆ If the column is part of the key, its order in the index is returned as an integer.
◆ If the column is not part of the key, a value of -1 is returned.</td></tr>
<tr><td><code>length</code></td><td>Returns the length of the column.</td></tr>
<tr><td><code>precision</code></td><td>If a numeric data type, returns the precision of the column.</td></tr>
<tr><td><code>scale</code></td><td>If a numeric data type, returns the scale.</td></tr>
</table>

Return Values

```
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_EXCEEDED_MAX_LENGTH
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_OK
```

GET_COLUMN_INDEX_FROM_NAME

Valid for Extract and Replicat

Use the GET_COLUMN_INDEX_FROM_NAME function to determine the column index associated with a specified column name. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include “usrdecs.h”
short result_code;
env_value_def env_value;
ERCALLBACK (GET_COLUMN_INDEX_FROM_NAME, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input Can be the following:

Input	Description
buffer	A pointer to the column name
actual_length	The length of the column name within the buffer.
source_or_target	One of the following to indicate whether to use the source or target table to look up column information. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL

Output Can be the following:

Output	Description
index	The returned column index for the specified column name.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_COLUMN_NAME_FROM_INDEX

Valid for Extract and Replicat

Use the GET_COLUMN_NAME_FROM_INDEX function to determine the column name associated with a specified column index. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include “usrdecs.h”
short result_code;
env_value_def env_value;
ERCALLBACK (GET_COLUMN_NAME_FROM_INDEX, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input Can be the following:

Input	Description
buffer	A pointer to a buffer to accept the returned column name. The column name is null-terminated.
max_length	The maximum length of your allocated buffer to accept the resulting column name. This is returned as a NULL terminated string.
index	The column index of the column name to be returned.
source_or_target	One of the following to indicate whether to use the source or target table to look up column information. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL

Output Can be the following:

Output	Description
buffer	The null-terminated column name.
actual length	The string length of the returned column name. The actual length does not include the null terminator.
value_truncated	A flag (0 or 1) to indicate whether or not the value was truncated. Truncation occurs if the length of the column name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_COLUMN_VALUE_FROM_INDEX

Valid for Extract and Replicat

Use the GET_COLUMN_VALUE_FROM_INDEX function to retrieve the column value from the data record using the specified column index. Column values are the basis for most logic within the user exit. You can base complex logic on the values of individual columns within the data record. You can specify the character format of the returned value.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (CHAR/VARCHAR2/CLOB, NCHAR/NVARCHAR2/NCLOB), a SQL date/timestamp/interval/number type)
- The column_value_mode indicator is set to EXIT_FN_CNVTED_SESS_CHAR_FORMAT.

Syntax

```
#include “usrdecs.h”
short result_code;
column_def column;
ERCALLBACK (GET_COLUMN_VALUE_FROM_INDEX, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input Can be the following:

Input	Description
`column_value`	A pointer to a buffer to accept the returned column value.
`max_value_length`	The maximum length of the returned column value. Typically, the maximum length is the length of the column value buffer. If ASCII format is specified with column_value_mode, the column value is null-terminated and the maximum length should equal the maximum length of the column value.

Input	Description
`column_index`	The column index of the column value to be returned.
`column_value_mode`	Indicates the format of the column value. EXIT_FN_CHAR_FORMAT EXIT_FN_RAW_FORMAT EXIT_FN_CNVTED_SESS_CHAR_FORMAT **EXIT_FN_CHAR_FORMAT** ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type UTF16_BE, which is converted to UTF8.) **Note**: A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of 0 becomes the string terminator. ◆ Dates are in the format CCYY-MM-DD HH:MI:SS.FFFFFF, in which the fractional time is database-dependent. ◆ Numeric values are in their string format. For example, 123.45 is represented as "123.45". ◆ Non-printable characters or binary values are converted to hexidecimal notation. ◆ Floating point types are output as null-terminated strings, to the first 14 significant digits. **EXIT_FN_RAW_FORMAT** Internal Oracle GoldenGate canonical format: This format includes a two-byte null indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type. **EXIT_FN_CNVTED_SESS_CHAR_FORMAT** User exit character set: This only applies if the column data type is: ◆ a character-based type, single or multi-byte ◆ a numeric type with a string representation This format is not null-terminated.

`source_or_target`	One of the following to indicate whether to use the source or the target data record to retrieve the column value.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Input	Description
`requesting_before_after_ind`	Set when processing an after image record and you want the before-image column value of either an update or a primary key update. To get the "before" value of the column while processing an "after image" of a primary key update or a regular (non-key) update record, set the requesting_before_after_ind flag to BEFORE_IMAGE_VAL. ◆ To access the before image of the key columns of a primary key update, nothing else is necessary. ◆ To access non-key columns of a primary key update or any column of a regular update, the before image must be available. The default setting is AFTER_IMAGE_VAL (get the after image of the column) when an explicit input for requesting_before_after_ind is not specified. To make a before image available, you can use the GETUPDATEBEFORES parameter or you can use the INCLUDEUPDATEBEFORES option within the CUSEREXIT parameter statement. Note that: ◆ GETUPDATEBEFORES causes an Extract process to write before-image records to the trail and also to make an EXIT_CALL_PROCESS_RECORD call to the user exit with the before images. ◆ INCLUDEUPDATEBEFORES does not cause an EXIT_CALL_PROCESS_RECORD call to the user exit nor, in the case of Extract, does it cause the process to write the before image to the trail.

Output Can be the following:

Output	Description
`column_value`	A pointer to the returned column value. If column_value_mode is specified as EXIT_FN_CHAR_FORMAT, the column value is returned as a null-terminated ASCII string; otherwise, the column value is returned in the Oracle GoldenGate internal canonical format. In ASCII format, dates are returned in the following format: YYYY-MM-DD HH:MI:SS.FFFFFF The inclusion of fractional time is database-dependent.
`actual_value_length`	The string length of the returned column name, in bytes. The actual length does not include a null terminator when column_value_mode is specified as EXIT_FN_CHAR_FORMAT.

Output	Description
`null_value`	A flag (0 or 1) indicating whether or not the column value is null. If the null_value flag is 1, then the column value buffer is filled with null bytes.
`value_truncated`	A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the column value exceeds the maximum buffer length. If column_value_mode was specified as EXIT_FN_CHAR_FORMAT, the null terminator is included in the length of the column.
`char more_lob_data`	A flag that indicates if more LOB data is present beyond the initial 4K that can be stored in the base record. When a LOB is larger than the 4K limit, it is stored in LOB fragments. You must allocate the appropriate amount of memory to contain the returned values. Oracle GoldenGate will access LOB columns up to 8K of data at all times, filling up the buffer to the amount that the user exit has allocated. If the LOB is larger than that which was allocated, subsequent callbacks are required to obtain the total column data, until all data has been sent to the user exit. To determine the end of the data, evaluate more_lob_data. The user exit sets this flag to either CHAR_NO_VAL or CHAR_YES_VAL before accessing a new column. If this flag is still initialized after first callback and is not set to either CHAR_YES_VAL or CAR_NO_VAL, then one of the following is true: ◆ Enough memory was allocated to handle the LOB. ◆ It is not a LOB. ◆ It was not over the 4K limit of the base trail record size. It is recommended that you obtain the source table metadata to determine if a column might be a LOB.

Return Values

```
EXIT_FN_RET_BAD_COLUMN_DATA
EXIT_FN_RET_COLUMN_NOT_FOUND
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_COLUMN_VALUE_FROM_NAME

Valid for Extract and Replicat

Use the GET_COLUMN_VALUE_FROM_NAME function to retrieve the column value from the data record by using the specified column name. Column values are the basis for most logic within the user exit. You can base complex logic on the values of individual columns within the data record.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (CHAR/VARCHAR2/CLOB, NCHAR/NVARCHAR2/NCLOB), a SQL date/timestamp/interval/number type)
- The column_value_mode indicator is set to EXIT_FN_CNVTED_SESS_CHAR_FORMAT.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include “usrdecs.h”
short result_code;
column_def column;
ERCALLBACK (GET_COLUMN_VALUE_FROM_NAME, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input Can be the following:

Input	Description
`column_value`	A pointer to a buffer to accept the returned column value.
`max_value_length`	The maximum length of the returned column value. Typically, the maximum length is the length of the column value buffer. If ASCII format is specified (see column_value_mode) the column value is null-terminated, and the maximum length should equal the maximum length of the column value.
`column_name`	The name of the column for the column value to be returned.
`column_value_mode`	Indicates the character set of the column value. EXIT_FN_CHAR_FORMAT EXIT_FN_RAW_FORMAT EXIT_FN_CNVTED_SESS_CHAR_FORMAT

Input	Description
	EXIT_FN_CHAR_FORMAT ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type UTF16_BE, which is converted to UTF8.) **Note**: A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of 0 becomes the string terminator. ◆ Dates are in the format CCYY-MM-DD HH:MI:SS.FFFFFF, in which the fractional time is database-dependent. ◆ Numeric values are in their string format. For example, 123.45 is represented as "123.45". ◆ Non-printable characters or binary values are converted to hexidecimal notation. ◆ Floating point types are output as null-terminated strings, to the first 14 significant digits. **EXIT_FN_RAW_FORMAT** Internal Oracle GoldenGate canonical format: This format includes a two-byte null indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type. **EXIT_FN_CNVTED_SESS_CHAR_FORMAT** User exit character set: This only applies if the column data type is: ◆ a character-based type, single or multi-byte ◆ a numeric type with a string representation This format is not null-terminated.

`source_or_target`	One of the following indicating whether to use the source or target data record to retrieve the column value.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Input	Description
`requesting_before_after_ind`	Set when processing an after image record and you want the before columns of either an update or a primary key update. To get the "before" value of the column while processing an "after image" of a primary key update or a regular (non-key) update record, set the requesting_before_after_ind flag to BEFORE_IMAGE_VAL. ◆ To access the before image of the key columns of a primary key update, nothing else is necessary. ◆ To access non-key columns of a primary key update or any column of a regular update, the before image must be available. The default setting is AFTER_IMAGE_VAL (get the after image of the column) when an explicit input for requesting_before_after_ind is not specified. To make a before image available, you can use the GETUPDATEBEFORES parameter or you can use the INCLUDEUPDATEBEFORES option within the CUSEREXIT parameter statement. Note that: ◆ GETUPDATEBEFORES causes an Extract process to write before-image records to the trail and also to make an EXIT_CALL_PROCESS_RECORD call to the user exit with the before images. ◆ INCLUDEUPDATEBEFORES does not cause an EXIT_CALL_PROCESS_RECORD call to the user exit nor, in the case of Extract, does it cause the process to write the before image to the trail.

Output Can be the following:

Output	Description
`column_value`	A pointer to the returned column value. If column_value_mode is specified as EXIT_FN_CHAR_FORMAT, the column value is returned as a null-terminated ASCII string; otherwise, the column value is returned in the Oracle GoldenGate internal canonical format. In ASCII format, dates are returned in the following format: CCYY-MM-DD HH:MI:SS.FFFFFF The inclusion of fractional time is database-dependent.
`actual length`	The string length of the returned column name. The actual length does not include a null terminator when column_value_mode is specified as EXIT_FN_CHAR_FORMAT.

Output	Description
`null_value`	A flag (0 or 1) indicating whether or not the column value is null. If the null_value flag is 1, then the column value buffer is filled with null bytes.
`value_truncated`	A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the column value exceeds the maximum buffer length. If column_value_mode was specified as EXIT_FN_CHAR_FORMAT, the null terminator is included in the length of the column.
`char more_lob_data`	A flag that indicates if more LOB data is present beyond the initial 4K that can be stored in the base record. When a LOB is larger than the 4K limit, it is stored in LOB fragments. You must allocate the appropriate amount of memory to contain the returned values. Oracle GoldenGate will access LOB columns up to 8K of data at all times, filling up the buffer to the amount that the user exit has allocated. If the LOB is larger than that which was allocated, subsequent callbacks are required to obtain the total column data, until all data has been sent to the user exit. To determine the end of the data, evaluate more_lob_data. The user exit sets this flag to either CAR_NO_VAL or CHAR_YES_VAL before accessing a new column. If this flag is still initialized after first callback and is not set to either CHAR_YES_VAL or CAR_NO_VAL, then one of the following is true: ◆ Enough memory was allocated to handle the LOB. ◆ It is not a LOB. ◆ It was not over the 4K limit of the base trail record size. It is recommended that you obtain the source table metadata to determine if a column might be a LOB.

Return Values

```
EXIT_FN_RET_BAD_COLUMN_DATA
EXIT_FN_RET_COLUMN_NOT_FOUND
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

Example

```
memset (&col_meta, 0, sizeof(col_meta));
if (record.mapped)
col_meta.source_or_target = EXIT_FN_TARGET_VAL;
else
col_meta.source_or_target = EXIT_FN_SOURCE_VAL;
col_meta.source_or_target = EXIT_FN_SOURCE_VAL;
col_meta.column_name = (char *)malloc(100);
col_meta.max_name_length = 100;
col_meta.column_index = 1;

call_callback (GET_COL_METADATA_FROM_NAME, &col_meta, &result_code);
```

GET_DATABASE_METADATA

Valid for Extract and Replicat

Use the GET_DATABASE_METADATA function to return the metadata of the database that is associated with a record.

Syntax

Buffer

```
typedef struct
{
char*    dbName;
long      dbName_max_length;
long      dbName_actual_length;
unsigned char    dbNameMetadata[MAXDBOBJTYPE];
char*   locale;
long     locale_max_length;
long     locale_actual_length;
} database_def;
typedef struct
{
    database_def  source_db_def;
    database_def  target_db_def;
} database_defs;
```

Input Can be the following:

Input	Description
`dbname`	A pointer to a buffer to accept the database name.
`dbname_max_length`	The maximum length of the buffer to hold the name.
`dbname_actual_length`	The actual length of the database name.
`dbNameMetadata`	The name metadata for case-sensitivity, which is the same value that is written by Extract and the data pump to a trail. See the Oracle GoldenGate *Administration Guide* for a list of macros that can be used by the user exit to check database object name metadata, given an object name type.
`locale`	A null-terminated character string specifying the locale of the database. This is returned as a conjunction of: ◆ ISO-639 two-letter language code ◆ ISO-3166 two-letter country code ◆ Variant code using '_' U+005F as separator. Example: "en_US", "ja_Japen"
`locale_max_length`	The maximum length of the buffer to accept the locale.

Input		Description
locale_actual_length		The actual length of the locale.
database_def	source_db_def	Directs the process to return metadata for the source database.
database_def	target_db_def	Directs the process to return metadata for the target database.

Output

Return Values

GET_DDL_RECORD_PROPERTIES

Valid for Extract and Replicat, for databases for which DDL replication is supported

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.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set. This includes the DDL type, the object type, the object name, the owner name, and the DDL text itself.

Syntax

```
#include "usrdecs.h"
short result_code;
ddl_record_def ddl_rec;
ERCALLBACK (GET_DDL_RECORD_PROPERTIES, &ddl_rec, &result_code);
```

Buffer

```
typedef struct
{
char *ddl_type;
long ddl_type_max_length; /* Maximum Description length PASSED IN BY USER */
long ddl_type_length; /* Actual length */

char *object_type;
long object_type_max_length; /* Maximum Description length PASSED IN BY USER */
long object_type_length; /* Actual length */

char *object_name;
long object_max_length; /* Maximum Description length PASSED IN BY USER */
long object_length; /* Actual length */

char *owner_name;
long owner_max_length; /* Maximum Description length PASSED IN BY USER */
long owner_length; /* Actual length */

char *ddl_text;
long ddl_text_max_length; /* Maximum Description length PASSED IN BY USER */
long ddl_text_length; /* Actual length */

short ddl_text_truncated; /* Was value truncated? */
short source_or_target; /* Source or target value? */
} ddl_record_def;
```

Input Can be the following:

Input	Description

`ddl_type_length`
`object_type_length`
`object_length`
`owner_length`
`ddl_text_length`
	A pointer to one buffer for each of these items to accept the returned column values. These items are as follows:
	◆ ddl_type_length contains the length of the type of DDL operation, for example a CREATE or ALTER.
	◆ object_type_length contains the length of type of database object that is affected by the DDL operation, for example TABLE or INDEX.
	◆ object_length contains the length of the name of the object.
	◆ owner_length contains the length of the owner of the object (schema or database).
	◆ ddl_text_length contains the length of the actual DDL statement text.

`ddl_type_max_length`	The maximum length of the DDL operation type that is returned by *ddl_type. The DDL type is any DDL command that is valid for the database, such as ALTER.
`object_type_max_length`	The maximum length of the object type that is returned by *object_type. The object type is any object that is valid for the database, such as TABLE, INDEX, and TRIGGER.

Input	Description
`object_max_length`	The maximum length of the name of the object that is returned by *object_name.
`owner_max_length`	The maximum length of the name of the owner that is returned by *owner_name.
`ddl_text_max_length`	The maximum length of the text of the DDL statement that is returned by *ddl_text.

`source_or_target`	One of the following indicating whether to return the operation type for the source or the target data record.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description

`ddl_type_length`
`object_type_length`
`object_length`
`owner_length`
`ddl_text_length`
	All of these fields return the actual length of the value that was requested. (See the input for descriptions.)

`ddl_text_truncated`	A flag (0 or 1) to indicate whether or not the DDL text was truncated. Truncation occurs if the length of the DDL text plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_OK
EXIT_FN_RET_NOT_SUPPORTED
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INCOMPLETE_DDL_REC
```

GET_ENV_VALUE

Valid for Extract and Replicat

Use the GET_ENV_VALUE function to return information about the Oracle GoldenGate environment. The information that is supplied is the same as that of the @GETENV column-conversion function and is specified by using the same input values. For more information about the valid information types, environment variables, and return values, see the @GETENV documentation on page 440.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

Syntax

```
#include "usrdecs.h"
short result_code;
getenv_value_def env_ptr;
ERCALLBACK (GET_ENV_VALUE, &env_ptr, &result_code);
```

Buffer

```
typedef struct
{
char *information_type;
char *env_value_name;
char *return_value;
long max_return_length;
long actual_length;
short value_truncated;
} getenv_value_def;
```

Input Can be the following:

Input	Description
information_type	The information type that is to be returned, for example "GGENVIRONMENT" or "GGHEADER". The information type must be supplied within double quotes. For a list of information types and subsequent detailed descriptions, see page 440.
env_value_name	The environment value that is wanted from the information type. The environment value must be supplied within double quotes. For valid values, see the **Environment value** list for the information type that you are using, referring to the documentation that starts on page 440. For example, if using the "GGENVIRONMENT" information type, a valid environment value would be "GROUPNAME" .
max_return_length	The maximum length of the buffer for this data.

Output Can be the following:

Output	Description
return_value	A valid **Return value** that is listed for the supplied environment value, as listed in the documentation for that environment variable.
actual_length	The actual length of the data in this buffer.
value_truncated	A flag (0 or 1) to indicate whether or not the value was truncated. Truncation occurs if the length of the value plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_OK
EXIT_FN_RET_ENV_NOT_FOUND
EXIT_FN_RET_INVALID_PARAM
```

GET_ERROR_INFO

Valid for Extract and Replicat

Use the GET_ERROR_INFO function to retrieve error information associated with a discard record. The user exit can use this information in custom error handling logic. For example, the user exit could send an e-mail message with detailed error information.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the message data that is exchanged between the user exit and the process is interpreted in the session character set.

Syntax

```
#include "usrdecs.h"
short result_code;
error_info_def error_info;
ERCALLBACK (GET_ERROR_INFO, &error_info, &result_code);
```

Buffer

```
typedef struct
{
long error_num;
char *error_msg;
long max_length;
long actual_length;
short msg_truncated;
} error_info_def;
```

Input Can be the following:

Input	Description
error_msg	A pointer to a buffer to accept the returned error message.
max_length	The maximum length of your allocated error_msg buffer to accept any resulting error message. This is returned as a NULL terminated string.

Output Can be the following:

Output	Description
error_num	The SQL or system error number associated with the discarded record.
error_msg	A pointer to the null-terminated error message string associated with the discarded record.
actual_length	The length of the error message, not including the null terminator.
msg_truncated	A flag (0 or 1) indicating whether or not the error message was truncated. Truncation occurs if the length of the error message plus a null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_GMT_TIMESTAMP

Valid for Extract and Replicat

Use the GET_GMT_TIMESTAMP function to retrieve the operation commit timestamp in GMT format. This function requires compiling with Version 2 usrdecs.h or later.

Syntax

```
#include “usrdecs.h”
short result_code;
record_def record;
ERCALLBACK (GET_GMT_TIMESTAMP, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input None

Output Can be the following:

Output	Description
`timestamp`	The returned 64-bit I/O timestamp in GMT format.
`io_datetime`	A null-terminated string containing the local I/O date and time: YYYY-MM-DD HH:MI:SS.FFFFFF The format of the datetime string is in the session character set.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_MARKER_INFO

Valid for Extract (data pump only) and Replicat

Use the GET_MARKER_INFO function to retrieve marker information sent from a NonStop source system when Replicat is applying data. Use markers to trigger custom processing within a user exit.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, all of the returned marker data is interpreted in the session character set.

Syntax

```
#include "usrdecs.h"
short result_code;
marker_info_def marker_info;
ERCALLBACK (GET_MARKER_INFO, &marker_info, &result_code);
```

Buffer

```
typedef struct
{
char *processed;
char *added;
char *text;
char *group;
char *program;
char *node;
} marker_info_def;
```

Input Can be the following:

Output	Description
processed	A pointer to a buffer to accept the processed return value.
added	A pointer to a buffer to accept the added return value.
text	A pointer to a buffer to accept the text return value.
group	A pointer to a buffer to accept the group return value.
program	A pointer to a buffer to accept the program return value.
node	A pointer to a buffer to accept the node return value.

Output Can be the following:

Output	Description
processed	A null-terminated string in the format of YYYY-MM-DD HH:MI:SS indicating the local date and time that the marker was processed.
added	A null-terminated string in the format of YYYY-MM-DD HH:MI:SS indicating the local date and time that the marker was added.
text	A null-terminated string containing the text associated with the marker.
group	A null-terminated string indicating the Replicat group that processed the marker.

Output	Description
`program`	A null-terminated string indicating the program that processed the marker.
`node`	A null-terminated string representing the Himalaya node on which the marker was originated.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_OPERATION_TYPE

Valid for Extract and Replicat

Use the GET_OPERATION_TYPE function to determine the operation type associated with a record. Knowing the operation type can be useful in a user exit. For example, the user exit can perform complex validations any time a delete is encountered. It also is important to know when a compressed record is being processed if the user exit is manipulating the full data record.

As an alternative, you can use the GET_RECORD_BUFFER function to determine the operation type (see page 512).

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_OPERATION_TYPE, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input Can be the following:

Input	Description

`source_or_target`	One of the following indicating whether to return the operation type for the source or the target data record.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description

`io_type`	Returned as one of the following:
	DDL type
	`SQL_DDL_VAL`
	DML types
	`DELETE_VAL`
	`INSERT_VAL`
	`UPDATE_VAL`
	Compressed Enscribe update
	`UPDATE_COMP_ENSCRIBE_VAL`
	Compressed SQL update
	`UPDATE_COMP_SQL_VAL`
	`UPDATE_COMP_PK_SQL_VAL`
	Other
	`TRUNCATE_TABLE_VAL`

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_POSITION

Valid for Extract (data pump only) and Replicat

Use the GET_POSITION function is obtain a read position of an Extract data pump or Replicat in the Oracle GoldenGate trail.

Syntax

```
#include "usrdecs.h"
short result_code;
ERCALLBACK (GET_POSITION &position_def, &result_code);
```

Buffer

```
typedef struct
{
char *position;
long position_len;
short position_type;
short ascii_or_internal;
} position_def;
```

Input Can be the following:

Input	Description
position_len	Allocation length for the position length.
position_type	Can be one of the following: ◆ STARTUP_CHECKPOINT The start position in the trail. ◆ CURRENT_CHECKPOINT The position of the last read in the trail.
column_value_mode	An indicator for the format in which the column value was passed. Currently, only the default Oracle GoldenGate canonical format is supported, as represented by: EXIT_FN_RAW_FORMAT

Output Can be the following:

Output	Description
*position	A pointer to a buffer representing the position values. This buffer is declared in the position_def as two binary values (unsigned int32t and int32t) as seqnorba for eight bytes in a char field. The user exit must move the data to the correct data type. Using this function on a Little Endian platform will cause the process to "reverse bytes" on the two fields individually.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_NOT_SUPPORTED
EXIT_FN_RET_OK
```

GET_RECORD_BUFFER

Valid for Extract and Replicat

Use the GET_RECORD_BUFFER function to obtain information for custom column conversions. User exits can be used for data mapping between dissimilar source and target records when the COLMAP option of the MAP or TABLE parameter is not sufficient. For example, you can use a user exit to convert a proprietary date field (for example, YYDDD) in an Enscribe database to a standard SQL date in the target record, while other columns are mapped by the Extract process by means of the COLMAP option.

You can use the SET_RECORD_BUFFER function (see page 537) to modify the data retrieved with GET_RECORD_BUFFER. However, it requires an understanding of the data record as written in the internal Oracle GoldenGate canonical format. As an alternative, you can set column values in the data record with the SET_COLUMN_VALUE_BY_INDEX function (see page 530) or the SET_COLUMN_VALUE_BY_NAME function (see page 533).

Deletes, inserts and updates appear in the buffer as full record images.

Compressed SQL updates have the following format:

```
<index><length><value>[<index><length><value>][...]
```

Where:

- <index> is a two-byte index into the list of columns of the table (first column is zero).
- <length> is the two-byte length of the table.
- <value> is the actual column value, including one of the following two-byte null indicators when applicable. `0` is not null. `-1` is null.

For SQL records, you can use the DECOMPRESS_RECORD function (page 482) to decompress the record for possible manipulation and then use the COMPRESS_RECORD function (page 481) to compress it again, as expected by the process.

Compressed Enscribe updates have the following format:

```
<offset><length><value>[<offset><length><value>][...]
```

Where:

- <offset> is the offset into the Enscribe record of the data fragment that changed.
- <length> is the length of the fragment.
- <value> is the data. Fragments can span field boundaries, so full fields are not always retrieved (unless compression is off or FETCHCOMPS is used).

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_RECORD_BUFFER, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input Can be the following:

Input	Description

`source_or_target`	One of the following indicating whether to return the record buffer for the source or target data record.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

`requesting_before_after_ind`	Optional. Set when requesting a record buffer on a record io_type of `UPDATE_COMP_PK_SQL_VAL` (primary key update). Use one of the following to indicate which portion of the primary key update is to be accessed. The default is AFTER_IMAGE_VAL.
	`BEFORE_IMAGE_VAL`
	`AFTER_IMAGE_VAL`

Output Can be the following:

Output	Description
`buffer`	A pointer to the record buffer. Typically, buffer is a pointer to a buffer of type exit_rec_buf_def. The exit_rec_buf_def buffer contains the actual record about to be processed by Extract or Replicat. The buffer is supplied when the call type is EXIT_CALL_DISCARD_RECORD. Exit routines can change the contents of this buffer, for example, to perform custom mapping functions. The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.
`length`	The returned length of the record buffer.

`io_type`	Returned as one of the following:
	DDL type
	`SQL_DDL_VAL`
	DML types
	`DELETE_VAL`
	`INSERT_VAL`
	`UPDATE_VAL`
	Compressed Enscribe update
	`UPDATE_COMP_ENSCRIBE_VAL`
	Compressed SQL update
	`UPDATE_COMP_SQL_VAL`
	`UPDATE_COMP_PK_SQL_VAL`
	Other
	`TRUNCATE_TABLE_VAL`

`mapped`	A flag (0 or 1) indicating whether or not this is a mapped record buffer.

Output	Description
before_after_ind	One of the following to indicate whether the record is a before or after image. BEFORE_IMAGE_VAL AFTER_IMAGE_VAL

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_RECORD_LENGTH

Valid for Extract and Replicat

Use the GET_RECORD_LENGTH function to retrieve the length of the data record. As an alternative, you can use the GET_RECORD_BUFFER function to retrieve the length of the data record.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_RECORD_LENGTH, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input Can be the following:

Input	Description
source_or_target	One of the following indicating whether to return the record length for the source or target data record. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL

Output Can be the following:

Output	Description
`length`	The returned length of the data record.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_RECORD_TYPE

Valid for Extract and Replicat

Use the GET_RECORD_TYPE function to retrieve the type of record being processed. The record can be either a SQL or Enscribe record. The record type is important when manipulating the record buffer, because each record type has a different format.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_RECORD_TYPE, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input Can be the following:

Input	Description

`source_or_target`	One of the following indicating whether or not to return the record type for the source or target data record.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description

`record_type`	The returned record type. Can be one of the following:
	For SQL records:
	`EXIT_REC_TYPE_SQL`
	For Enscribe records:
	`EXIT_REC_TYPE_ENSCRIBE`

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_SCHEMA_NAME_ONLY

Valid for Extract and Replicat

Use the GET_SCHEMA_NAME_ONLY function to retrieve the schema, but not the catalog or name, of the source or target table associated with the record being processed. To return the fully qualified table name, see:

GET_TABLE_NAME

To return other parts of the table name, see:

GET_TABLE_NAME_ONLY

GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_TABLE_NAME, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input Can be the following:

Input	Description
buffer	A pointer to a buffer to accept the returned schema name. The name is null-terminated.
max_length	The maximum length of your allocated buffer to accept the schema name. This is returned as a NULL terminated string.
source_or_target	One of the following indicating whether to return the source or target schema name. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL

Output Can be the following:

Output	Description
buffer	The fully qualified, null-terminated schema name. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the schema name is interpreted in the session character set.
actual length	The string length of the returned name. The actual length does not include the null terminator.
value_truncated	A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the schema name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_SESSION_CHARSET

Valid for Extract and Replicat

Use GET_SESSION_CHARSET to get the current user exit session character set. This character set can be set through callback function SET_SESSION_CHARSET. The character set of the user exit session indicates the encoding of any character-based callback structure members that are used between the user exit and the caller process (Extract, data pump, Replicat), including metadata such as (but not limited to):

- database names and locales
- table and column names
- DDL text

- error messages
- character-type columns such as CHAR and NCHAR
- date-time and numeric columns that are represented in string form

The valid values of the session character set are defined in the header file ucharset.h. This function can be called at any time that the user exit has control.

Syntax

```
#include    usrdecs.h
short result_code;
session_def session_charset_def;
ERCALLBACK (GET_SESSION_CHARSET, &session_charset_def, &result_code);
```

Buffer

```
typedef struct
{
ULibCharSet  session_charset;
} session_def;
```

Input None

Output `session_charset_def.session_charset`

Return Values `EXIT_FN_RET_OK`

GET_STATISTICS

Valid for Extract and Replicat

Use the GET_STATISTICS function to retrieve the current processing statistics for the Extract or Replicat process. For example, the user exit can output statistics to a custom report should a fatal error occur during Extract or Replicat processing.

Statistics are automatically handled based on which process type has requested the data:

- The Extract process will always treat the request as a source table, counting that table once regardless of the number of times output.
- The Replicat process will always treat the request as a set of target tables. The set includes all counts to the target regardless of the number of source tables.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include “usrdecs.h”
short result_code;
statistics_def statistics;
ERCALLBACK (GET_STATISTICS, &statistics, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
short group;
exit_timestamp_string start_datetime;
long num_inserts;
long num_updates;
long num_befores;
long num_deletes;
long num_discards;
long num_ignores;
long total_db_operations;
long total_operations;
/* Version 2 CALLBACK_STRUCT_VERSION */
long num_truncates;
} statistics_def;
```

Input Can be the following:

Input	Description	
table_name	A null-terminated string specifying the source table name. Statistics are always recorded against the source records. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the table name and the date are interpreted in the session character set.	
group	Can be one of the following:	
	EXIT_STAT_GROUP_STARTUP	Retrieves statistics since the Oracle GoldenGate process was last started.
	EXIT_STAT_GROUP_DAILY	Retrieves statistics since midnight of the current day.
	EXIT_STAT_GROUP_HOURLY	Retrieves statistics since the start of the current hour.
	EXIT_STAT_GROUP_RECENT	Retrieves statistics since the statistics were reset using GGSCI.
	EXIT_STAT_GROUP_REPORT	Retrieves statistics since the last report was generated.
	EXIT_STAT_GROUP_USEREXIT	Retrieves statistics since the last time the user exit reset the statistics with RESET_USEREXIT_STATS.

Output Can be the following:

Output	Description
start_datetime	A null-terminated string in the format of YYYY-MM-DD HH:MI:SS indicating the local date and time that statistics started to be recorded for the specified group.
num_inserts	The returned number of inserts processed by Extract or Replicat.
num_updates	The returned number of updates processed by Extract or Replicat.
num_befores	The returned number of update before images processed by Extract or Replicat.
num_deletes	The returned number of deletes processed by Extract or Replicat.
num_discards	The returned number of records discarded by Extract or Replicat.
num_ignores	The returned number of records ignored by Extract or Replicat.
total_db_operations	The returned number of total database operations processed by Extract or Replicat.
total_operations	The returned number of total operations processed by Extract or Replicat, including discards and ignores.
num_truncates	The returned number of truncates processed by Extract or Replicat.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_TABLE_NOT_FOUND
EXIT_FN_RET_OK
```

GET_TABLE_COLUMN_COUNT

Valid for Extract and Replicat

Use the GET_TABLE_COLUMN_COUNT function to retrieve the total number of columns in a table, including the number of key columns.

Syntax

```
#include “usrdecs.h”
short result_code;
table_def table;
ERCALLBACK (GET_TABLE_COLUMN_COUNT, &table, &result_code);
```

Buffer

```
typedef struct
{
short num_columns;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
short num_key_columns;
} table_def;
```

Input Can be the following:

Input	Description

`source_or_target`	One of the following indicating whether to return the total number of columns for the source or target table.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description
`num_columns`	The returned total number of columns in the specified table.
`num_key_columns`	The returned total number of columns that are being used by Oracle GoldenGate as the key for the specified table.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_TABLE_METADATA

Valid for Extract and Replicat

Use the GET_TABLE_METADATA function to retrieve metadata about the table that associated with the record that is being processed.

Syntax

```
#include "usrdecs.h"
short result_code;
table_metadata_def tbl_meta_rec;
ERCALLBACK (GET_TABLE_METADATA, &tbl_meta_rec, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
short value_truncated;
long max_name_length;
long actual_name_length;
short num_columns;
short num_key_columns;
short *key_columns;
short num_keys_returned;
BOOL using_pseudo_key;
short source_or_target;
} table_metadata_def;
```

Input Can be the following:

Input	Description
table_name	A pointer to a buffer to accept the table_name return value
key_columns	A pointer to an array of key_columns indexes.
max_name_length	The maximum length of the returned table name. Typically, the maximum length is the length of the table name buffer. Since the returned table name is null-terminated, the maximum length should equal the maximum length of the table name.
source_or_target	One of the following indicating whether to return the source or target table name. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL

Output Can be the following:

Output	Description
table_name	The name of the table associated with the record that is being processed. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the table name is interpreted in the session character set.
value_truncated	A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the table name plus the null terminator exceeds the maximum buffer length.
actual_name_length	The string length of the returned table name. The actual length does not include the null terminator.
num_columns	The number of columns in the table.

Output	Description
num_key_columns	The number of columns in the key that is being used by Oracle GoldenGate.
key_columns	The values for the key columns. You must know the expected number of keys multiplied by the length of the columns, and then allocate the appropriate amount of buffer.
num_keys_returned	The number of key columns that are requested.
using_pseudo_key	A flag that indicates whether or not KEYCOLS-specified columns are being used as a key. Returns TRUE or FALSE.

Return Values

```
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_EXCEEDED_MAX_LENGTH
EXIT_FN_RET_OK
```

GET_TABLE_NAME

Valid for Extract and Replicat

Use the GET_TABLE_NAME function to retrieve the fully qualified name of the source or target table associated with the record being processed. To return only part of the fully qualified name, see also:

GET_TABLE_NAME_ONLY

GET_SCHEMA_NAME_ONLY

GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include “usrdecs.h”
short result_code;
env_value_def env_value;
ERCALLBACK (GET_TABLE_NAME, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input Can be the following:

Input	Description
`buffer`	A pointer to a buffer to accept the returned table name. The table name is null-terminated.
`max_length`	The maximum length of your allocated buffer to accept the table name. This is returned as a NULL terminated string.

`source_or_target`	One of the following indicating whether to return the source or target table name.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description
`buffer`	The fully qualified, null-terminated table name, for example schema.table or catalog.schema.table, depending on the database platform. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the table name is interpreted in the session character set.
`actual length`	The string length of the returned table name. The actual length does not include the null terminator.
`value_truncated`	A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the table name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_TABLE_NAME_ONLY

Valid for Extract and Replicat

Use the GET_TABLE_NAME_ONLY function to retrieve the name, but not the catalog or schema, of the source or target table associated with the record being processed. To return the fully qualified name, see:

GET_TABLE_NAME

To return other parts of the table name, see:

GET_SCHEMA_NAME_ONLY

GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include “usrdecs.h”
short result_code;
env_value_def env_value;
ERCALLBACK (GET_TABLE_NAME, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input Can be the following:

Input	Description
`buffer`	A pointer to a buffer to accept the returned table name. The table name is null-terminated.
`max_length`	The maximum length of your allocated buffer to accept the table name. This is returned as a NULL terminated string.

`source_or_target`	One of the following indicating whether to return the source or target table name.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output Can be the following:

Output	Description
`buffer`	The fully qualified, null-terminated table name, for example schema.table or catalog.schema.table, depending on the database platform. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the table name is interpreted in the session character set.
`actual length`	The string length of the returned table name. The actual length does not include the null terminator.
`value_truncated`	A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the table name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_TIMESTAMP

Valid for Extract and Replicat

Use the GET_TIMESTAMP function to retrieve the I/O timestamp associated with a source data record in ASCII datetime format. The timestamp is then converted to local time and approximates the time of the original database operation.

NOTE The ASCII commit timestamp can vary with the varying regional use of Daylight Savings Time. The user exit callback should return the ASCII datetime as a GMT time to avoid this variance. The Oracle GoldenGate trail uses GMT format. See GET_GMT_TIMESTAMP.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_TIMESTAMP, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input None

Output Can be the following:

Output	Description
`timestamp`	The returned 64-bit I/O timestamp in ASCII format.
`io_datetime`	A null-terminated string containing the local I/O date and time, in the format of: YYYY-MM-DD HH:MI:SS.FFFFFF

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_TRANSACTION_IND

Valid for Extract and Replicat

Use the GET_TRANSACTION_IND function to determine whether a data record is the first, last or middle operation in a transaction. This can be useful when, for example, a user exit can compile the details of each transaction and output a special summary record.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_TRANSACTION_IND, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input None

Output Can be the following:

Output	Description	
transaction_ind	The returned transaction indicator, represented as one of the following:	
	BEGIN_TRANS_VAL	The record is the beginning of a transaction.
	MIDDLE_TRANS_VAL	The record is in the middle of a transaction.
	END_TRANS_VAL	The record is the end of a transaction.
	WHOLE_TRANS_VAL	The record is the only one in the transaction.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_USER_TOKEN_VALUE

Valid for Extract and Replicat

Use the GET_USER_TOKEN_VALUE function to obtain the value of a user token from a trail record. No character-set conversion is performed on the token value.

Syntax

```
#include "usrdecs.h"
```

Buffer

```
typedef struct
{
char *token_name;
char *token_value;
long max_length;
long actual_length;
short value_truncated;
} token_value_def;
```

Input Can be one of the following:

Input	Description
`token_name`	A pointer to a buffer representing the name of a token. It is assumed that the token name is encoded in the default character set of the operating system that hosts the Extract TABLE statement where the token is configured. The user exit prepares the token name in the character set that is specified with SET_SESSION_CHARSET, but converts it back to the operating system character set before retrieving the matching token value.
`max_length`	The maximum length of your allocated token_name buffer to accept any resulting token value. This is returned as a NULL terminated string.

Output Can be the following.

Input	Description
`token_value`	A pointer to a buffer representing the return value (if any) of a token. The token value is passed back to the user exit as-is, without any character-set conversion.
`actual_length`	The actual length of the token value that is returned. A value of 0 is returned if the token is found and there is no value present.
`value_truncated`	A flag of either 0 or 1 that indicates whether or not the token value was truncated. Truncation occurs if the length of the table name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_TOKEN_NOT_FOUND
EXIT_FN_RET_OK
```

OUTPUT_MESSAGE_TO_REPORT

Valid for Extract and Replicat

Use the OUTPUT_MESSAGE_TO_REPORT function to output a message to the report file. If a character session for the user exit is set with SET_SESSION_CHARSET, the message is interpreted in the session character set but is converted to the default character set of the operating system before being written to the report file.

Syntax

```
#include “usrdecs.h”
short result_code;
char  message[500];
ERCALLBACK (OUTPUT_MESSAGE_TO_REPORT, message, &result_code);
```

Buffer None

Input Can be the following.

Input	Description
message	A null-terminated string.

Output None

Return Values EXIT_FN_RET_OK

RESET_USEREXIT_STATS

Valid for Extract and Replicat

Use the RESET_USEREXIT_STATS function to reset the EXIT_STAT_GROUP_USEREXIT statistics for the Oracle GoldenGate process since the last call to GET_STATISTICS was processed. This function enables the user exit to control when to reset the group statistics that are returned by the GET_STATISTICS function, but does not permit any of the other statistics to be reset.

Syntax

```
#include "usrdecs.h"
short result_code;
call_callback (RESET_USEREXIT_STATS, NULL, &result_code);
```

Input None

Output None

Return Values None

SET_COLUMN_VALUE_BY_INDEX

Valid for Extract and Replicat

Use the SET_COLUMN_VALUE_BY_INDEX or SET_COLUMN_VALUE_BY_NAME function to modify a single column value without manipulating the entire data record. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (CHAR/VARCHAR2/CLOB, NCHAR/NVARCHAR2/NCLOB), a SQL date/timestamp/interval/number type)
- The column_value_mode indicator is set to EXIT_FN_CNVTED_SESS_CHAR_FORMAT.

Syntax

```
#include “usrdecs.h”
short result_code;
column_def column;
ERCALLBACK (SET_COLUMN_VALUE_BY_INDEX, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input Can be the following:

Input	Description
`column_value`	A pointer to a buffer representing the new column value.
`actual_value_length`	The length of the new column value, in bytes. The actual length should not include the null terminator if the new column value is in ASCII format.
`null_value`	A flag (0 or 1) indicating whether the new column value is null. If the null_value flag is set to 1, the column value in the data record is set to null.
`remove_column`	A flag (0 or 1) indicating whether to remove the column from a compressed update if it exists. This flag should only be set if the operation type for the record is UPDATE_COMP_SQL_VAL, PK_UPDATE_SQL_VAL, or UPDATE_COMP_ENSCRIBE_VAL.
`column_index`	The column index of the new column value to be copied into the data record buffer. Column indexes start at zero.

Input	Description
column_value_mode	Indicates the format of the column value. EXIT_FN_CHAR_FORMAT EXIT_FN_RAW_FORMAT EXIT_FN_CNVTED_SESS_CHAR_FORMAT **EXIT_FN_CHAR_FORMAT** ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type UTF16_BE, which is converted to UTF8.) **Note**: A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of 0 becomes the string terminator. ◆ Dates are in the format CCYY-MM-DD HH:MI:SS.FFFFFF, in which the fractional time is database-dependent. ◆ Numeric values are in their string format. For example, 123.45 is represented as "123.45". ◆ Non-printable characters or binary values are converted to hexidecimal notation. ◆ Floating point types are output as null-terminated strings, to the first 14 significant digits. **EXIT_FN_RAW_FORMAT** Internal Oracle GoldenGate canonical format: This format includes a two-byte null indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type. **EXIT_FN_CNVTED_SESS_CHAR_FORMAT** User exit character set: This only applies if the column data type is: ◆ a character-based type, single or multi-byte ◆ a numeric type with a string representation This format is not null-terminated.
source_or_target	One of the following indicating whether the source or target record is being modified. EXIT_FN_SOURCE_VAL EXIT_FN_TARGET_VAL

Input	Description
`requesting_before_after_ind`	Set when setting a column value on a record io_type of UPDATE_COMP_PK_SQL_VAL (primary key update). Use one of the following to indicate which portion of the primary key update is to be accessed. The default is AFTER_IMAGE_VAL. ◆ BEFORE_IMAGE_VAL ◆ AFTER_IMAGE_VAL

Output None

Return Values

```
EXIT_FN_RET_BAD_COLUMN_DATA
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
EXIT_FN_RET_NOT_SUPPORTED
EXIT_FN_RET_INVALID_COLUMN_TYPE
```

SET_COLUMN_VALUE_BY_NAME

Valid for Extract and Replicat

Use the SET_COLUMN_VALUE_BY_NAME or SET_COLUMN_VALUE_BY_INDEX function to modify a single column value without manipulating the entire data record.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (CHAR/VARCHAR2/CLOB, NCHAR/NVARCHAR2/NCLOB), a SQL date/timestamp/interval/number type)
- The column_value_mode indicator is set to EXIT_FN_CNVTED_SESS_CHAR_FORMAT.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include "usrdecs.h"
short result_code;
column_def column;
ERCALLBACK (SET_COLUMN_VALUE_BY_NAME, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input Can be the following:

Input	Description
`column_value`	A pointer to a buffer representing the new column value.
`actual_value_length`	The length of the new column value, in bytes. The actual length should not include the null terminator if the new column value is in ASCII format.
`null_value`	A flag (0 or 1) indicating whether the new column value is null. If the null_value flag is set to 1, the column value in the data record is set to null.
`remove_column`	A flag (0 or 1) indicating whether to remove the column from a compressed update if it exists. This flag should only be set if the operation type for the record is UPDATE_COMP_SQL_VAL, PK_UPDATE_SQL_VAL, or UPDATE_COMP_ENSCRIBE_VAL.
`column_name`	The name of the column that corresponds to the new column value to be copied into the data record buffer.
`column_value_mode`	Indicates the format of the column value. EXIT_FN_CHAR_FORMAT EXIT_FN_RAW_FORMAT EXIT_FN_CNVTED_SESS_CHAR_FORMAT **EXIT_FN_CHAR_FORMAT** ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type UTF16_BE, which is converted to UTF8.)

Input	Description
	Note: A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of 0 becomes the string terminator. ◆ Dates are in the format CCYY-MM-DD HH:MI:SS.FFFFFF, in which the fractional time is database-dependent. ◆ Numeric values are in their string format. For example, 123.45 is represented as "123.45". ◆ Non-printable characters or binary values are converted to hexidecimal notation. ◆ Floating point types are output as null-terminated strings, to the first 14 significant digits. **EXIT_FN_RAW_FORMAT** Internal Oracle GoldenGate canonical format: This format includes a two-byte null indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type. **EXIT_FN_CNVTED_SESS_CHAR_FORMAT** User exit character set: This only applies if the column data type is: ◆ a character-based type, single or multi-byte ◆ a numeric type with a string representation This format is not null-terminated.

`source_or_target`	One of the following indicating whether the source or the target data record is being modified.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

`requesting_before_after_ind`	Set when setting a column value on a record io_type of UPDATE_COMP_PK_SQL_VAL (primary key update). Use one of the following to indicate which portion of the primary key update is to be accessed. The default is AFTER_IMAGE_VAL. ◆ BEFORE_IMAGE_VAL ◆ AFTER_IMAGE_VAL

Output None

Return Values

```
EXIT_FN_RET_BAD_COLUMN_DATA
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
EXIT_FN_RET_NOT_SUPPORTED
EXIT_FN_RET_INVALID_COLUMN_TYPE
```

SET_OPERATION_TYPE

Valid for Extract and Replicat

Use the SET_OPERATION_TYPE function to change the operation type associated with a data record. For example, a delete on a specified table can be turned into an insert into another table. The record header's before/after indicator is modified as appropriate for insert and delete operations.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (SET_OPERATION_TYPE, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input Can be the following:

Input	Description
io_type	Returned as one of the following for deletes, inserts, and updates, respectively: DELETE_VAL INSERT_VAL UPDATE_VAL For a compressed Enscribe update, the following is returned: UPDATE_COMP_ENSCRIBE_VAL

Input	Description
	For a compressed SQL update, the following is returned: `UPDATE_COMP_SQL_VAL` If the new operation type is an insert or delete, the before/after indicator for the record is set to one of the following: **Insert:** `AFTER_IMAGE_VAL` (after image) **Delete:** `BEFORE_IMAGE_VAL` (before image)

`source_or_target`	One of the following indicating whether to set the operation type for the source or target data record.
	`EXIT_FN_SOURCE_VAL`
	`EXIT_FN_TARGET_VAL`

Output None

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

SET_RECORD_BUFFER

Valid for Extract and Replicat

Use the SET_RECORD_BUFFER function for compatibility with user exits, and for complex data record manipulation. This function manipulates the entire record. It is best to modify individual column values, rather than the entire record, because the Oracle GoldenGate internal record formats must be known in order to accurately modify the data record buffer directly. To modify column values, use the SET_COLUMN_VALUE_BY_INDEX and SET_COLUMN_VALUE_BY_NAME functions. These functions are sufficient to handle most custom mapping within a user exit.

Syntax

```
#include “usrdecs.h”
short result_code;
record_def record;
ERCALLBACK (SET_RECORD_BUFFER, &record_def, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input Can be the following:

Input	Description
`buffer`	A pointer to the new record buffer. Typically, buffer is a pointer to a buffer of type exit_rec_buf_def. The exit_rec_buf_def buffer contains the actual record about to be processed by Extract or Replicat. The buffer is supplied when the call type is EXIT_CALL_DISCARD_RECORD. Exit routines can change the contents of this buffer, for example to perform custom mapping functions. The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.
`length`	The new length of the record buffer.

Output None

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
EXIT_FN_RET_NOT_SUPPORTED
```

SET_SESSION_CHARSET

Valid for Extract and Replicat

Use the SET_SESSION_CHARSET function to set the character set of the user exit. The character set of the user exit session indicates the encoding of any character-based callback structure members that are used between the user exit and the caller process (Extract, data pump, Replicat), including metadata such as (but not limited to):

- database names and locales
- table and column names
- DDL text
- error messages

- character-type columns such as CHAR and NCHAR
- date-time and numeric columns that are represented in string form

This function can be called at any time that the user exit has control. When the user exit sets the session character set, it takes effect immediately, and all character values start being converted to the specified set. The recommended place to call this function is with call type EXIT_CALL_START.

NOTE SET_SESSION_CHARSET is not thread-safe.

If SET_SESSION_CHARSET is not called, the session gets set to the default character set of the operating system, which is a predefined enumerated type value in ULIB_CS_DEFAULT in the ucharset.h file. When the session character set is a default from ULIB_CS_DEFAULT, no conversion is performed by Oracle GoldenGate for character-type values that are exchanged between the user exit and the caller process. In addition, the object-name metadata of the database are considered to be the default character set of the operating system. Keep in mind that the default may not be correct.

The character set of the user exit is printed to the report file when the user exit is loaded and when SET_SESSION_CHARSET is called. If the session character set is ULIB_CS_DEFAULT, there is a message stating that no column data character-set conversion is being performed.

For more information about globalization support, see the Oracle GoldenGate *Windows and UNIX Administrator's Guide*.

Syntax

```
#include usrdecs.h
short result_code;
session_def session_charset_def;
ERCALLBACK (SET_SESSION_CHARSET, &session_charset_def, &result_code);
```

Buffer

```
typedef struct
{
ULibCharSet  session_charset;
} session_def;
```

Input Can be the following:

Input	Description
session_charset	The valid values of the session character set are defined in the header file ucharset.h.

Output None

Return Values EXIT_FN_RET_OK

SET_TABLE_NAME

Valid for Extract and data pumps

Use the SET_TABLE_NAME function to change the table name associated with a data record. For example, a delete on a specified table can be changed to an insert into a history table. You can change the table name only during Extract processing.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include “usrdecs.h”
short result_code;
record_def record;
ERCALLBACK (SET_TABLE_NAME, &record_def, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input Can be the following:

Input	Description
`table_name`	A null-terminated string specifying the new table name to be associated with the data record. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the table name is interpreted in the session character set.

Output None

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

Index