Oracle® GoldenGate

Reference Guide

version 10.4

October 2009

Reference Guide, version 10.4

Contents

PREFACE

About the GoldenGate Guides

The complete GoldenGate documentation set contains the following components:

Oracle® GoldenGate for Mainframe

- *GoldenGate for HP NonStop Administrator Guide:* Introduces GoldenGate components and explains how to plan for, configure, and implement GoldenGate on the NonStop platform.
- *GoldenGate for HP NonStop Reference Guide:* Provides detailed information about GoldenGate parameters, commands, and functions for the NonStop platform.

Windows and UNIX platforms

- *Installation and Setup guides*: There is one such guide for each database that is supported by GoldenGate.
- *GoldenGate for Windows and UNIX Administrator Guide*: Introduces GoldenGate components and explains how to plan for, configure, and implement GoldenGate on the Windows and UNIX platforms.
- *Reference Guide*: Provides detailed information about GoldenGate parameters, commands, and functions for the Windows and UNIX platforms.
- *Troubleshooting and Tuning Guide:* Provides suggestions for improving the performance of GoldenGate in different situations, and provides solutions to common problems.

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>] [, ...])
  ```

New in Version 10.4

Database-specific enhancements

DB2 LUW

- GoldenGate now supports Multi Dimensional Clustered Tables (MDC) for DB2 LUW 9.5 and later.
- GoldenGate now supports Materialized Query Tables. GoldenGate does not replicate the MQT itself, but only the base tables. The target database automatically maintains the content of the MQT based on the changes that are applied to the base tables by Replicat.

DB2 z/OS

A new SQLID option was added to DBLOGIN that 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.

Oracle

- GoldenGate now supports Oracle 10.2 running on zLinux SUSE on S/390. A new LOGSOURCE target of S390 has been added to TRANLOGOPTIONS.
- For Oracle versions 10g and later, you can now use a simple parameter statement instead of a trace table to prevent Replicat transactions from being captured in a bi-directional configuration. This enhancement eliminates the overhead of creating and writing to the trace table. The new EXCLUDEUSER and EXCLUDEUSERID options of TRANLOGOPTIONS support this enhancement. These options also can be used to exclude the work of other database users, and multiple instances of either parameter can be used. For more information, see the *Reference Guide*. For Oracle versions 9i and earlier, the trace table is still required.

- Concurrent with the EXCLUDEUSER and EXCLUDEUSERID enhancements, @GETENV was enhanced with two new functions: @GETENV ("TRANSACTION", "USERID") returns the user-id for the Oracle user that issued the last committed transaction. @GETENV ("TRANSACTION", "USERNAME") returns the name of the Oracle user that issued the last committed transaction.
- GoldenGate now supports Oracle Spatial objects, object tables, Oracle Multimedia ORDDicom (DICOM), and XMLType stored as an object. A new XMLBUFSIZE option to DBOPTIONS 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.
- GoldenGate now supports cluster tables.
- GoldenGate now supports the capture and replication of Oracle DDL statements of up to 2 MB in length (including some internal GoldenGate maintenance information). Extract will skip statements that are greater than the supported length, but a new ddl_ddl2file.sql script can be used to save the skipped DDL to a text file in the USER_DUMP_DEST directory of Oracle.

  NOTE To use the new support, the DDL trigger must be reinstalled in INITIALSETUP mode, which removes all of the DDL history. Follow the procedure for restoring an existing DLL environment to a clean state in the Oracle Installation and Setup Guide.

- A new INSERTAPPEND and NOINSERTAPPEND parameter set controls whether or not Replicat uses an APPEND hint when it applies inserts to Oracle target tables. INSERTAPPEND is appropriate for use as a performance improvement when the replicated transactions are large and contain multiple inserts into the same table. 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.
- Schema name translation for DDL operations on objects of UNMAPPED scope was revised. When Oracle DDL is of UNMAPPED scope in the Replicat configuration, Replicat will set the current owner of the Replicat session to the owner of the source DDL object, then execute the DDL as that owner, then restore Replicat as the current owner of the Replicat session.

SQL Server

GoldenGate now supports delivery to a SQL Server 2008 target. See the GoldenGate for SQL Server Installation and Setup Guide for more information.

Teradata

Database name translation for DDL operations on objects of UNMAPPED scope was revised. When Teradata DDL is of UNMAPPED scope in the Replicat configuration, it is applied to the target in one of these ways:

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

Other enhancements

General

- GoldenGate now supports the replication of DDL statements that contain a space before or after the owner and object names (or both before and after). For example, fin . customers.

Column Conversion Functions

- A new OSVARIABLE option of @GETENV returns the string value of a specified operating-system environment variable. For example, @GETENV ("OSVARIABLE", "HOME") would return the value of the HOME variable. Because the function only returns an exact match, other variables that contain "HOME," such as ORACLE_HOME, would not be returned. This option is valid for Extract and Replicat.

Parameters

- A new TRANSACTIONTIMEOUT parameter for Replicat 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. TRANSACTIONTIMEOUT helps 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.
- A new GLOBALS parameter OUTPUTFILEUMASK specifies an octal umask that will be used by GoldenGate processes to create trail files and discard files. This parameter is not valid for WIN32 systems.
- The default for CACHESIZE is now 8GB for 64-bit systems and 2GB for 32-bit systems.

Commands

- The SEND EXTRACT <group> STATUS command now makes it clear when Extract is going through a recovery after an abend event. This feature is especially useful when a very long-running transaction was open at the point of failure and Extract must search far back into the logs to find the begin-transaction record. The new status messages appear on the Current status line and look as follows:
  - In recovery[1] – Extract is recovering to its input checkpoint.
  - In recovery[2] – Extract is recovering to its output checkpoint.
  - Recovery complete – The recovery is finished, and normal processing will resume.
- To support TRANSACTIONTIMEOUT, the SEND REPLICAT command with STATUS has two more status conditions:
  - 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.

Documentation enhancements and corrections

The documentation for the @GETENV options DBTRANSACTION and DBRECORD was changed to

reflect the accurate names of TRANSACTION and RECORD. Various other minor errors were corrected in this section.

The FORMAT option was added to the RMTTASK documentation. The same versioning concept that applies to trails and files also applies to the data format that is sent by Extract to a remote process.

The default for AUTORESTART RESETMINUTES was changed to the correct value of 20 minutes.

CHAPTER 1

GGSCI Commands

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

Command Summary

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 GoldenGate to replicate UPDATE operations.
Checkpoint table commands	Create and manage the 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 GoldenGate's parent process and is responsible for the management of GoldenGate processes and files, resources, user interface, and the reporting of thresholds and errors.

Command summary

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
```

REFRESH MANAGER

Use REFRESH MANAGER to execute the Manager parameter file. This command enables you to change any Manager parameter except the port number without stopping and restarting the Manager process.

To change the Manager port number, first stop Manager with the STOP MANAGER command (or stop it from the Cluster Administrator, if using a Windows cluster). Edit the Manager parameter file to change the port number and any other parameters as needed, and then start Manager again with the START MANAGER command.

Syntax

```
REFRESH 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 have been allocated by Manager. (Requires ports to be specified with the DYNAMICPORTLIST parameter.)
GETPORTINFO [DETAIL]	By default, retrieves the current list of ports that have been allocated by Manager and their corresponding process IDs. (Requires ports to be specified with the DYNAMICPORTLIST parameter.) DETAIL provides additional details, including all ports listed with DYNAMICPORTLIST and which ones are assigned to a process.
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 254.

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         2006/01/23 21:08:32    7840
2    ORAEXT2   2450      0       None         2006/01/23 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      2006-01-01
1        8001    0                 10:30:23
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\GGSLOG                  735275        1        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 confirming 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

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 *GoldenGate for Windows and UNIX Administrator Guide* before creating an Extract group.

The GoldenGate GGSCI command interface fully supports up to 300 concurrent Extract and Replicat groups per instance of GoldenGate Manager. At the supported level, all groups can be controlled and viewed in full with GGSCI commands such as the INFO and STATUS commands. Beyond the supported level, group information is not displayed and errors can occur. GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at 300 or below in order to manage your environment effectively.

Syntax For a regular, passive, or data pump Extract

```
ADD EXTRACT <group name>
{, SOURCEISTABLE |
   , TRANLOG [<bsds name>] |
   , 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> |
   , 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>} | {PORT <port}
[, RMTNAME <name>]
[, DESC “<description>”]
```

Argument	Description
<group name>	The name of the Extract group. Use the following naming conventions.

Argument	Description
	◆ You can use up to eight ASCII characters, including nonalphanumeric characters such as the underscore (_). Any ASCII character can be used, so long as the operating system allows that character to be in a filename. This is because a group is identified by its associated checkpoint file. ◆ The following ASCII characters are not allowed in a file name: `{ \ / : * ? " < > | }` ◆ On HP UX, Linux, and Solaris, it is possible to create a file name with a colon (:) or an asterisk (*), although it is not best practice. ◆ Group names are not case-sensitive. ◆ Use only one word. ◆ Do not use the word "port" as a group name. However, you can use the string "port" as part of the group name. ◆ Do not place a numeric value at the end of a group name, such as fin1, fin10, and so forth. You can place a numeric value at the beginning of a group name, such as 1_fin, 1fin, and so forth. **Example 1** ext_1 **Example 2** ex+2t **Example 3** ex!2t
Data source options	
`SOURCEISTABLE`	Creates an Extract task that extracts entire records from the database for an initial load using the 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 service options. Task parameters must be specified in the parameter file. For more information about initial load methods, see the *GoldenGate for Windows and UNIX Administrator Guide*.
`TRANLOG [<bsds name>]`	Specifies the transaction log as the data source. Use this option for log-based extraction. TRANLOG requires the BEGIN option. Use the <bsds name> option for DB2 on a z/OS system to specify the BSDS (Bootstrap Data Set) 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*. GoldenGate does not perform any validations of the BSDS specification.
`VAM`	Indicates that a Vendor Access Module will be used to extract data from the data source.

Argument	Description
`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 fully qualified path name of the file, for example c:\ggs\dirdat\extfile.
`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 fully qualified path name of the trail, for example c:\ggs\dirdat\aa.
`VAMTRAILSOURCE <VAM trail name>`	Specifies a VAM trail. Use this option when using VAM-based extraction for Teradata in maximum protection mode. Specify the 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.
Service options	
`BEGIN <begin spec>`	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 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.

Argument	Description
`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 GoldenGate Technical Support before using this option.
`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.
`LOGNUM <log number>, LOGPOS <byte offset>`	Valid for c-tree. Specifies the location in a c-tree transaction log at which to start capturing data. ◆ <log number> is the number of the c-tree log file. ◆ <byte offset> is the relative position from the beginning of the file (0 based).
`LSN <value>`	Valid for SQL Server or Ingres. Specifies the LSN in a SQL Server or Ingres transaction log at which to start capturing data. The LSN specified should exist in a log backup or the online log. An alias for this option is EXTLSN. **SQL Server** 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.

Argument	Description
	Ingres For Ingres, an LSN is two, 4-byte unsigned integers, separated by a colon. For example, to specify an LSN of 1206396546,43927 (as viewed in an Ingres utility), you would enter 1206396546:43927.
`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. ◆ 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.
`PARAMS <parameter file>`	Specifies an Extract parameter file in a location other than the default of dirprm within the GoldenGate directory. Use the fully qualified path name.
`REPORT <report file>`	Specifies an Extract report file in a location other than the default of dirrpt within the GoldenGate directory. Use the fully qualified path name.
`THREADS <n>`	Specifies the number of redo threads when extracting data from an Oracle RAC clustered database configuration.
`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 *GoldenGate for Windows and UNIX Administrator 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`.

Argument	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>} | {PORT <port>}`	Use for an alias Extract. Specify one of the following: ◆ MGRPORT specifies the port on the remote system where Manager is running. Use this option when using a dynamic Collector. ◆ PORT specifies a static Collector port. Use instead of MGRPORT only if running a static Collector.
`RMTNAME <name>`	Use for an alias Extract. Specifies the passive Extract name, if different from that of the alias Extract.

Example 4 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 5 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 6 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 31, 2006.

```
ADD EXTRACT finance, TRANLOG, BEGIN 2006-01-31 08:00
```

Example 7 The following creates an Extract group named "finance" that interfaces with a VAM in either maximum performance or maximum protection mode. No BEGIN point is used for Teradata sources.

```
ADD EXTRACT finance, VAM
```

Example 8 The following creates a VAM-sort Extract group named "finance." The process reads from the VAM trail /ggs/dirdat/vt.

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

Example 9 The following creates a data-pump Extract group named "finance." It reads from the GoldenGate trail c:\ggs\dirdat\lt.

```
ADD EXTRACT finance, EXTTRAILSOURCE c:\ggs\dirdat\lt
```

Example 10 The following creates an initial-load Extract named "load."

```
ADD EXTRACT load, SOURCEISTABLE
```

Example 11 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 12 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.

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

Syntax

```
ALTER EXTRACT <group name>
[, <ADD EXTRACT attribute>]
[, THREAD <number>]
[, ETROLLOVER]
```

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

`<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, GoldenGate recommends deleting the Extract group and then adding it again.
	If using the BEGIN option, do not combine other options in the statement. Issue separate statements, for example:
	`ALTER EXTRACT finance, BEGIN 2006-01-01`
	`ALTER EXTRACT finance, ETROLLOVER`

`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.

Example 1 The following alters Extract to start processing data from January 1, 2006.

```
ALTER EXTRACT finance, BEGIN 2006-01-01
```

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 YYYY-MM-DD
```

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

```
ALTER EXTRACT sales, LSN 1234:123:1
```

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

```
ALTER EXTRACT finance, ETROLLOVER
```

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 but leaves the parameter file intact. You can then re-create the group or delete the parameter file as needed.

To delete associated trail files, delete them manually through the operating system.

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

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.

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 2006-01-05 16:09 Status RUNNING
Checkpoint Lag          00:01:30 (updated 97:16:45 ago)
Log Read Checkpoint File /rdbms/data/oradata/redo03a.log
                         2006-01-05 16:05:17  Seqno 2952, RBA 7598080
```

About time lag

Time 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 33).

Showing detail

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

Figure 2 INFO EXTRACT with DETAIL

```
EXTRACT    ORAEXT    Last Started 2006-03-15 16:16   Status STOPPED
Checkpoint Lag       00:00:00 (updated 114:24:48 ago)
Log Read Checkpoint  File C:\ORACLE\ORADATA\ORA920\REDO03.LOG
                     2006-03-15 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      2006-03-15 16:07  2006-03-15
16:17
C:\ORACLE\ORADATA\ORA920\REDO03.LOG      2006-03-15 15:55  2006-03-15
16:07
C:\ORACLE\ORADATA\ORA920\REDO03.LOG      2006-03-15 15:42  2006-03-15
15:55
C:\ORACLE\ORADATA\ORA920\REDO03.LOG      2006-03-15 15:42  2006-03-15
15:42
  Not Available                        * Initialized *   2006-03-15 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 GoldenGate 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, SHOWCH

```
EXTRACT    JC108XT Last Started 2006-06-09 14:15   Status ABENDED
Checkpoint Lag       00:00:00 (updated 00:00:01 ago)
Log Read Checkpoint  File /orarac/oradata/racq/redo01.log
          2006-06-09 14:16:45  Thread 1, Seqno 47, RBA 68748800
Log Read Checkpoint  File /orarac/oradata/racq/redo04.log
          2006-06-09 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: 2006-06-09 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: 2006-06-09 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: 2006-06-09 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: 2006-06-09 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: 2006-06-09 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: 2006-06-09 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: 2006-06-09 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 = 2006-06-09 14:15:14
   Last Update Time = 2006-06-09 14:16:50
   Stop Status = A
   Last Result = 400
```

Table 1 About Extract checkpoints

Checkpoint	Description	
Read Checkpoint	The following describes the Extract read checkpoints. Extract places a read checkpoint in the data source.	
Startup Checkpoint	The first checkpoint made in the data source when the process started. Comprising this statistic are:	
	Thread #	The number of the Extract thread that made the checkpoint, if 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.

Table 1 About Extract checkpoints (continued)

Checkpoint	Description	
	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 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 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.	
Write Checkpoints	The following describes an Extract write checkpoint. Extract places a write checkpoint in the GoldenGate trail.	
Current Checkpoint	The position in the trail where Extract is currently writing. 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 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 output	The Header, File Information, Configuration, and Status statistics at the end of the SHOWCH display are for use by GoldenGate support specialists. They contain internal information that is useful when resolving a support case.	

Syntax

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

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.

Argument	Description
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.
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.

Example 1 `INFO EXTRACT fin*, SHOWCH`

Example 2 `INFO EXTRACT *, TASKS`

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*
```

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>, {
CACHEMGR {CACHESTATS | CACHEQUEUES | CACHEPOOL} |
FORCESTOP |
FORCETRANS <ID> [THREAD <n>] [FORCE] |
GETLAG |
GETTCPSTATS |
LOGEND |
REPORT |
ROLLOVER |
SHOWTRANS [<ID>] [THREAD <n>] [COUNT <n>]
   [DURATION <duration><unit>] [TABULAR]
   [FILE <name> [DETAIL]] |
SKIPTRANS <ID> [THREAD <n>] [FORCE]
STATUS |
STOP |
TLTRACE {DEBUG | OFF | <level>} [DDLINCLUDE | DDL[ONLY]] [FILE] <file name> |
TRACE[2] <tracefile> |
TRACE[2] OFF |
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.
`CACHEMGR {CACHESTATS | CACHEQUEUES | CACHEPOOL}`	Retrieves statistics about the GoldenGate memory cache manager. ◆ CACHESTATS returns statistics about the overall utilization of virtual memory, such as that which is being used, that which has been recycled for possible reassignment, that which is in anonymous memory, and that which corresponds to paged virtual memory. Also provided are the size of the cache and the thresholds related to buffer allocation and paging, and statistics related to the cache files on disk and their related I/O. ◆ CACHEQUEUES returns statistics about the queues, which are memory buffers that have been recycled for future use. Of particular interest are a statistic that shows the number of times a buffer of the required size was found, and the number of times that such a buffer could not be found, resulting in a new request for more virtual memory. ◆ CACHEPOOL returns statistics about the cache sub-pools, which consist of one sub-pool per log reader thread (one or more in Oracle RAC; one otherwise) for most transaction row data, and another sub-pool for BLOB data and possibly other large items. The actual allocation of data to one or the other of these pool types might not always correspond to the type of data that is being stored. These statistics can be used as the basis for adjusting the CACHEMGR parameter. However, there is generally no need to view these statistics or change CACHEMGR because the cache manager is self-adjusting. Do not make any adjustments to the memory cache manager without the guidance of a GoldenGate support analyst.
`FORCESTOP`	Forces Extract to stop, bypassing any notifications. This command will stop the process immediately.
`FORCETRANS <ID> [THREAD <n>] [FORCE]`	(Oracle only) Forces Extract to write a transaction specified by its ID number to the trail as a committed transaction. Get the ID number with SHOWTRANS or from an Extract runtime message. Extract will ignore any data added to the transaction after this command is issued.

Argument	Description
	Options: ◆ Use THREAD <n> to specify which thread generated the transaction in a 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. In order to use FORCETRANS, the transaction specified 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. 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 may see the following message: `This transaction cannot be forced to extract trail at this moment. Please try again in a few minutes.` 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 33).

Argument	Description
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. ◆ 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). If compression is enabled, the following statistics are present: ◆ Compression CPU time: The amount of CPU used to perform the compression. ◆ Compress time: total amount of time that compression took, such as waits for CPU resources. ◆ 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.
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 296.

Argument	Description
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.
SHOWTRANS [<ID>] [THREAD <n>] [COUNT <n>] [DURATION <duration><unit>] [TABULAR] | [FILE <name> [DETAIL]]	(Oracle only) Displays information about open transactions. SHOWTRANS shows: ◆ Process checkpoint (indicating 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 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. Without options, SHOWTRANS displays all open transactions that will fit into the 25K buffer. To further control output, see the following options. SHOWTRANS options: ◆ <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 thread number 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.

Argument	Description

	◆ 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`

	◆ 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.
	To write a hex and plain character dump of the data, use FILE with DETAIL. 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. For additional information about SHOWTRANS, see the *GoldenGate for Windows and UNIX Administrator Guide*.

`SKIPTRANS <ID>`
`[THREAD <n>]`
`[FORCE]`
	(Oracle only) Forces Extract to skip the specified transaction, thereby removing any current data from memory and ignoring any subsequent data. Get the ID number with SHOWTRANS or from an Extract runtime message.
	Use THREAD <n> to specify which thread generated the transaction in a RAC environment if there are duplicate transaction IDs.
	Use FORCE to bypass the confirmation prompt.
	Note: In order to use SKIPTRANS, the transaction specified 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.
	After using SKIPTRANS, wait at least five minutes if you intend to issue SEND EXTRACT with FORCESTOP. Otherwise, the transaction will still be present.

Argument	Description
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 ◆ 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 either the source table, transaction log, or VAM interface ◆ Adding record to transaction list – adding a record to the file memory transaction list ◆ At EOF (end of file) – no more records to process In addition to the preceding statuses, the following status notations appear during an Extract recovery after an abend event. They tell you the recovery stage that Extract is in, which is especially useful if the recovery is long (usually because there was a long-running transaction when the abend occurred). 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. This can take a long time if Extract must go far back into past transaction logs to find the location of the last read checkpoint. ◆ In recovery[2] – Extract is recovering from its checkpoint to the end of the GoldenGate trail. ◆ Recovery complete – The recovery is finished, and normal processing will resume. The following is output for each database type. **c-tree log-based extraction** ◆ Group name and process ID ◆ Processing status ◆ Timestamp **DB2 LUW** ◆ Group name and process ID ◆ Processing status ◆ LSN ◆ Timestamp

Argument	Description
	DB2 on z/OS log-based extraction ◆ Group name and process ID ◆ Processing status ◆ Log RBA ◆ Timestamp ◆ BSDS **Oracle and Oracle RAC log-based extraction** ◆ 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 log-based extraction** ◆ Group name and process ID ◆ Processing status ◆ Timestamp **Teradata log-based extraction, primary Extract** ◆ Group name and process ID ◆ Processing status ◆ Timestamp **Teradata log-based extraction, 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

Argument	Description

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.`

TLTRACE {DEBUG | OFF | <level>}
[SIZELIMIT <size>]
[DDLINCLUDE | DDL[ONLY]]
[FILE] <file name>
	(Valid for Oracle) Turns detailed tracing of Oracle transaction log activity on or off.
	The <level> option specifies the detail of the trace, expressed as an integer between 1 and 99. Trace information is sent to the report file. This option is for debugging and should be used with guidance from GoldenGate Technical Support.
	
	SIZELIMIT controls the size, in bytes, of the trace file; without it, the size of the file is unlimited. Valid size values are from 10240 to the maximum file size that is permitted by the operating system.
	
	DDLINCLUDE | DDLONLY
	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 fully qualified name of a file to which 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 EXTRACT <group> TLTRACE FILE <file name> DDLINCLUDE`
	If no other options will follow the file name, the FILE keyword can be omitted, for example:
	`SEND EXTRACT <group> TLTRACE DDLINCLUDE <file name>`

Argument	Description
`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 GoldenGate Technical Support for assistance if the trace reveals significant processing bottlenecks.
`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 344.
`TRANLOGOPTIONS TRANSCLEANUPFREQUENCY <minutes>`	Valid for Oracle RAC. Specifies the interval, in minutes, after which 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 344.

`VAMMESSAGE "<Teradata command>"`
 	
`VAMMESSAGE { "ARSTATS" | "INCLUDELIST [filter]" | "EXCLUDELIST [filter]" }`
 	
`VAMMESSAGE "OPENTRANS"`
	Sends a command to a Vendor Access Module (VAM).
	<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 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.

Argument	Description
	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. ◆ "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 `SEND EXTRACT finance, SHOWTRANS COUNT 10`

Example 6 `SEND EXTRACT finance, SKIPTRANS 5.17.27634 THREAD 2`

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 *GoldenGate for Windows and UNIX Administrator 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 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 2006-06-08 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 2006-06-08 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 2006-06-08 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 GoldenGate. DB2 does not log update statements if they do not physically change a row, so GoldenGate cannot detect them or include them in statistics.

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 296.
`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.

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 will 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 GoldenGate trail. See page 34.

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`

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 *GoldenGate for Windows and UNIX Administrator Guide*.

The GoldenGate GGSCI command interface fully supports up to 300 concurrent Extract and Replicat groups per instance of GoldenGate Manager. At the supported level, all groups can be controlled and viewed in full with GGSCI commands such as the INFO and STATUS commands. Beyond the supported level, group information is not displayed and errors can occur. GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at 300 or below in order to manage your environment effectively.

Syntax

```
ADD REPLICAT <group name>
{, SPECIALRUN |
   , EXTFILE <full path name> |
   , EXTTRAIL <full path 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. Use the following naming conventions.

Argument	Description
	◆ You can use up to eight ASCII characters, including nonalphanumeric characters such as the underscore (_). Any ASCII character can be used, so long as the operating system allows that character to be in a filename. This is because a group is identified by its associated checkpoint file. ◆ The following ASCII characters are not allowed in a file name: `{ \ / : * ? " < > | }` ◆ On HP UX, Linux, and Solaris, it is possible to create a file name with a colon (:) or an asterisk (*), although it is not best practice. ◆ Group names are not case-sensitive. ◆ Use only one word. ◆ Do not use the word "port" as a group name. However, you can use the string "port" as part of the group name. ◆ Do not place a numeric value at the end of a group name, such as fin1, fin10, and so forth. You can place a numeric value at the beginning of a group name, such as 1_fin, 1fin, and so forth. **Example 1** ext_1 **Example 2** ex+2t **Example 3** ex!2t
`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 it with the START REPLICAT command in GGSCI. For more information on when to use special runs, see the *GoldenGate for Windows and UNIX Administrator Guide*.
Data source options	
`EXTFILE <full path name>`	Specifies the fully qualified name of an extract file that is specified with RMTFILE in the Extract parameter file.
`EXTTRAIL <full path name>`	Specifies the fully qualified name of a trail that was created with the ADD RMTTRAIL or ADD EXTTRAIL command.
Service options	
`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 GoldenGate Technical Support before using this option.
EXTRBA <rba>	Specifies the relative byte address within the trail file that is specified by EXTSEQNO. Contact GoldenGate Technical Support before using this option.
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 the 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 the default CHECKPOINTTABLE specification in the GLOBALS file.
PARAMS <parameter file>	Specifies a parameter file in a location other than the default of dirprm within the GoldenGate directory. Use the fully qualified name.
REPORT <report file>	Specifies a process report file in a location other than the default of dirrpt within the GoldenGate directory. Use the fully qualified name.
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 d:\ggs\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.

Argument	Description
`<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 2006-06-07 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 replication 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.

About lag values

The following explains the lag values shown by INFO REPLICAT.

Byte lag is the difference, in bytes, between the read position of Replicat in the trail file at the time of the last checkpoint and the current end-of-file indicator.

Time 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 56).

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 2006-01-21 11:40 Status RUNNING
Checkpoint Lag           00:00:00 (updated 232:39:41 ago)
Log Read Checkpoint File C:\GGS\DIRDAT\RT000000
                         2006-01-11 18:54:33.000000 RBA 4735245

Extract Source           Begin                     End

C:\GGS\DIRDAT\RT000000   2006-01-11 18:54          2006-01-11 18:54
C:\GGS\DIRDAT\RT000000   * Initialized *           2006-01-11 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 2006-06-12 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 = 2006-06-12 13:10:13
   Last Update Time = 2006-06-07 21:23:31
   Stop Status = A
   Last Result = 400
```

Table 2 About Replicat checkpoints

Checkpoint	Description	
Startup Checkpoint	The first checkpoint made in the trail when the process started. 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 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 output	The Header, File Information, Configuration, and Status statistics at the end of the SHOWCH display are for use by GoldenGate support specialists. 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>]] |
STATUS |
STOP |
TRACE[2] [DDLINCLUDE | DDL[ONLY]] [FILE] <file name> |
TRACE[2] OFF |
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.

Argument	Description
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. For more information, see page 193. <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 (*).
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 (*).
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

Argument	Description
	Possible processing status messages are: ◆ Delaying – waiting for more data ◆ 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.
`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 GoldenGate Technical Support for assistance if the trace reveals significant processing bottlenecks. 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.

Argument	Description

	[FILE] <file name>
	The fully qualified name of a file to which 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.
`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 point 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 point 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 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 *GoldenGate for Windows and UNIX Administrator 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 batch-type processes, such as initial loads or batch change synchronization runs (when SPECIALRUN is used in the Replicat parameter file or with ADD REPLICAT).

Argument	Description

`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 table that follows this one for CSN descriptions. 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 GoldenGate user can use the Logdump utility. For more information about using Logdump, see the *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>`
	
	Limitations:
	◆ Valid only when the trail that Replicat is reading is part of an online change synchronization configuration (with checkpoints). Not valid for batch-type processes, such as initial loads or batch change synchronization runs (when SPECIALRUN is used in the Replicat parameter file or with ADD REPLICAT).
	◆ To support starting at, or after, a CSN, the trail must be of GoldenGate version 10.0 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.

Table 3 GoldenGate CSN values per database[1]

Database	CSN Value

Ingres	`<LSN-high>:<LSN-low>`
	Where:
	◆ <LSN-high> is the newest log file in the range. Up to 4 bytes padded with leading zeroes.
	◆ <LSN-low> is the oldest log file in the range. Up to 4 bytes padded with leading zeroes.
	The valid range of a 4-byte integer is 0 to 4294967295.
	The two components together comprise the Ingres LSN.
	Example:
	`1206396546:43927`

Oracle	`<system change number>`
	Where:
	◆ <system change number> is the Oracle SCN value.
	Example:
	`6488359`

Sybase	`<time_high>.<time_low>.<page>.<row>`
	Where:
	◆ <time_high> and <time_low> represent an instance ID for the log page. It is stored in the header of each database log page. <time_high> is 2-bytes and <time_low> is 4-bytes, each padded with leading zeroes.
	◆ <page> is the database logical page number, padded with zeroes.
	◆ <row> is the row number, padded with zeroes.
	Taken together, these components represent a unique location in the log stream. The valid range of a 2-byte integer for a timestamp-high is 0 - 65535. For a 4-byte integer for a timestamp-low, it is: 0 - 4294967295.
	Example:
	`00001.0000067330.0000013478.00026`

DB2 LUW	`<LSN>`
	Where:
	◆ <LSN> is the variable-length, decimal-based DB2 log sequence number.
	Example:
	`1234567890`

DB2 z/OS	`<RBA>`
	Where:
	◆ <RBA> is the 6-byte relative byte address of the commit record within the transaction log.
	Example:
	`1274565892`

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

Database	CSN Value

SQL Server	Can be any of these, depending on how the database returns it:
	◆ Colon separated hex string (8:8:4) padded with leading zeroes and 0X prefix
	◆ Colon separated decimal string (10:10:5) padded with leading zeroes
	◆ Colon separated hex string with 0X prefix and without leading zeroes
	◆ Colon separated decimal string without leading zeroes
	◆ Decimal string
	Where:
	◆ The first value is the virtual log file number, the second is the segment number within the virtual log, and the third is the entry number.
	Examples:
	`0X00000d7e:0000036b:01bd`
	`0000003454:0000000875:00445`
	`0Xd7e:36b:1bd`
	`3454:875:445`
	`3454000000087500445`

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

SQL/MX	`<sequence number>.<RBA>`
	Where:
	◆ <sequence number> is the 6-digit decimal NonStop TMF audit trail sequence number padded with leading zeroes.
	◆ <RBA> is the 10-digit decimal relative byte address within that file, padded with leading zeroes.
	Together these specify the location in the TMF Master Audit Trail (MAT).
	Example:
	`000042.0000068242`

Teradata	`<sequence ID>`
	Where:
	◆ <sequence ID> is a generic VAM fixed-length printable sequence ID.
	Example:
	`0x08000000000000000D700000021`

[1] All database platforms except Oracle, DB2 LUW, and DB2 z/OS have fixed-length CSNs, which are padded with leading zeroes as required to fill the fixed length. CSNs that contain multiple fields can be padded within each field, such as the Sybase CSN.

For more information about the CSN, see the *GoldenGate for Windows and UNIX Administrator Guide*.

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>]
[, 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. 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 (*).

Argument	Description
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 296.
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.

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.
`!`	(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. 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 GoldenGate trails. A trail is a series of files in which GoldenGate temporarily stores extracted data on disk until it has been applied to the target location.

Command summary

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>]
```

Argument	Description
`<trail name>`	The fully qualified path name of the trail. The trail name can contain only two characters. GoldenGate appends this name with a six-digit sequence number whenever a new file is created. For example, a trail named /ggs/dirdat/tr would have files named /ggs/dirdat/tr000001, /ggs/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 10.

Example

```
ADD EXTTRAIL c:\ggs\dirdat\aa, EXTRACT finance, MEGABYTES 20
```

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>]
```

Argument	Description
`<trail name>`	The fully qualified path name of the trail. The actual trail name can contain only two characters. GoldenGate appends this name with a six-digit sequence number whenever a new file is created. For example, a trail named /ggs/dirdat/tr would have files named /ggs/dirdat/tr000001, /ggs/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 10.

Example `ADD RMTTRAIL c:\ggs\dirdat\aa, EXTRACT finance, MEGABYTES 20`

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.

You can change the size of trail files with the MEGABYTES option. To change the file size, follow this procedure.

1. Issue the following command to view the path name of the trail that you want to alter and the name of the associated Extract group. Use a wildcard to view all trails.

   ```
   INFO EXTTRAIL *
   ```

2. Issue the following command to change the file size.

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

3. Issue the following command to cause Extract to switch to the next file in the trail.

   ```
   SEND EXTRACT <group name>, ROLLOVER
   ```

Syntax

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

Argument	Description
`<trail name>`	The fully qualified path name of the trail, for example c:\ggs\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 10.

Example

```
ALTER EXTTRAIL c:\ggs\dirdat\aa, EXTRACT finance,
MEGABYTES 20
```

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.

You can change the size of trail files with the MEGABYTES option. To change the file size, follow this procedure.

1. Issue the following command to view the path name of the trail that you want to alter and the name of the associated Extract group. Use a wildcard to view all trails.

   ```
   INFO RMTTRAIL *
   ```

2. Issue the following command to change the file size.

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

3. Issue the following command to cause Extract to switch to the next file in the trail.

   ```
   SEND EXTRACT <group name>, ROLLOVER
   ```

Syntax

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

Argument	Description
`<trail name>`	The fully qualified path name of the trail, for example c:\ggs\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 10.

Example

```
ALTER RMTTRAIL c:\ggs\dirdat\aa, EXTRACT finance,
MEGABYTES 20
```

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 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 fully qualified path name of the trail, including the two-character trail prefix.

Example

```
DELETE EXTTRAIL /home/ggs/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 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 fully qualified path name of the trail, including the two-character trail prefix.

Example

```
DELETE RMTTRAIL /home/ggs/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: 10M
```

Syntax

```
INFO EXTTRAIL <trail name>
```

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

Example 1

```
INFO EXTTRAIL c:\ggs\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: 10M
```

Syntax `INFO RMTTRAIL <trail name>`

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

Example 1 `INFO RMTTRAIL c:\ggs\dirdat\aa`

Example 2 `INFO RMTTRAIL *`

Parameter commands

Use parameter commands to view and manage 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.

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. Use the full path name.

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.

Syntax `SET EDITOR <program name>`

Argument	Description
`<program name>`	Any editor that can save text in ASCII format.

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.

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

Argument	Description
`MGR`	Shows the Manager parameter file.
`<group>`	Shows the parameter file for the specified Extract or Replicat group.
`<file name>`	Shows the specified file. Use 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

Use DBLOGIN to establish a database connection through GGSCI in preparation to issue other GoldenGate commands that affect the database.

Syntax

```
DBLOGIN
{SOURCEDB <dsn> |
USERID <user>[, PASSWORD <password>] [SYSDBA] |
SOURCEDB <dsn>, USERID <user>[, PASSWORD <password>][SQLID <sqlid>]
```

Argument	Description
`SOURCEDB <dsn>`	A datasource name. Required for all databases that use ODBC.
`USERID <user>` `[, PASSWORD <password>]`	A database user and that user's password. Use if database credentials are required. PASSWORD is optional. When used, the password is echoed. If PASSWORD is omitted, GoldenGate prompts for a password, and the password is not echoed. If the password is case-sensitive, type it that way.
`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.

Example 1

```
DBLOGIN USERID ggs, PASSWORD ggs123
```

Example 2

```
DBLOGIN SOURCEDB ctdb@host1, USERID ggs, PASSWORD ggs SYSDBA
```

ENCRYPT PASSWORD

Use ENCRYPT PASSWORD to encrypt a database login password that is specified with the USERID or ASMUSERID parameter or with a CREATE USER <user> IDENTIFIED BY <password> command. Without options, ENCRYPT PASSWORD generates a random encryption key, but you can specify a key from a lookup file by using the ENCRYPTKEY option.

ENCRYPT PASSWORD prints the encrypted password to the screen. Copy and paste it into the PASSWORD argument of the USERID, ASMUSERID, or DEFAULTUSERPASSWORD parameters.

Password encryption is not supported for SQL/MX databases.

Syntax

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

Argument	Description
`<password>`	The login password. Do *not* enclose the password within quotes. If the password is case-sensitive, type it that way.

Argument	Description
`ENCRYPTKEY <keyname>`	Optional, specifies the logical name of an encryption key contained in the ENCKEYS lookup file. GoldenGate uses the key name to look up the actual key in the file. To use the <keyname> option, generate the key and store it in an ENCKEYS lookup file on the local system. For more information, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Example `ENCRYPT PASSWORD ny14072 ENCRYPTKEY superkey2`

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
```

Trandata commands

Use trandata commands with log-based extraction to do one of the following:

- Turn on the REPLICATE attribute for c-tree files.
- Control the logging of key values (and other columns if needed) for Oracle tables.
- Control table replication for Sybase tables.
- Add supplemental log data for SQL Server tables.

Command summary

ADD TRANDATA

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

ADD TRANDATA is only required 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 GoldenGate installation guide for your database for any special requirements that apply to making transaction details available to GoldenGate.

c-tree databases

You can use ADD TRANDATA to turn on the REPLICATE attribute of a c-tree file without bouncing the c-tree server. As an alternative, you can turn on REPLICATE by setting one or more REPLICATE parameters in the c-tree server configuration file ctsrvr.cfg before starting the server.

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 provide the extended logging information that 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 system procedure. The default behavior for ADD TRANDATA is to always propagate data if any column changes.

ADD TRANDATA options employ database features to control how the database propagates LOB data for the specified table. By default, Sybase propagates all LOB information, and that information is written to the trail whether or not it has changed.

Oracle databases

The following is how ADD TRANDATA works for Oracle databases.

Oracle 8.x

For Oracle 8.x, ADD TRANDATA installs an update trigger on the specified table. When the trigger fires, it sets each key value to itself, causing the values to be logged. The trigger is unobtrusive. It only executes for update operations performed on the table where it is installed, and it has virtually no effect on CPU load or database performance. To use this trigger for tables in any schema other than the GoldenGate schema, the GoldenGate user must be granted CREATE ANY TRIGGER privileges.

NOTE This trigger will not fire if applications update, insert, or delete LOBs by using the DBMS_LOB package or the Oracle OCI interface.

Oracle 9.x and later

For Oracle 9.x or later, ADD TRANDATA by default enables table-level supplemental logging. The supplemental log group includes one of the following sets of columns, in the listed 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
3. First unique key alphanumerically with no virtual columns, no UDTs, or no function-based columns, but can include nullable columns
4. If none of the preceding key types exist (even though there might be other types of keys defined on the table) 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 GoldenGate configuration.

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.

If supplemental logging cannot be enabled, you can use the USETRIGGER option to install the update trigger. However, supplemental logging provides the best results.

The update trigger is required if the database compatibility is set to 8i.

One of the following additional steps must be taken when using ADD TRANDATA for Oracle 9i or later, depending on the logging method that was specified:

- **Supplemental logging:** Besides table-level logging, minimal supplemental logging must be enabled at the *database level* in order for GoldenGate to process updates to primary keys and chained rows. This is a known Oracle issue and must be done through the database interface, not through GoldenGate. To verify that supplemental logging is enabled at the database level, issue the following statement:

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

  - For Oracle 9i, the output of the query must be YES. When other LOG_DATA options are enabled, LOG_DATA_MIN is automatically enabled, but you should verify it to be certain.

- For Oracle 10g, the output of the query must be YES or IMPLICIT. As of Oracle 10g, LOG_DATA_MIN must be explicitly set, because it is not enabled automatically when other LOG_DATA options are set.

- **Update triggers:** If using ADD TRANDATA with the USETRIGGER option, add the TRANLOGOPTIONS parameter with the FETCHCHAINEDUPDATES option to the Extract parameter file. For more information about this parameter, see page 344.

Additional information for Oracle

The following are additional options for Oracle supplemental logging:

- Use the COLS option to log non-key columns as needed.
- 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 GoldenGate has started extracting data, you will need to stop and start the Extract process that is reading from the affected table before any data is changed.
- When you use an update trigger, any columns that precede the primary key columns are included in the trigger definition automatically, and therefore are logged. This is normal behavior to accommodate row chaining.
- Oracle 8i has a limitation where, if a child table in a foreign-key relationship does not have an index on the foreign-key column, Oracle locks the table when the GoldenGate update trigger updates the key on that table. To prevent the locking, create an index on the foreign-key column.
- When creating an update trigger or supplemental log group with ADD TRANDATA, GoldenGate forms a name as follows:
  - Triggers: appends the table name, an underscore, and object ID to a prefix of GGS_UB_.
  - Log groups: appends the table name, an underscore, and object ID to a prefix of GGS_.

  Because Oracle limits an object name to 30 characters, GoldenGate truncates long table names as needed so the prefix and object ID can be included. For example, the name of a trigger on the ACCT_HQ_A_SALES_COMP_TOTAL_DETAIL table would be GGS_UB_ACCT_HQ_A_SALES_C_33276.

  In GoldenGate versions prior to 8.0.2, the object ID is not used in the trigger and log group names. If upgrading from one of those versions, do either of the following so that the correct name is found when the DELETE TRANDATA or INFO TRANDATA commands are used:

  - The preferred method is to convert existing update triggers and log groups to the new name format. First use DELETE TRANDATA, and then use ADD TRANDATA.
  - If current names must be retained, use the OLDFORMAT option when using INFO TRANDATA and DELETE TRANDATA.

Syntax

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

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 table name but not the owner name.
`COLS (<column list>)`	(Oracle 9.x and later) 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`.

`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, 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, 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`	(Sybase) Controls how the database propagates LOB data for the specified table. ◆ LOBSNEVER prevents LOB data from being propagated. ◆ LOBSALWAYS always propagates LOB data whether or not it has changed in a transaction. This is the default. ◆ LOBSIFCHANGED propagates LOB data only if it was changed during a transaction.
`NOKEY`	(Oracle 9.x and later) 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.

Argument	Description
OLDFORMAT	(Oracle) Uses the naming format for update triggers and log groups that existed prior to GoldenGate version 8.0.2. This format appends GGS_ to log group names and GGS_UB_ to update-trigger names created with ADD TRANDATA, but does not include an object ID. If the name exceeds Oracle's 30-character maximum, GoldenGate truncates the table name. If more than one name gets truncated to the same characters (when tables have similar names), there will be duplicate triggers or log-groups, with no object ID to differentiate them. This can cause INFO TRANDATA and DELETE TRANDATA to affect the wrong object. Do not use this option without guidance from a GoldenGate support representative.
USETRIGGER	(Oracle 9.x and later) Forces an update trigger to be installed instead of enabling supplemental logging. Use this option for an Oracle 9i or later database if you prefer not to enable supplemental logging or if the database compatibility is set to 8i. Note: With supplemental logging disabled, GoldenGate might abend if a table is defined with LOB columns between key columns.

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; a Sybase table to be marked for replication; or the enabling of the REPLICATE attribute of a c-tree file.

```
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 TRANDATA

Use DELETE TRANDATA to do one of the following:

- c-tree: Disable replication for the specified file.
- DB2 LUW and DB2 on z/OS: Alters the table to DATA CAPTURE NONE.
- Oracle: Delete GoldenGate update triggers or 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>
[, OLDFORMAT]
[, USETRIGGER]
```

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.
`OLDFORMAT`	(Oracle) Forces the command to search by table name only, and exclude the object ID. Use this option if the names of log groups or update triggers are in a format supported by GoldenGate versions prior to 8.0.2, or if ADD TRANDATA was issued with the OLDFORMAT option.
`USETRIGGER`	Must be used to delete an update trigger if the ADD TRANDATA <owner.table>, USETRIGGER command was used to add the trigger (Oracle 9i and later).

Example 1

```
DELETE TRANDATA finance.acct
```

Example 2

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

Example 3

```
DELETE TRANDATA finance.acct, USETRIGGER
```

INFO TRANDATA

Use INFO TRANDATA to get the following information:

- c-tree: Determine whether or not replication is enabled.
- DB2 LUW and DB2 on z/OS: Determine whether DATA CAPTURE is enabled or not.
- Oracle: Determine whether GoldenGate update triggers are installed or supplemental logging is enabled.
- Sybase: Determine whether replication is enabled or not, and whether all LOB columns have identical logging settings (as specified with ADD TRANDATA, [LOBSNEVER | LOBSALWAYS | LOBSIFCHANGED].
- SQL Server: Determine whether or not extended logging is enabled.

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

Syntax

```
INFO TRANDATA <owner.table>
[, OLDFORMAT]
[, USETRIGGER]
```

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.
`OLDFORMAT`	(Oracle) Forces the command to search by table name only, and exclude the object ID. Use this option if the names of log groups or update triggers are in a format supported by GoldenGate versions prior to 8.0.2, or if ADD TRANDATA was issued with the OLDFORMAT option.
`USETRIGGER`	Required when GoldenGate update triggers are being used to accurately determine whether or not a trigger exists for the specified table.

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 GoldenGate to track the current position of Replicat in the trail. For more information about using a checkpoint table, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Command summary

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 checkpointing to be part of the Replicat transaction. This system 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 a GoldenGate instance if you specify it with the CHECKPOINTTABLE parameter in a GLOBALS file. More than one instance of GoldenGate (multiple installations) can use the same checkpoint table. GoldenGate keeps track of the checkpoints even when the same Replicat group name exists

in different instances. For more information, see the *GoldenGate for Windows and UNIX Administrator 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. Use this argument to create a non-default table that will be specified with the CHECKPOINTTABLE argument of ADD REPLICAT. It is recommended, but not required, that the table be created in a schema dedicated to 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 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.

   ```
   DBLOGIN
   {SOURCEDB <dsn> |
   USERID <user>[, PASSWORD <password>] |
   SOURCEDB <dsn>, USERID <user>[, PASSWORD <password>]}
   ```

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 2006-01-06 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 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 *GoldenGate for Windows and UNIX Administrator Guide*.

Command summary

ADD TRACETABLE

Use ADD TRACETABLE to create a trace table in the Oracle database. The trace table must reside in the schema of the 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 4 Description of trace table

Name	Null?	Type	Description
GROUP_ID	NOT NULL	VARCHAR2(8)	The name of the Replicat group or batch process.
DB_USER		VARCHAR2(30)	The user ID of the Replicat group or batch process.

Table 4 Description of trace table

Name	Null?	Type	Description
LAST_UPDATE		DATE	The timestamp of the transaction.

Syntax

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

Argument	Description
`<owner>.<table>`	Optional, required 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. The best practice is to 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 343.

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, GoldenGate displays the name and the date and time that it was created; otherwise 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 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.

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 5 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.

Table 5 DUMPDDL tables (continued)

Table	Description
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 GoldenGate DDL schema during the installation of the DDL objects (see the *GoldenGate for Windows and UNIX Administrator 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.

Additional DUMPDDL guidelines

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 non-standard characters (such as =, ?, *) in table or column names. The format also supports multi-byte systems (using UTF-8 as the character set and multi-byte encoding, for example Chinese or German).

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 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 89). To display a list of previous commands, use the HISTORY command (see page 90).

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
```

CREATE SUBDIRS

Use CREATE SUBDIRS when installing GoldenGate. This command creates the default directories within the 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 90).

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 6 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..                  ddddddddd`
	`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 a 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 88) or the FC command (page 89) 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.

Figure 10 Sample INFO ALL output

```
Program     Status      Group       Lag         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 GoldenGate for HP NonStop software.

The following is an example of the output.

```
Processed           Added               Diff      Prog     Group      Node

2006-02-16:14:41:15 2006-02-16:14:41:08 00:00:07  EXTRACT  PQACMD     \QAMD
GROUPCMD REPLICAT RQACMD CLOSEFILES
2006-02-16:14:41:13 2006-02-16:14:41:08 00:00:05  EXTRACT  PQACMD     \QAMD
TACLCMD REPLICAT RQACMD FUP PURGEDATA $QA16.QAETAR
2006-02-16:14:41:13 2006-02-16:14:41:08 00:00:05  EXTRACT  EQACMD     \QAMD
GROUPCMD REPLICAT RQACMD CLOSEFILES
2006-02-16:14:41:12 2006-02-16:14:41:07 00:00:05  EXTRACT  PQACMD     \QAMD
TACLCMD EXTRACT EQACMD FUP PURGEDATA $DATA07.QASRC
2006-02-16:14:41:11 2006-02-16:14:41:08 00:00:03  EXTRACT  EQACMD     \QAMD
TACLCMD REPLICAT RQACMD FUP PURGEDATA $QA16.QAETAR
2006-02-16:14:41:10 2006-02-16:14:41:06 00:00:04  EXTRACT  PQACMD     \QAMD
TESTING FREE-FORM TEXT TO ALL GROUPS
2006-02-16:14:41:09 2006-02-16:14:41:08 00:00:01  GGSCI    N/A        \QAMD
GROUPCMD REPLICAT RQACMD CLOSEFILES
2006-02-16:14:41:08 2006-02-16:14:41:07 00:00:01  EXTRACT  EQACMD     \QAMD
TACLCMD EXTRACT EQACMD FUP PURGEDATA $DATA07.QASRC
2006-02-16:14:41:08 2006-02-16:14:41:08 00:00:00  GGSCI    N/A        \QAMD
```

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 GoldenGate commands. OBEY is useful for executing commands that are frequently used in sequence.

Syntax `OBEY <file name>`

Argument	Description
`<file name>`	The fully qualified name of the file containing the list of commands.

Example `OBEY ./mycommands.txt`

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

```
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
start extract *
info extract *, 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 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.

Figure 12 Sample VERSIONS output

```
Operating System:
HP-UX
Release B.11.00, Version U

Database:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
PL/SQL Release 8.1.7.0.0 - Production
CORE    8.1.7.0.0       Production
TNS for HPUX: Version 8.1.7.0.0 - Development
NLSRTL Version 3.4.1.0.0 - Production
```

Syntax `VERSIONS`

VIEW GGSEVT

Use VIEW GGSEVT to view the GoldenGate error log (ggserr.log file). This file contains information about GoldenGate events, such as process startup, shutdown, and exception conditions. This information is recorded in the system error log, too, but viewing the 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 13 Sample VIEW GGSEVT output

```
2006-01-08 11:20:56  GGS INFO     301  GoldenGate Manager for Oracle,
mgr.prm:  Command received from GUI (START GGSCI ).
2006-01-08 11:20:56  GGS INFO     302  GoldenGate Manager for Oracle,
mgr.prm:  Manager started GGSCI process on port 7840.

2006-01-08 11:21:31  GGS INFO     301  GoldenGate Manager for Oracle,
mgr.prm:  Command received from GUI (START GGSCI ).

2006-01-08 11:21:31  GGS INFO     302  GoldenGate Manager for Oracle,
mgr.prm:  Manager started GGSCI process on port 7841.
2006-01-08 11:24:15  GGS INFO     301  GoldenGate Manager for Oracle,
mgr.prm:  Command received from GUI (START GGSCI ).
2006-01-08 11:24:15  GGS INFO     302  GoldenGate Manager for Oracle,
mgr.prm:  Manager started GGSCI process on port 7842.
```

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 14 Sample report

```
***********************************************************************
** Running with the following parameters **
***********************************************************************
sourceisfile
userid ggs, password ********
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 2006-01-13 11:07:36 (activity since 2006-01-13 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 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>`	A fully qualified file name, 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 c:\ggs\dirrpt\orders.rpt
```

CHAPTER 2

GoldenGate Parameters

This chapter describes all of the GoldenGate parameters that are available to users for configuring, running, and managing GoldenGate processes. The first part of the chapter provides a summary of the parameters for each process. The summary is followed by an alphabetical reference of all parameters.

Using GoldenGate parameters

The following are basic guidelines for using GoldenGate parameter files. For more information, see the the *GoldenGate for Windows and UNIX Administrator Guide*.

Creating a parameter file

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

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

- Notepad on Microsoft Windows systems
- The vi editor on UNIX systems

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

To create a parameter file in GGSCI

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

```
EDIT PARAMS <group name>
```

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

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

```
EDIT PARAMS extora
```

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

```
EDIT PARAMS MGR
```

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

   GoldenGate parameters have the following syntax:

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

   Where:

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

     ```
     USERID ggs, PASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH, &
     ENCRYPTKEY superx128
     RMTHOST sysb, MGRPORT 8040
     RMTTRAIL /home/ggs/dirdat/c1, PURGE
     ```

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

5. Save and close the file.

To create a parameter file with a text editor

To create a parameter file with a text editor, follow the same instructions as in "To create a parameter file in GGSCI" — except skip the first two steps and start at step 3 after you open a text file.

When creating a parameter file outside GGSCI, make certain to:

- Save the parameter file with the name of the Extract or Replicat group that owns it, or save it with the name "mgr" if the Manager process owns it.
- Save the parameter file in the dirprm directory of the GoldenGate installation directory.

Changing a parameter file

You can change parameter settings after processing has started:

- You can change any Manager parameter except the port number while the Manager process is running; to change the port number, you must stop Manager first.
- Extract and Replicat must be stopped before editing the parameter file. If Extract or Replicat is running, changes to parameter settings can have unpredictable and adverse consequences, especially if you are adding tables or changing mapping or filtering rules.

To change Manager parameters

1. If changing the port number, stop the Manager process by using the STOP MANAGER command in GGSCI (or use the Cluster Administrator if using a Windows cluster). If you are not changing the port, do not stop Manager.

   ```
   STOP MANAGER
   ```

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

   ```
   EDIT PARAMS mgr
   ```

3. Make the edits, and then save the file.
4. Do one of the following:

   - Refresh the Manager process if it is still running,
     ```
     REFRESH MANAGER
     ```
   - Start Manager if it is not running.
     ```
     START MANAGER
     ```

To change Extract or Replicat parameters

1. Stop the process.

   ```
   STOP {EXTRACT | REPLICAT} <group name>
   ```

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

   ```
   EDIT PARAMS mgr
   ```

3. Make the edits, and then save the file.
4. Start the process.

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

Parameter Categories

The following are the categories of GoldenGate parameters.

GLOBALS parameters summary

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

Table 7 All GLOBALS parameters

Parameter	Description
MGRSERVNAME	Specifies the name of the Manager process when it is installed as a Windows service.
CHECKPOINTTABLE	Specifies a default checkpoint table.
GGSCHEMA	Specifies the name of the schema that contains the database objects that support DDL synchronization for Oracle.
DDLTABLE	Specifies a non-default name for the DDL history table that supports DDL synchronization for Oracle.
MARKERTABLE	Specifies a non-default name for the DDL marker table that supports DDL synchronization for Oracle.
OUTPUTFILEUMASK	Specifies a umask that can be used by GoldenGate processes to create trail files and discard files.

Manager parameters summary

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

Table 8 Manager parameters: General

Parameter	Description
COMMENT	Allows insertion of comments in a parameter file.

Table 9 Manager parameters: Port management

Parameter	Description
DYNAMICPORTLIST	Specifies the ports that Manager can dynamically allocate.
DYNAMICPORTREASSIGNDELAY	Specifies a time to wait before assigning a port number that was previously assigned.
PORT	Establishes the TCP/IP port number on which Manager listens for requests.

Table 10 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.
USETHREADS | NOUSETHREADS	Controls whether or not Manager uses a thread on a Windows system or creates a child process on a UNIX system to perform background tasks.

Table 11 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 12 Manager parameters: Database login

Parameter	Description
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 13 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.
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.

Extract parameters summary

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 14 Extract parameters: General

Parameter	Description
CHECKPARAMS	Verifies parameter file syntax.
COMMENT	Denotes comments in a parameter file.
ETOLDFORMAT	Generates trails in a format that is compatible with Replicat versions prior to GoldenGate version 6.0.
GETENV	Retrieves variables that were set with the SETENV parameter.
OBEY	Processes parameter statements contained in a different parameter file.
RECOVERYOPTIONS	Controls the recovery mode of the Extract process.
SETENV	Specifies a value for a UNIX environment variable from within the GGSCI interface.
TCPSOURCETIMER | NOTCPSOURCETIMER	Adjusts timestamps of records transferred to other systems when those systems reflect different times.

Table 14 Extract parameters: General (continued)

Parameter	Description
TRACETABLE | NOTRACETABLE	Causes Extract to ignore database changes generated by Replicat during bidirectional synchronization. Supports Oracle.

Table 15 Extract parameters: Processing method

Parameter	Description
BEGIN	Specifies when to begin a processing run.
DSOPTIONS	Specifies Extract processing options when a Vendor Access Module (VAM) is being used.
END	Specifies when to end a processing run.
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.
SPECIALRUN	Specifies a one-time processing task that does not checkpoint from run to run.
VAM	Indicates that a Vendor Access Module (VAM) is being used to provide transactional data to the Extract process.

Table 16 Extract parameters: Database login

Parameter	Description
SOURCEDB	Specifies the data source as part of the login information.
USERID	Specifies database connection information.

Table 17 Extract parameters: Selecting 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.
COMPRESSDELETES | NOCOMPRESSDELETES	Controls whether 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.
DDL	Enables and filters the capture of DDL operations.
DDLSUBST	Enables string substitution in DDL processing.
FETCHOPTIONS	Controls certain aspects of the way that GoldenGate fetches data.
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.
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.
TRIMSPACES | NOTRIMSPACES	Controls whether trailing spaces are trimmed or not when mapping CHAR to VARCHAR columns.

Table 17 Extract parameters: Selecting and mapping data (continued)

Parameter	Description
WILDCARDRESOLVE	Defines rules for processing wildcard table specifications in a TABLE statement.

Table 18 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 19 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 20 Extract parameters: Custom processing

Parameter	Description
CUSEREXIT	Invokes a user exit routine during processing.
INCLUDE	Invokes a macro library.
MACRO	Defines a GoldenGate macro.
MACROCHAR	Defines a macro character other than the default of #.

Table 20 Extract parameters: Custom processing (continued)

Parameter	Description
SQLEXEC	Executes a stored procedure or query during Extract processing.
VARWIDTHNCHAR | NOVARWIDTHNCHAR	Controls whether length information is written to the trail for NCHAR columns.

Table 21 Extract parameters: Reporting

Parameter	Description
CMDTRACE	Displays macro expansion steps in the Extract 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.
TLTRACE	Traces transaction log extraction activity.
TRACE/TRACE2	Shows Extract processing information to assist in revealing processing bottlenecks.
WARNLONGTRANS	Defines a long-running transaction and controls the frequency of checking for and reporting them.

Table 22 Extract parameters: Error handling

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

Table 23 Extract parameters: Tuning

Parameter	Description
ALLOCFILES	Controls the number of incremental memory structures allocated when the value of NUMFILES is reached.

Table 23 Extract parameters: Tuning (continued)

Parameter	Description
CACHEMGR	Controls the virtual memory cache manager.
CHECKPOINTSECS	Controls how often Extract 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 Extract delays before searching for more data to process.
FLUSHSECS | FLUSHCSECS	Determines the amount of time that record data remains buffered before being written to the trail.
FUNCTIONSTACKSIZE	Controls the size of the memory stack that is used for processing GoldenGate functions.
GROUPTRANSOPS	Controls the number of records that are sent to the trail in one batch.
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.
NUMFILES	Controls the initial allocation of memory dedicated to storing information about tables to be processed by GoldenGate.
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 log-processing options.
TRANSMEMORY	Controls the amount of memory and temporary disk space available for caching uncommitted transaction data.

Table 24 Extract parameters: 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.
ROLLOVER	Specifies the way that trail files are aged.

Table 25 Extract parameters: Security

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

Replicat parameters summary

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 26 Replicat parameters: General

Parameter	Description
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.
TRACETABLE | NOTRACETABLE	Specifies a trace table to which Replicat adds a record whenever it updates the target database. Supports Oracle.

Table 27 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 28 Replicat parameters: Database login

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

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.
ASCIITOEBCDIC	Converts incoming ASCII text to EBCDIC for DB2 on z/OS systems running UNIX System Services.
ASSUMETARGETDEFS	Assumes that source and target tables have the same column structure.
COLMATCH	Establishes global column-mapping rules.
DDL	Enables and filters the capture of DDL operations.
DDLSUBST	Enables string substitution in DDL processing.

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

Parameter	Description
GETDELETES | IGNOREDELETES	Controls the replication of delete operations.
GETINSERTS | IGNOREINSERTS	Controls the replication of insert operations.
GETUPDATEAFTERS | IGNOREUPDATEAFTERS	Controls the replication of update after images.
GETUPDATEBEFORES | IGNOREUPDATEBEFORES	Controls the replication of update before images.
GETUPDATES | IGNOREUPDATES	Controls the replication of update operations.
GETTRUNCATES | IGNORETRUNCATES	Includes or excludes the replication of TRUNCATE statements.
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	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.
REPLACEBADCHAR	Replaces invalid character values with another value.
REPLACEBADNUM	Replaces invalid numeric values with another value.
SOURCEDEFS	Specifies a file that contains source data definitions created by the DEFGEN utility.
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.
TRIMSPACES | NOTRIMSPACES	Controls whether trailing spaces are trimmed or not when mapping CHAR to VARCHAR columns.
UPDATEDELETES | NOUPDATEDELETES	Changes deletes to updates.

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

Parameter	Description
USEDATEPREFIX	Prefixes data values for DATE data types with a DATE literal, as required by Teradata databases.
USETIMEPREFIX	Prefixes data values for TIME datatypes with a TIME literal, as required by Teradata databases.
USETIMESTAMPPREFIX	Prefixes data values for TIMESTAMP datatypes with a TIMESTAMP literal, as required by Teradata databases.

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: Custom processing

Parameter	Description
CUSEREXIT	Invokes a user exit routine during processing.
DEFERAPPLYINTERVAL	Specifies a length of time for Replicat to wait before applying replicated operations to the target database.
INCLUDE	References a macro library in a parameter file.
MACRO	Defines a GoldenGate macro.
SQLEXEC	Executes a stored procedure or query during Replicat processing.

Table 32 Replicat parameters: Reporting

Parameter	Description
CMDTRACE	Displays macro expansion steps in the report.
LIST | NOLIST	Displays or suppresses the listing of macros in the report file.
REPORT	Schedules a statistical report at a specified date or time.

Table 32 Replicat parameters: Reporting (continued)

Parameter	Description
REPORTCOUNT	Reports the number of records processed.
SHOWSYNTAX	Causes Replicat to print its SQL statements to the report file.
STATOPTIONS	Specifies information to include in statistical displays.
TRACE/TRACE2	Shows Replicat processing information to assist in revealing bottlenecks.

Table 33 Replicat parameters: Error handling

Parameter	Description
CHECKSEQUENCEVALUE | NOCHECKSEQUENCEVALUE	Controls whether or not Replicat verifies that a target sequence value is higher than the one on the source and corrects any disparity that it finds.
DDLERROR	Controls error handling for DDL replication.
DISCARDFILE	Contains records that could not be processed.
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 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.
SQLDUPERR	Specifies the database error number that indicates a duplicate record. Use with OVERRIDEDUPS.
WARNRATE	Determines how often database errors are reported.

Table 34 Replicat parameters: Tuning

Parameter	Description
ALLOCFILES	Controls the amount of incremental memory that is allocated when the amount of memory specified with NUMFILES is reached.
BATCHSQL	Increases the throughput of Replicat processing by arranging similar SQL statements into arrays and applying them at an accelerated rate.
CHECKPOINTSECS	Controls how often Replicat writes a checkpoint when checkpoints are not being generated as the result of transaction commits.
DBOPTIONS	Specifies database options.
DDLOPTIONS	Specifies DDL processing options.
DYNAMICRESOLUTION | NODYNAMICRESOLUTION	Makes Replicat start faster when there is a large number of tables specified for synchronization.
DYNSQL | NODYNSQL	Causes Replicat to use literal SQL statements rather than a compile-once, execute-many strategy.
EOFDELAY | EOFDELAYCSECS	Determines how many seconds Replicat delays before looking for more data to process.
FUNCTIONSTACKSIZE	Controls the size of the memory stack that is used for processing GoldenGate functions.
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 GoldenGate.

Table 34 Replicat parameters: Tuning (continued)

Parameter	Description
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.
TRANSMEMORY	Controls the amount of memory and temporary disk space available for caching uncommitted transaction data.
WILDCARDRESOLVE	Alters the rules by which wildcard specifications in a MAP statement are resolved.

Table 35 Replicat parameters: Maintenance

Parameter	Description
DISCARDROLLOVER	Specifies when to create new discard files.
PURGEOLDEXTRACTS	Purges obsolete trail files.
REPORTROLLOVER	Specifies when to create new report files.

Table 36 Replicat parameters: Security

Parameter	Description
DECRYPTTRAIL	Decrypts data in a trail or extract file.

DEFGEN parameters summary

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 37 All DEFGEN parameters

Parameters	Description
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 37 All DEFGEN parameters (continued)

Parameters	Description
TABLE for DDLGEN, DEFGEN	Identifies a table for which you want to capture a definition.
USERID	Specifies database connection information.

DDLGEN parameters summary

DDLGEN generates DDL syntax that creates target tables based on the definitions of the source tables. DDLGEN reduces the work necessary to create databases on the target system.

Table 38 All DDLGEN parameters

Parameters	Description
SOURCEDB	Specifies the data source as part of the login information.
TABLE for DDLGEN, DEFGEN	Specifies a table or group of tables for which to create target definitions.
USERID	Specifies database connection information.

Alphabetical Reference

This begins the alphabetical reference for GoldenGate parameters on the Windows and UNIX platforms.

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 248). 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. GoldenGate's memory structures 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 GoldenUser, PASSWORD ****
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");
```

By default, ALLOWDUPTARGETMAP is disabled because wildcards can be used for the same purpose.

Default `NOALLOWDUPTARGETMAP`

Syntax `ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP`

ASCIITOEBCDIC

Valid for Extract and Replicat

Use the ASCIITOEBCDIC parameter to control the conversion of data from ASCII to EBCDIC format. Unicode data is never converted to either format.

This parameter is only required if the Extract is of a GoldenGate version prior to v9.5.0 build 004.

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 GoldenGate not to look up source structures from a source-definitions file.

For structures to be identical, they must contain identical column names (including case, if applicable) and data types, and they must appear in the same order in each table. If source and target tables do not have the same structure, use the SOURCEDEFS parameter instead of ASSUMETARGETDEFS. See "SOURCEDEFS" on page 289.

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. GoldenGate will send the start command to the source system. If AUTORESTART is used locally for a passive Extract group, it is ignored.

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, 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 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 during which retries are counted. The default is 20 minutes. 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. GoldenGate will send the start command to the source system. If AUTOSTART is used locally for a passive Extract group, it is ignored.

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)

Argument	Description
<group name>	A group name or wildcard specification for multiple groups. When wildcarding is used, 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:

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

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 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 245). 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:

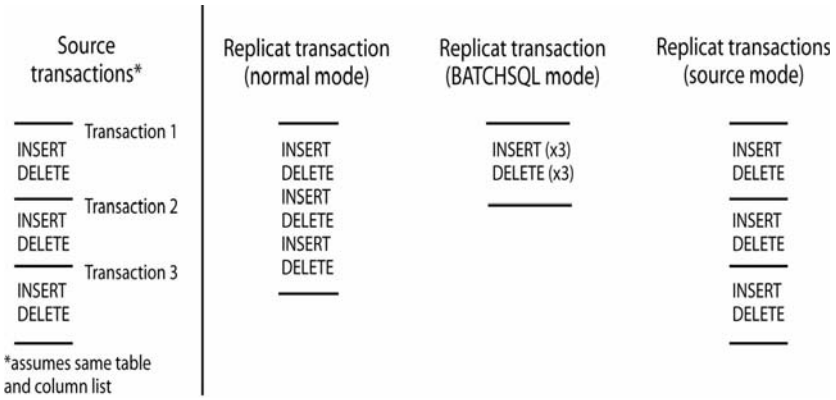
- Statements that contain LOB or LONG data.
- Statements that contain rows longer than 25k in length.
- Statements where the target table has one or more unique keys besides the primary key. Such statements cannot be processed in batches because BATCHSQL does not guarantee the correct ordering for non-primary keys if their values could change.
- 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 191).
- 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 15 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 Replicat's performance 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. The default is 50.
`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.
`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 (1M). ◆ Maximum value is 1000000000 (1B). ◆ Default is 20 megabytes.

Option	Description
OPSPERBATCH <n>	Sets the maximum number of row operations that one batch can contain. After OPSPERBATCH is reached, a target transaction is executed. The 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. The 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 a GoldenGate support analyst.

Example `BATCHSQL BATCHESPERQUEUE 100, OPSPERBATCH 2000`

BEGIN

Valid for Extract and Replicat

Use the BEGIN parameter when SPECIALRUN is specified to create a one-time batch processing run. Processing starts with the first record in the database transaction log or 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.

For instructions on how to configure a batch run, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Default None

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

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

Example `BEGIN 1995-08-12 09:22:55`

BLOBMEMORY

This parameter is an alias for LOBMEMORY (see page 199).

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 GoldenGate processing.

Example `BOOTDELAYMINUTES 5`

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 GoldenGate Technical Support before using NOBINARYCHARS.

For a complete guide to the methods of loading data with GoldenGate, see the *GoldenGate for Windows and UNIX Administrator 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 GoldenGate Technical Support for guidance. The cache manager of 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 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 one containing information that is relative to a transaction that is being processed by GoldenGate. The sizes of the initial and incremental buffers are controlled by the CACHEBUFFERSIZE option of CACHEMGR.

The GoldenGate cache manager takes advantage of the memory management functions of the operating system to ensure that 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 GoldenGate process is controlled by the operating system, not the GoldenGate program.

The cache manager keeps a GoldenGate process working within the soft limit of its global cache size, only allocating virtual memory (not physical memory) on demand. 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 GoldenGate Extract and Replicat process that will be running. To determine the required swap space:

1. Start up one Extract or Replicat.
2. Run GGSCI.
3. View the report file and find the line PROCESS VM AVAIL FROM OS (min).
4. Round up the value to the next full gigabyte if needed. For example, round up 1.76GB to 2 GB.
5. Multiply that value by the number of Extract and Replicat processes that will be running. The result is the maximum amount of swap space that could be required.

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

NOTE The cache manager is also used internally by 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. You might also need to adjust the initial memory allocation with the CACHEBUFFERSIZE option.

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 34.

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)
CACHEBUFFERSIZE: 64K
CACHEBUFFERSIZE (soft max): 4M
CACHESIZE: 1G
CACHEPAGEOUTSIZE (normal): 4M
PROCESS VM AVAIL FROM OS (min): 1.79G
CACHESIZEMAX (strict force to disk): 1.58G
```

Where:

- CACHEBUFFERSIZE shows the default initial buffer allocation unit. This value is dependent on the operating system. On Win32, for example, it is 64 KB. On Unix-based systems, it is typically the hardware page size (4 KB or 8 KB). The buffer size will not be permitted to go below the minimum initial allocation unit that is permitted by the operating system.
- CACHEBUFFERSIZE (soft max) shows the maximum default size of any buffer allocation, as set by the CACHEBUFFERSIZE option.
- CACHESIZE shows the amount 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, GoldenGate maintains data that it swaps to disk in the dirtmp sub-directory of the GoldenGate installation directory. The cache manager assumes that all of the free space on the file system is available. To avoid contention for disk space between GoldenGate and other applications, it is best practice to assign GoldenGate its own disk for its temporary writes. 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 a GoldenGate support case for assistance.

Default None

Syntax

```
CACHEMGR {
   [, CACHESIZE <size>]
   [, CACHEBUFFERSIZE <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. The default is 8GB for 64-bit systems and 2GB for 32-bit systems. 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 either be rejected or 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. 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.
`CACHEBUFFERSIZE <size>`	Controls the initial buffer size allocation that is made by the cache manager. It must be a power of 2 and, even if specified explicitly, might be adjusted internally. Do not set it to be smaller than the operating system's minimal virtual memory allocation unit. On Windows, for example, it is 64 KB. On Unix-based systems, it is typically the hardware page size. The operating system will allocate its minimum size regardless of the CACHEBUFFERSIZE setting. The default for CACHEBUFFERSIZE is the operating system default size. If a transaction's cache virtual memory requirements grow beyond CACHEBUFFERSIZE, 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. If the cache manager statistics show that an application's usage profile is such that most transactions cache approximately the same amount of data, there may be some nominal efficiency gains from increasing CACHEBUFFER size, as long as it does not result in additional file-caching.

Argument	Description
`CACHEPAGEOUTSIZE <size>`	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 4 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. ◆ the statistics on file caching. Before changing this parameter, contact GoldenGate Technical Support.
`CACHEDIRECTORY <path> [<size>]`	Specifies the name of the directory to which GoldenGate writes transaction data to disk temporarily when the soft limit set by CACHESIZE is reached. The default without this parameter is the dirtmp sub-directory of the GoldenGate installation directory. The directory must be on its own file system. If GoldenGate has its own file system, it can be within that file system.

Argument	Description
	◆ <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 GoldenGate's swap space because of resource limitations. You can specify more than one directory, but each one must be on a different file system. 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 CACHEBUFFERSIZE 64KB, CACHESIZE 500MB, CACHEDIRECTORY /ggs/temp, CACHEDIRECTORY /ggs2/temp`

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. Start the process. Without processing data, GoldenGate audits the syntax and connects to the database to verify that listed tables exist. If there is a syntax failure, the process abends with GoldenGate error 190. If the syntax succeeds, the process stops and writes a message to the report file that the parameters processed successfully.
3. Do one of the following:

- If the syntax is correct, edit the file to remove the CHECKPARAMS parameter, and then start the process again to begin processing.
- If the syntax is not correct, edit the parameter file to fix the syntax based on the report's findings, and then 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 directed to do so by GoldenGate Technical Support.

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 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`

CHECKSEQUENCEVALUE | NOCHECKSEQUENCEVALUE

Valid for Replicat

Use the CHECKSEQUENCEVALUE and NOCHECKSEQUENCEVALUE parameters to control whether or not Replicat verifies that a target sequence value is higher than the one on the source. If the target value is either too low or too high, Replicat brings the source and target values back into an appropriate state of disparity.

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 * source_RAC_nodes) <= target_highwater_value

Or...

source_highwater_value + (source_cache_size * source_increment_size * source_RAC_nodes) >= 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, and the source INCREMENTBY value is 1, and there are two source RAC nodes, the target highwater value should be at least 80:

40 + (20*1*2) <80

If the target highwater value is less than 80, 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');
```

Keep the default of CHECKSEQUENCEVALUE unless you know there will not be any gaps in the sequence updates (such as from a trail corruption or process failure) and you want to increase the performance of GoldenGate.

Default CHECKSEQUENCEVALUE

Syntax CHECKSEQUENCEVALUE | NOCHECKSEQUENCEVALUE

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 *GoldenGate for Windows and UNIX Administrator 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>`	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>`	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 to insert comments within a parameter file. Anything on the same line after COMMENT is ignored during processing. Two hyphens (--) also denote a comment.

COMMENT can be used 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 database.`

Example 2 `-- GoldenGate param file for fin database.`

COMPRESSDELETES | NOCOMPRESSDELETES

Valid for Extract

Use the COMPRESSDELETES and NOCOMPRESSDELETES parameters for a log-based Extract group 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.

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.

COMPRESSUPDATES and NOCOMPRESSUPDATES apply globally for all TABLE statements in a parameter file.

This parameter supports the following databases:

- DB2 LUW and DB2 z/OS
- Teradata version 12 or later
- SQL Server

Do not use it for databases other than those listed.

These parameters 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 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 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 GoldenGate solution.

For help with creating and implementing user exits, see the *GoldenGate for Windows and UNIX Administrator 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 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, 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 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 precede the TARGETDB or SOURCEDB parameter statement and/or the USERID statement. Some options apply only to Extract or Replicat.

Default None

Syntax

```
DBOPTIONS
[ALLOWLOBDATATRUNCATE | NOALLOWLOBDATATRUNCATE]
[ALLOWUNUSEDCOLUMN]
[CATALOGCONNECT | NOCATALOGCONNECT]
[CONNECTIONPORT <port>]
[DISABLELOBCACHING]
[EMPTYLOBSTRING '<string>']
[FETCHBATCHSIZE <num_recs>]
[FETCHLOBS | NOFETCHLOBS]
[HOST <host ID>]
[LIMITROWS | NOLIMITROWS]
[LOBWRITESIZE <size>]
[SHOWINFOMESSAGES]
[SHOWWARNINGS]
[SPTHREAD | NOSPTHREAD]
[TDSPACKETSIZE <bytes>]
[TRUSTEDCONNECTION]
[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.
ALLOWLOBDATATRUNCATE | NOALLOWLOBDATATRUNCATE	Valid for Replicat for Sybase. ALLOWLOBDATATRUNCATE prevents Replicat from aborting 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 abort with an error message if the replicated LOB is too large.

Argument	Description
`CATALOGCONNECT | NOCATALOGCONNECT`	Valid for Extract and Replicat for ODBC databases. By default, GoldenGate creates a new connection for catalog queries, but you can use NOCATALOGCONNECT to prevent that. On DB2 for z/OS, NOCATALOGCONNECT prevents 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 GoldenGate may issue commit locks on the system catalog tablespaces until it receives the commit for its open connection. To prevent commit locks, GoldenGate recommends using RRSAF (mvsattachtype=RRSAF), which supports multiple connections.
`CONNECTIONPORT <port>`	Valid for Replicat for multi-daemon MySQL. Specifies the TCP/IP port.
`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, 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 Orace, DB2, Ingres, SQL/MX, SQL Server, and Teradata. Enables array fetches for initial loads to improve performance, rather than one row at a time. The default is 1,000 records per fetch. 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.
`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, GoldenGate captures changes to LOBs from the transaction logs. The default is FETCHLOBS.

Argument	Description

LIMITROWS | NOLIMITROWS	Valid for Replicat for 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. 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`
	The alteration is only applied if the table has no unique key.
	
	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.

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	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 GoldenGate processing.

SHOWWARNINGS	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.

Argument	Description
SPTHREAD | NOSPTHREAD	Valid for Extract and Replicat. Creates a separate database connection thread for stored procedures. The default is NOSPTHREAD.
TDSPACKETSIZE <bytes>	Valid for Replicat. Sets the TDS packet size for replication to a Sybase target. Must be set to a multiple of 512. Valid values are 512 (the default) through 8192.
TRUSTEDCONNECTION	Valid for Extract and Replicat for SQL Server. Causes GoldenGate to connect by using trusted connection = yes. Contact GoldenGate Technical Support before using this option.
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 TDSPACKETSIZE 2048`

Example 3 `DBOPTIONS FETCHBATCHSIZE 2000`

Example 4 `DBOOPTION XMLBUFSIZE 2097152`

DDL

Valid for Extract and Replicat

Use the DDL parameter to enable DDL support and filter DDL operations. When used without options, the DDL parameter 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 GoldenGate trail and applies them to the target. This is the same as the default behavior without this parameter.

When used with options, DDL 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.

- *When combined, multiple option specifications are linked logically as AND statements.*
- All criteria specified with multiple options must be satisfied for a DDL statement to be replicated.
- When using complex DDL filtering criteria, it is recommended that you test your configuration in a test environment before using it in production.

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

Do not use DDL for an Extract data pump or for a VAM-sort Extract. These process types do not permit mapping or conversion of DDL and will propogate DDL records automatically in PASSTHRU mode (see page 250). 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 GoldenGate DDL support, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Syntax

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

Table 39 DDL inclusion and exclusion options

Option	Description

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

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

Table 39 DDL inclusion and exclusion options

Option	Description

OBJNAME "<name>"	Use OBJNAME to apply INCLUDE or EXCLUDE to the name of an object, for example a table name. This option takes a double-quoted string as input, and wildcards can be used. If you do not qualify the object name, the owner is assumed to be the GoldenGate user that is specified with the USERID parameter. For example:
	Owner is GoldenGate:
	`DDL INCLUDE OBJNAME "tab_customers"`
	Owner is accounts:
	`DDL INCLUDE OBJNAME "accounts.*"`
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`
	
	In this example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.
	
	For DDL that creates triggers and indexes, the value for OBJNAME must be the name of the base object, not the name of the trigger or index. For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig."
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct."
	`ALTER TABLE hr.accounts RENAME TO acct;`

INSTR '<string>'	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax itself, but not within comments. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive.

Table 39 DDL inclusion and exclusion options

Option	Description

`INSTRCOMMENTS '<comment_string>'`	Use INSTRCOMMENTS to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within a comment, but not within the DDL command itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.
	For example, the following excludes DDL statements that include "source" in the comments.
	`DDL INCLUDE ALL EXCLUDE INSTRCOMMENTS 'SOURCE ONLY'`
	In this example, DDL statements such as the following are not replicated.
	`CREATE USER john IDENTIFIED BY john /*source only*/;`
	Enclose the string within single quotes. The string search is not case sensitive. You can combine INSTR and INSTRCOMMENTS to filter on a string in the command syntax and in the comments of the same DDL statement.

`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`

`INSTRCOMMENTSWORDS '<word list>'`	Works the same way as INSTRWORDS, but only applies to comments within a DDL statement, not the DDL syntax itself. By using INSTRCOMMENTS, you can use comments as a filtering agent. You can combine INSTRWORDS and INSTRCOMMENTSWORDS to filter on a string in the command syntax and in the comments of the same DDL statement.

Example 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.”

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.

Argument	Description
`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>}
[RETRYOP MAXRETRIES <n> [RETRYDELAY <delay>]]
{INCLUDE <inclusion clause> | EXCLUDE <exclusion clause>}
[IGNOREMISSINGTABLES | ABENDONMISSINGTABLES]
```

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

Table 40 DDL inclusion and exclusion options

Option	Description

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

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

Table 40 DDL inclusion and exclusion options

Option	Description

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the name of an object, for example a table name. This option takes a double-quoted string as input, and wildcards can be used. If you do not qualify the object name, the owner is assumed to be the GoldenGate user that is specified with the USERID parameter. For example:
	Owner is GoldenGate:
	`DDL INCLUDE OBJNAME "tab_customers"`
	Owner is accounts:
	`DDL INCLUDE OBJNAME "accounts.*"`
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`
	
	In this example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.
	
	For DDL that creates triggers and indexes, the value for OBJNAME must be the name of the base object, not the name of the trigger or index. For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig."
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct."
	`ALTER TABLE hr.accounts RENAME TO acct;`

`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax itself, but not within comments. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive.

Table 40 DDL inclusion and exclusion options

Option	Description

`INSTRCOMMENTS '<comment_string>'`	Use INSTRCOMMENTS to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within a comment, but not within the DDL command itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.
	For example, the following excludes DDL statements that include "source" in the comments.
	`DDL INCLUDE ALL EXCLUDE INSTRCOMMENTS 'SOURCE ONLY'`
	In this example, DDL statements such as the following are not replicated.
	`CREATE USER john IDENTIFIED BY john /*source only*/;`
	Enclose the string within single quotes. The string search is not case sensitive. You can combine INSTR and INSTRCOMMENTS to filter on a string in the command syntax and in the comments of the same DDL statement.

`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`

`INSTRCOMMENTSWORDS '<word list>'`	Works the same way as INSTRWORDS, but only applies to comments within a DDL statement, not the DDL syntax itself. By using INSTRCOMMENTS, you can use comments as a filtering agent. You can combine INSTRWORDS and INSTRCOMMENTSWORDS to filter on a string in the command syntax and in the comments of the same DDL statement.

Example 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 GoldenGate DDL support, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Default See the argument descriptions

Syntax

```
DDLOPTIONS
[, ADDTRANDATA]
[, DEFAULTUSERPASSWORDPASSWORD <password>
   [ENCRYPTKEY DEFAULT | ENCRYPTKEY <keyname>]]
[, GETAPPLOPS | IGNOREAPPLOPS]
[, GETREPLICATES | IGNOREREPLICATES]
[, MAPDERIVED | NOMAPDERIVED]
[, NOCROSSRENAME]
[, REMOVECOMMENTS {BEFORE | AFTER}]
[, REPLICATEPASSWORD | NOREPLICATEPASSWORD]
[, REPORT | NOREPORT]
```

Argument	Description
ADDTRANDATA	Valid for Extract. (Oracle only) Use ADDTRANDATA to: ◆ enable Oracle's 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. 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. 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.

Argument	Description

	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.
	`EDIT PARAMS <group>`
	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>`
	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.

`DEFAULTUSERPASSWORD <password> [ENCRYPTKEY DEFAULT | ENCRYPTKEY <keyname>]`	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. ◆ DEFAULTUSERPASSWORD <password> specifies a clear-text password. If the password is case-sensitive, type it that way. ◆ DEFAULTUSERPASSWORD <encrypted password> ENCRYPTKEY DEFAULT specifies a password that was encrypted by using a random key generated by GoldenGate. On the target, the password is decrypted automatically without requiring an ENCKEYS file. ◆ DDLOPTIONS DEFAULTUSERPASSWORD <encrypted password> ENCRYPTKEY <keyname> specifies a password that was encrypted with a user-defined method and specifies a lookup key in an ENCKEYS file on the target system.

Argument	Description
	When using DEFAULTUSERPASSWORD, use the NOREPLICATEPASSWORD option of DDLOPTIONS for Extract. For more information about GoldenGate encryption options, see the *GoldenGate for Windows and UNIX Administrator Guide*.
`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 IGNOREREPLICATES. ◆ 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.
`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 *GoldenGate for Windows and UNIX Administrator Guide*.

Argument	Description
NOCROSSRENAME	Valid for Extract (Oracle and Teradata). enforce the rule that objects which are excluded from the GoldenGate configuration cannot be renamed to names that are in the configuration. If an object is renamed to a name that is in the configuration, Extract generates a warning so that the appropriate action can be taken. An example of how this notification is useful is to prevent errors if a renamed object has a structure that is not supported by GoldenGate. NOCROSSRENAME improves performance on Oracle RAC by eliminating processing that otherwise is required to keep track of excluded tables in case they get renamed to an included name. NOCROSSRENAME applies globally to all TABLE and TABLEEXCLUDE statements in the parameter file, as well as to all other objects that are excluded but not specified by TABLE or TABLEEXCLUDE. NOCROSSRENAME functionality is similar to that of the NORENAME option of the TABLEEXCLUDE parameter (see page 337), except that NORENAME allows more selectivity.
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 155). ◆ 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), 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.

Argument	Description
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.

Example

```
DDLOPTIONS DEFAULTUSERPASSWORD ocean

DDLOPTIONS DEFAULTUSERPASSWORD AACAAAAAAAAAAADADESGTFTATAOEEIKB ENCRYPTKEY
superkey1

DDLOPTIONS DEFAULTUSERPASSWORD AACAAAAAAAAAAADALHSFYDIEWDEIEIHC 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 GoldenGate DDL configuration, but the sales-owned objects are not.
- You can use multiple DDSUBST 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>`	Use one or more INCLUDE and EXCLUDE statements to filter the DDL operations for which the string substitution rules are applied. See the following table.

Table 41 DDL inclusion and exclusion options

Option	Description

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

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

Table 41 DDL inclusion and exclusion options

Option	Description

`OBJNAME "<name>"`	Use OBJNAME to apply INCLUDE or EXCLUDE to the name of an object, for example a table name. This option takes a double-quoted string as input, and wildcards can be used. If you do not qualify the object name, the owner is assumed to be the GoldenGate user that is specified with the USERID parameter. For example:
	Owner is GoldenGate:
	`DDL INCLUDE OBJNAME "tab_customers"`
	Owner is accounts:
	`DDL INCLUDE OBJNAME "accounts.*"`
	When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME "fin2.*".
	`MAP fin.exp_*, TARGET fin2.*;`
	
	In this example, a CREATE TABLE statement executes like this on the source:
	`CREATE TABLE fin.exp_phone;`
	And like this on the target:
	`CREATE TABLE fin2.exp_phone;`
	If a target owner is not specified in the MAP statement, Replicat maps it to the database user that is specified with the USERID parameter.
	
	For DDL that creates triggers and indexes, the value for OBJNAME must be the name of the base object, not the name of the trigger or index. For example, to include the following DDL statement, the correct value is "hr.accounts," not "hr.insert_trig."
	`CREATE TRIGGER hr.insert_trig ON hr.accounts;`
	For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is "hr.acct."
	`ALTER TABLE hr.accounts RENAME TO acct;`

`INSTR '<string>'`	Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax itself, but not within comments. For example, the following excludes DDL that creates an index. `DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'` Enclose the string within single quotes. The string search is not case sensitive.

Table 41 DDL inclusion and exclusion options

Option	Description

`INSTRCOMMENTS '<comment_string>'`	Use INSTRCOMMENTS to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within a comment, but not within the DDL command itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.
	For example, the following excludes DDL statements that include "source" in the comments.
	`DDL INCLUDE ALL EXCLUDE INSTRCOMMENTS 'SOURCE ONLY'`
	In this example, DDL statements such as the following are not replicated.
	`CREATE USER john IDENTIFIED BY john /*source only*/;`
	Enclose the string within single quotes. The string search is not case sensitive. You can combine INSTR and INSTRCOMMENTS to filter on a string in the command syntax and in the comments of the same DDL statement.

`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`

`INSTRCOMMENTSWORDS '<word list>'`	Works the same way as INSTRWORDS, but only applies to comments within a DDL statement, not the DDL syntax itself. By using INSTRCOMMENTS, you can use comments as a filtering agent. You can combine INSTRWORDS and INSTRCOMMENTSWORDS to filter on a string in the command syntax and in the comments of the same DDL statement.

Example 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 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 GoldenGate installation directory.

This parameter is valid only for Oracle. For more information about the DDL history table and params.sql, see *GoldenGate for Windows and UNIX Administrator 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. Use DECRYPTTRAIL only when the ENCRYPTTRAIL parameter is used for an upstream Extract process to encrypt the trail.

Default None

Syntax `DECRYPTTRAIL`

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.

Default None

Syntax `DEFSFILE <filename> [APPEND | PURGE]`

Argument	Description
`<filename>`	The fully qualified file name. The file is created when you run DEFGEN.

Argument	Description
APPEND	Directs DEFGEN to write new content (from the current run) at the end of any existing content, if the specified file already exists.
PURGE	Directs DEFGEN to purge the specified file before writing new content from the current run. This is the default.

Example
```
DEFSFILE /home/ggs/dirdef/orcldef
```

Syntax

DISCARDFILE

Valid for Extract and Replicat

Use the DISCARDFILE parameter to generate a discard file to which 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. GoldenGate creates the specified discard file in the dirrpt sub-directory of the 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 fully qualified name of the discard file.

To prevent having 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 GoldenGate will not write to an existing discard file.

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, GoldenGate does not generate a discard file.

Syntax
```
DISCARDFILE <file name>
[, APPEND | PURGE]
[, MAXBYTES <n> | MEGABYTES <n>]
```

Argument	Description
<file name>	The fully qualified name of the discard file. If the file is in the GoldenGate directory, the full name is not needed. GoldenGate qualifies the name with the GoldenGate installation directory. GoldenGate recommends using the same name as that of the process group.
APPEND	Adds new content to existing content if the file already exists.
PURGE	Purges the file before writing new content.

Argument	Description
`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.txt, 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.

Argument	Description

`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 Vendor Access Module (VAM). For more information about VAM extraction, see the *GoldenGate for Windows and UNIX Administrator 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 VAM will send transaction data changes to an Extract process that will save it to a regular 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 VAM and writes to the VAM trail. Specify the VAM trail with the ADD EXTTRAIL command in GGSCI.
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 GoldenGate.
RESTARTAPPEND	(Maximum performance mode) Directs Extract to append data to the end of the 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 the ports that Manager can dynamically allocate for dynamic communications between the source and target systems, such as to the Collector and Replicat processes and to GGSCI sessions. You can specify an individual port, a range of ports, or both.

To control how long Manager waits before attempting to reassign a port, use the DYNAMICPORTREASSIGNDELAY parameter.

Default None

Syntax `DYNAMICPORTLIST {<port> | <port>-<port>} [ , ...]`

Argument	Description

`<port>`	A port number that can be allocated. The maximum number of port entries is 256.
	◆ 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`

DYNAMICPORTREASSIGNDELAY

Valid for Manager

Use the DYNAMICPORTREASSIGNDELAY parameter to specify the number of seconds that Manager waits before it attempts to reassign a port that was previously assigned. Use this parameter with the DYNAMICPORTLIST parameter.

Default 3 seconds

Syntax `DYNAMICPORTREASSIGNDELAY <seconds>`

Argument	Description
`<seconds>`	The number of seconds to delay.

Example `DYNAMICPORTREASSIGNDELAY 5`

DYNAMICRESOLUTION | NODYNAMICRESOLUTION

Valid for Extract and Replicat

Use the DYNAMICRESOLUTION and NODYNAMICRESOLUTION parameters to control how table names are resolved.

Use DYNAMICRESOLUTION to make processing start sooner when there is a large number of tables specified in TABLE or MAP statements. By default, whenever a process starts, GoldenGate queries the database for the attributes of the tables and then builds an object record for them. The record is maintained in memory and on disk, and the process of building it can be time-consuming if the database is large.

DYNAMICRESOLUTION causes the object record to be built one table at a time, instead of all at once. A table's attributes are added to the record the first time its object ID enters the transaction log, which occurs with the first extracted transaction on that table. Record-building for other tables is deferred until activity occurs. DYNAMICRESOLUTION is the same as WILDCARDRESOLVE DYNAMIC.

NODYNAMICRESOLUTION causes the object record to be built at startup. This option is not supported for Teradata. NODYNAMICRESOLUTION is the same as WILDCARDRESOLVE IMMEDIATE.

For more information about WILDCARDRESOLVE, see page 369.

Default DYNAMICRESOLUTION

Syntax DYNAMICRESOLUTION

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 without NODYNSQL:
  ```
  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 GoldenGate Technical Support before using either of these parameters.

Default DYNSQL

Syntax DYNSQL | NODYNSQL

ENCRYPTTRAIL | NOENCRYPTTRAIL

Valid for Extract

Use the ENCRYPTTRAIL and NOENCRYPTTRAIL parameters to control whether or not GoldenGate encrypts data written to trail files. ENCRYPTTRAIL encrypts all records both across any data links and within the files themselves. NOENCRYPTTRAIL prevents encryption.

You can use encryption for trails or extract files specified with the following parameters in the Extract parameter file.

RMTTRAIL

EXTTRAIL

RMTFILE

EXTFILE

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 160) in the parameter files of all Replicat processes that read the encrypted files. You also can use DECRYPTTRAIL for an Extract data pump to decrypt the data for column mapping, filtering, transformation, and so forth. You can then leave it decrypted for downstream trails or files, or you can use ENCRYPTTRAIL to encrypt the data again before it is written to those files.

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 internal GoldenGate format.

ENCRYPTTRAIL encrypts only the data blocks. User tokens are not encrypted.

Default NOENCRYPTTRAIL (no encryption)

Syntax `ENCRYPTTRAIL | NOENCRYPTTRAIL`

Example In the following example, data for the emp table is encrypted, whereas data for the stores table is not.

```
ENCRYPTTRAIL
RMTTRAIL /home/ggsora/dirdat/em
TABLE hr.emp;
NOENCRYPTTRAIL
RMTTRAIL /home/ggsora/dirdat/st
TABLE ops.stores;
```

END

Valid for Extract and Replicat

Use the END parameter for a batch run to terminate a process when it encounters the first record in the data source whose timestamp is the specified point in time. Records dating up to that timestamp are processed in the batch.

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 in a batch job or as part of an online change synchronization configuration to post data as a point-in-time snapshot, rather than continuously updating the target tables.

For instructions on how to configure a batch run, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Default Continuous processing

Syntax
```
END {<date> [<time>] | RUNTIME}
```

Argument	Description
<date> [<time>]	Causes Extract or 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 Extract or 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 2009-01-12 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.
`<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 GoldenGate version 6.0.

From 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.

Default None

Syntax `ETOLDFORMAT`

EXTFILE

Valid for Extract and Replicat

Use the EXTFILE parameter to specify an extract file on the local system. The implementation of this parameter varies slightly, depending on the process.

- For Extract, use this parameter to specify a local file that will be read by a data pump Extract group or a Replicat group on the local system.
- For Replicat, use this parameter to specify a local extract file when using SPECIALRUN to generate a batch run.

On Solaris systems, the size of an extract file cannot exceed 2GB if it will be processed by Extract or Replicat. The size can be larger if the file will be processed by another application, such as a native load utility as part of an initial load.

EXTFILE must precede all associated TABLE or MAP statements. Multiple EXTFILE statements can be used to define different files.

About file versioning

Because all of the GoldenGate processes are decoupled and thus can be of different GoldenGate versions, each trail file or extract file has a version that is stored in the file header. By default, the version of a trail is the current version of the process that created the file. To set the version of a trail, use the FORMAT option of the EXTTRAIL, EXTFILE, RMTTRAIL, or RMTFILE parameter.

To ensure forward and backward compatibility of files among different GoldenGate process versions, the file header fields are written in a standardized token format. New tokens that are created by new versions of a process can be ignored by older versions, so that backward compatibility is maintained. Likewise, newer GoldenGate versions support older tokens. Additionally, if a token is deprecated by a new process version, a default value is assigned to the token so that older versions can still function properly. The token that specifies the file version is COMPATIBILITY and can be viewed in the Logdump utility and also by retrieving it with the GGFILEHEADER option of the @GETENV function.

A trail or extract file must have a version that is equal to, or lower than, that of the process that *reads* it. Otherwise the process will abend. Additionally, GoldenGate forces the output trail or file of a data pump to be the same version as that of its input trail or file. Upon restart, Extract rolls a trail to a new file to ensure that each file is of only one version (unless the file is empty).

Parameter dependencies

There is a dependency between the RECOVERYOPTIONS parameter and the FORMAT option of EXTTRAIL, RMTTRAIL, EXTFILE, and RMTFILE. 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 None

Syntax

```
EXTFILE <file name>
[, MAXFILES <number>]
[, MEGABYTES <megabytes>]
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<file name>`	Valid for Extract and Replicat. Specifies the fully qualified name of the extract file.
`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).

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 format depends on the version of the GoldenGate process; older GoldenGate versions contain less, or different, metadata than newer ones. The metadata tells the process whether the data records are of a version that it supports. ◆ FORMAT is a required keyword. ◆ RELEASE specifies a GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current GoldenGate version number. (If you use a 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. If append mode is not enabled for Extract, the file version number automatically defaults to 9 (GoldenGate version 9.x). The Extract mode is controlled by the RECOVERYOPTIONS parameter.

Example 1 `EXTFILE /ggs/dirdat/datafile`

Example 2 `EXTFILE /ggs/dirdat/extdat, MAXFILES 3, MEGABYTES 5`

Example 3 `EXTFILE /ggs/dirdat/extdat, FORMAT RELEASE 10.0`

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 *GoldenGate for Windows and UNIX Administrator 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. The implementation of this parameter varies slightly, depending on the process.

- For Extract, use this parameter to specify a local trail that was created with the ADD EXTTRAIL command. Typically, the trail is read by a data pump Extract group or by a Replicat group on the local system.
- For Replicat, this parameter should be used only when the SPECIALRUN parameter is specified to generate a batch run. For continuous change-synchronization, EXTTRAIL is specified for Replicat with the ADD REPLICAT command in GGSCI.

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 20 for more information.

About file versioning

Because all of the GoldenGate processes are decoupled and thus can be of different GoldenGate versions, each trail file or extract file has a version that is stored in the file header. By default, the version of a trail is the current version of the process that created the file. To set the version of a trail, use the FORMAT option of the EXTTRAIL, EXTFILE, RMTTRAIL, or RMTFILE parameter.

To ensure forward and backward compatibility of files among different GoldenGate process versions, the file header fields are written in a standardized token format. New tokens that are created by new versions of a process can be ignored by older versions, so that backward compatibility is maintained. Likewise, newer GoldenGate versions support older tokens. Additionally, if a token is deprecated by a new process version, a default value is assigned to the token so that older versions can still function properly. The token that specifies the file version is COMPATIBILITY and can be viewed in the Logdump utility and also by retrieving it with the GGFILEHEADER option of the @GETENV function.

A trail or extract file must have a version that is equal to, or lower than, that of the process that *reads* it. Otherwise the process will abend. Additionally, GoldenGate forces the output trail or file of a data pump to be the same version as that of its input trail or file. Upon restart, Extract rolls a trail to a new file to ensure that each file is of only one version (unless the file is empty).

Parameter dependencies

There is a dependency between the RECOVERYOPTIONS parameter and the FORMAT option of EXTTRAIL, RMTTRAIL, EXTFILE, and RMTFILE. 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 None

Syntax
```
EXTTRAIL <file name>
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<file name>`	The 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.
`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 format depends on the version of the GoldenGate process; older GoldenGate versions contain less, or different, metadata than newer ones. The metadata tells the process whether the data records are of a version that it supports. ◆ FORMAT is a required keyword. ◆ RELEASE specifies a GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current GoldenGate version number. (If you use a 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. If append mode is not enabled for Extract, the file version number automatically defaults to 9 (GoldenGate version 9.x). The Extract mode is controlled by the RECOVERYOPTIONS parameter.

Example 1 `EXTTRAIL /ggs/dirdat/ny`

Example 2 `EXTTRAIL /ggs/dirdat/ex, FORMAT RELEASE 10.0`

FETCHOPTIONS

Valid for Extract

Use the FETCHOPTIONS parameter to control certain aspects of the way that GoldenGate fetches data in the following circumstances:

- When the transaction record does not contain enough information for Extract to reconstruct an update operation.
- When 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
[, MISSINGROW <action>]
[, NOFETCH]
[, USEKEY | NOUSEKEY]
[, USELATESTVERSION | NOUSELATESTVERSION]
[, USESNAPSHOT | NOUSESNAPSHOT]
[, USEROWID | NOUSEROWID]
```

Argument	Description
`MISSINGROW <action>`	Provides a response when 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.
`USEKEY | NOUSEKEY`	Valid for Oracle. Determines whether or not 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 9i or later. 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.

Argument	Description
USESNAPSHOT | NOUSESNAPSHOT	Valid for Oracle 9i or later. The default, USESNAPSHOT, causes Extract to use the Flashback Query mechanism to fetch data needed to reconstruct operations containing LOB data, user defined data types, nested tables, and XMLType records from the undo tablespace. NOUSESNAPSHOT causes Extract to fetch the needed data from the source table. For more information about how GoldenGate fetches data from Oracle, see the *GoldenGate for Windows and UNIX Administrator Guide*.
USEROWID | NOUSEROWID	Valid for Oracle. Determines whether or not 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 *GoldenGate for Windows and UNIX Administrator 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 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 271).

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 of universal data 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.

NOTE Do not use FORMATASCII if the data will be processed by the Replicat process. Replicat expects the default of universal data format. Do not use FORMATASCII if FORMATSQL or FORMATXML is being used.

Default output

Without 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.
- 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.

You can customize the output format with optional arguments.

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).

Option	Description
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.
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

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,1997-02-17:14:09:46.421335,8,1873474,
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|1997-02-17:14:09:46.421335|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,1997-02-17:14:09:46.421335,Eric,San Fran,550,
V,CUSTOMER,1997-02-17:14:09:46.421335,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 of universal data format. FORMATSQL generates SQL statements (INSERT, UPDATE, DELETE) that can be applied to both SQL and Enscribe tables by utilities other than GoldenGate Replicat.

NOTE Do not use FORMATSQL if the data will be processed by the Replicat process. Replicat expects the default of universal data format. Do not use FORMATSQL if FORMATASCII or FORMATXML is being used.

A FORMATSQL statement affects all extract files or trails defined after it.

Default output

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. You can customize the output format with optional arguments.

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('1996-05-01','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 of universal data format. A FORMATXML statement affects all extract files or trails that are defined after it.

When using FORMATXML, use the NOBINARYCHARS parameter. NOBINARYCHARS is an undocumented parameter that causes GoldenGate to treat binary data as a null-terminated string. Contact GoldenGate Technical Support before using NOBINARYCHARS.

NOTE Do not use FORMATXML if the data will be processed by the Replicat process. Replicat expects the default of universal data format. Do not use FORMATXML if FORMATASCII or FORMATSQL is being used.

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 GoldenGate functions. The memory stack holds arguments supplied to and from a GoldenGate function. You should not need to use this parameter unless GoldenGate returns a message indicating that the size of the stack should be increased. This could happen when you are using a very large number of functions or arguments.

The default without FUNCTIONSTACKSIZE is 200 arguments, which optimizes GoldenGate's performance and its usage of system memory. Increasing this parameter can adversely affect GoldenGate's performance and use of system memory.

FUNCTIONSTACKSIZE must appear in the parameter file before any parameter clauses 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 a 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 GoldenGate to output the load files and performing the initial load, see the *GoldenGate for Windows and UNIX Administrator 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. 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.

Figure 16, Figure 17, and Figure 18 illustrate the GoldenGate templates, which contain placeholders for the target tables, the data file(s) produced by FORMATASCII, and other run parameters. GoldenGate replaces the placeholders with values based on parameters specified in the Replicat parameter file.

Figure 16 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 17 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 18 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. GoldenGate defaults to IMMEDIATE when GENLOADFILES is specified.

Default None

```
GENLOADFILES [<template file>]
```

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 GoldenGate home directory.

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 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 187.

For more information about configuring bidirectional replication, see the *GoldenGate for Windows and UNIX Administrator Guide*.

GETAPPLOPS and IGNOREAPPLOPS can be used globally or in association with specific TABLE statements. If multiple statements are used in a global manner in the same parameter file, only the last entry will be effective at run time.

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 150.

Default GETAPPLOPS

Syntax GETAPPLOPS | IGNOREAPPLOPS

GETDELETES | IGNOREDELETES

Valid for Extract and Replicat

Use the GETDELETES and IGNOREDELETES parameters to control whether or not 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 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.

GETREPLICATES and IGNOREREPLICATES can be used globally or in association with specific TABLE statements. If multiple statements are used in a global manner in the same parameter file, only the last entry will be effective at run time.

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 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.

Depending on which database you are using, you may or may not need to provide explicit instructions to Extract so that it can identify the DML transactions that are applied by Replicat:

DB2 on z/OS and LUW

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

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

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

Ingres

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

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

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

NonStop SQL/MX

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

```
TRANLOGOPTIONS FILTERTABLE <table_name>
```

Replicat writes a checkpoint to this table at the end of each of its transactions. The local Extract ignores transactions that contain an operation on the checkpoint table, thus identifying the Replicat transaction by default.

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

SQL Server

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

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

This parameter statement is only required if the Replicat transaction name is set to something other than the default of ggs_repl.

Sybase

Do any of the following:

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

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

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

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

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

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

Teradata

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

c-tree

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

Oracle

(**Oracle 10g and later**) Do either of the following to specify the Replicat database user. All transactions generated by this user will be excluded from being captured. This information is available to Extract in the transaction record.

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

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

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

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

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

Additional information about these parameters

- For more information about creating a checkpoint table, see the ADD CHECKPOINTTABLE command on page 81.
- For more information about creating and using a trace table, see the TRACETABLE parameter on page 343 and ADD TRACETABLE on page 84.
- For more information about TRANLOGOPTIONS, see page 344.
- For more information about SQLEXEC, see page 293.
- For more information about using a cascading or bidirectional configuration, see the *GoldenGate for Windows and UNIX Administrator Guide*.
- For more information about GETAPPLOPS and IGNOREAPPLOPS, see page 185.
- To use GETAPPLOPS or IGNOREAPPLOPS functionality for DDL operations, see the DDLOPTIONS parameter on page 150.

Default IGNOREREPLICATES

Syntax GETREPLICATES | IGNOREREPLICATES

GETTRUNCATES | IGNORETRUNCATES

Valid for Extract and Replicat

Use the GETTRUNCATES and IGNORETRUNCATES parameters to control whether or not

GoldenGate processes table truncate operations. By default, truncate operations are not captured from the source or replicated to the target.

Truncates are supported as follows:

- Extraction of truncate operations is supported for Oracle, SQL Server 2000, SQL Server 2005 if upgraded to CU6, and Sybase. For Sybase, table names must be unique across all schemas within a given database.
- Delivery of truncate operations is supported for Oracle, SQL Server 2000, SQL Server 2005 if upgraded to CU6, Sybase, DB2 LUW, DB2 z/OS, MySQL, Ingres, and other ODBC targets that support the TRUNCATE command.

Note that:

- DB2 LUW does not support a TRUNCATE command, so Replicat replicates a truncate operation by performing an IMPORT REPLACE from a NULL (blank) file.
- As of Oracle 10*g*r2, the database does not log truncates against an empty table, so those operations are not captured by GoldenGate. GoldenGate DDL support can be used for this purpose.
- As of Oracle version 10*g*r2, the database does not log truncates for empty partitions, so GoldenGate cannot capture table-level truncates if a table contains any empty partitions. For this reason, do not use GETTRUNCATES on any partitioned tables in those Oracle versions. GoldenGate DDL support can be used to capture truncates on tables that might include empty partitions.

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

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 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 GoldenGate. Before images contain column details that existed before a record was

updated. Use the GETUPDATEBEFORES parameter in the Extract parameter file to extract before images or in the Replicat parameter file to replicate before images.

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 GoldenGate's column-conversion functions 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 *GoldenGate for Windows and UNIX Administrator 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 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 GoldenGate DDL database objects that support the synchronization of Oracle DDL. This parameter is only valid for GoldenGate for Oracle.

Default None

Syntax GGSCHEMA <schema_name>

Argument	Description
<schema_name>	The name of the DDL schema.

GROUPTRANSOPS

Valid for Replicat

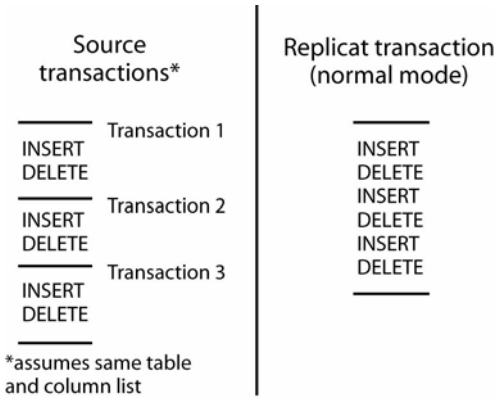
Syntax

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 GoldenGate's performance 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 19 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.

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 can occur during an initial load, when data from source tables is being loaded to target tables while GoldenGate is replicating transactional changes that are being made to those tables. When GoldenGate applies the 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.

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. See page 204.

Any of the preceding methods can be combined. The use within a MAP statement overrides other settings. The use 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 a duplicate-record error, the record applied by the initial load is overwritten by the change record. 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, the change record 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. GoldenGate extracts initial-load data from source table1, without record A.
6. 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.

HANDLECOLLLIONS should be disabled after the transactional changes captured during the initial load have been applied to the target tables, so that GoldenGate 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), or edit the file to add NOHANDLECOLLISIONS before MAP statements that you want it to affect.
- While Replicat is running, run GGSCI and use the SEND REPLICAT command with the NOHANDLECOLLISIONS option for the tables you want to affect (see page 56). 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.

Default None

Syntax
```
HANDLECOLLISIONS | NOHANDLECOLLISIONS
```

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 a transaction includes a transient primary key update. A transient primary key update is an update statement that affects the primary keys of multiple rows. This kind of statement typically

uses an x+n formula or some other manipulation that shifts the values such that one of the new values could be the same as one of the old ones.

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 on the target tables. Either INITIALLY DEFERRED or INITIALLY IMMEDIATE can be specified; Replicat alters the state to DEFERRED as needed. In this state, the constraints are only checked when the Replicat transaction is committed.

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 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 inserts to Oracle target tables. These parameters are valid only for Oracle databases.

INSERTAPPEND is appropriate for use as a performance improvement when the replicated transactions are large and contain multiple inserts into the same table. The best performance will be achieved when the BATCHSQL parameter is enabled. 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.

In a cascading configuration, do not use INSERTAPPEND for tables that will be a source for the local Extract process. GoldenGate cannot read redo log records that are generated for INSERTS that include an APPEND hint. The local Extract will skip those records, which may cause the target of that Extract to be out-of-sync with the source.

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.

For MAP syntax, see page 204.

Default NOINSERTAPPEND

Syntax INSERTAPPEND | NOINSERTAPPEND

Example In the following example, INSERTAPPEND is used for all tables in the fin schema, except for the inventory table.

```
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 *GoldenGate for Windows and UNIX Administrator Guide*.

This parameter also can be used within a MAP statement. See page 204.

Default None

Syntax INSERTALLRECORDS

INSERTDELETES | NOINSERTDELETES

Valid for Replicat

Use the INSERTDELETES and NOINSERTDELETES parameters to control whether or not

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.

Default NOINSERTDELETES

Syntax INSERTDELETES | NOINSERTDELETES

INSERTMISSINGUPDATES | NOINSERTMISSINGUPDATES

Valid for Replicat

Use the INSERTMISSINGUPDATES and NOINSERTMISSINGUPDATES parameters to control whether or not 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. 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 abort 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 GoldenGate converts uncompressed update operations to insert operations. The parameters are table-specific. One parameter remains in effect for all subsequent MAP statements, until the other parameter is encountered.

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 how often to report lag information to the error log. If the lag is greater than the value specified with the LAGCRITICAL parameter, Manager reports the lag as critical; otherwise, it reports the lag as an informational message. A value of zero (0) forces a message at the frequency specified with the LAGREPORTMINUTES or LAGREPORTHOURS parameter.

Default Do not report lag information

Syntax

```
LAGINFOSECONDS <seconds> |
LAGINFOMINUTES <minutes> |
LAGINFOHOURS <hours>
```

Argument	Description
`<seconds>`	The frequency, in seconds, to report lag information.
`<minutes>`	The frequency, in minutes, to report lag information.
`<hours>`	The frequency, in hours, to report lag information.

Example `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 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 GoldenGate's cache size 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 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 batch 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 a GoldenGate macro. For information about using macros, see the *GoldenGate for Windows and UNIX Administrator 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 are not case-sensitive. 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. 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. 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

Valid for Replicat

Use the MAP parameter to establish a relationship between one or more source and target objects. All target objects that you are synchronizing with 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.

You can specify the following objects with MAP:

- Tables
- Indexes
- Sequences
- Triggers
- Materialized views

Limitations of support

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 indexes, sequences, triggers, and materialized views, 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.

NOTE GoldenGate supports the replication of the actual data values of Oracle sequences and materialized views. GoldenGate supports the replication of DDL for indexes and triggers for both Oracle and Teradata, but not the content of those objects. See the *GoldenGate for Windows and UNIX Administrator Guide* for more information about DDL support.

Default None

Syntax

```
MAP <table spec>, TARGET <table spec>
[, DEF <definitions template>]
[, TARGETDEF <definitions template>]
[, COLMAP (<column mapping expression>)]
[, EVENTACTIONS (<action>)]
[, EXCEPTIONSONLY]
[, EXITPARAM "<parameter string>"]
[, FILTER (<filter specification>)]
[, HANDLECOLLISIONS | NOHANDLECOLLISIONS]
[, INSERTALLRECORDS]
[, INSERTAPPEND | NOINSERTAPPEND]
[, KEYCOLS (<column specification>)]
[, REPERROR (<error> , <response>)]
[, SQLEXEC (<SQL specification>)]
[, TRIMSPACES | NOTRIMSPACES]
[, WHERE (<where clause>)]
;
```

Table 42 Summary of MAP syntax components

Component	Description
MAP <table spec>	Specifies the source object. See "Object names and owners" on page 206.
TARGET <table spec>	Specifies the target object. See the guidelines following this table.
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.
EVENTACTIONS (<action>)	Triggers an action based on a record that satisfies a specified filter rule.
EXCEPTIONSONLY	Specifies error handling within an exceptions MAP statement.
EXITPARAM	Passes a parameter in the form of a literal string to a user exit.

Table 42 Summary of MAP syntax components (continued)

Component	Description
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.
REPERROR	Controls how Replicat responds to errors when executing the MAP statement.
SQLEXEC	Executes stored procedures and queries.
TRIMSPACES | NOTRIMSPACES	Controls whether trailing spaces are trimmed or not when mapping CHAR to VARCHAR columns.
WHERE	Selects records based on conditional operators.
;	Terminates the MAP statement and is required.

Object names and owners

Source and target object names must be fully qualified in GoldenGate parameter files, as in fin.emp.

Case sensitivity

Whether or not c-tree file and path names are case-sensitive depends on the requirements of the c-tree server host operating system.

If a database is case-sensitive, GoldenGate supports the case sensitivity of database names, owner names, object names, column names, and user names. Case-sensitive names must be specified in GoldenGate parameter files exactly as they appear in the database.

If a database is case-insensitive, or if it supports case-sensitivity but is configured to be case-insensitive, GoldenGate converts all names to upper case. The exception is Oracle 11g, where case-sensitive passwords are supported in GoldenGate input that requires passwords.

To preserve case-sensitivity

Case-sensitive names must be specified in GoldenGate parameter files exactly as they appear in the database. Enclose case-sensitive names in double quotes if the other database (the source or target of the case-sensitive objects) is not case-sensitive.

If replicating from a case-insensitive database to a case-sensitive database, the source object names must be entered in the Replicat MAP statements in upper case, to reflect the fact that they were written to the trail as uppercase by Extract.

For example:

```
MAP SALES.CUSTOMER, TARGET "Sales.Account";
```

NOTE Column names enclosed within quotes are treated as literals. For information about how GoldenGate observes case sensitivity for column names, see "Using COLMAP" on page 211.

Supported characters

GoldenGate supports alphanumeric characters in object names and the column names of key columns and non-key columns. GoldenGate also supports the following non-alphanumeric characters in columns that are not being used by GoldenGate as a key.

Table 43 Supported non-alphanumeric characters in object names and non-key column names[1]

Character	Description
~	Tilde
< >	Greater-than and less-than symbols
/	Forward slash
\	Backward slash
!	Exclamation point
@	At symbol
#	Pound symbol
$	Dollar symbol
%	Percent symbol
^	Carot symbol
( )	Open and close parentheses
_	Underscore
-	Dash
+	Plus sign
=	Equal symbol

Table 43 Supported non-alphanumeric characters in object names and non-key column names[1]

Character	Description
|	Pipe
[ ]	Begin and end brackets
{ }	Begin and end curly brackets (braces)

[1] The type of key that is being used by GoldenGate depends on the definition of a given table and whether there are any overrides by means of a KEYCOLS clause. GoldenGate will use a primary key, if available, or a unique key/index (selection is dependent on the database). In the absence of those definitions, all columns of the table are used, but a KEYCOLS clause overrides all existing key types. For columns that are being used by GoldenGate as a key, the characters in the names must be valid for inclusion in a WHERE clause. This list is all-inclusive; a given database platform may or may not support all listed characters.

Non-supported characters

GoldenGate does not support the following characters in object or column names:

Table 44 Non-supported characters in object and column names[1]

Character	Description
&	Ampersand
*	Asterisk
?	Question mark
:	Colon
;	Semi-colon
,	Comma
‘ ’	Single quotes
“ ”	Double quotes
‘	Accent mark (Diacritical mark)
.	Period
	Space

[1] This list is all-inclusive; a given database platform may or may not support all listed characters.

Using wildcards

The TABLE, SEQUENCE, and MAP parameters permit the use of wildcards to specify multiple objects in one statement. An asterisk (*) matches any number of characters.

By default, wildcarding is resolved in the following manner:

Source objects: If the name of the source object is stated explicitly, the resolution for that object and its target object occurs at process startup. When a source object name is wildcarded, the resolution for that object and its target object occurs when the first row for that source object is processed.

Target objects: When a target object is wildcarded, GoldenGate replaces the wildcard with the name of the corresponding source object. (See "Rules for using wildcards".)

The default behavior enables GoldenGate to capture source objects that are created after processing starts. To change the rules for resolving wildcards, use the WILDCARDRESOLVE parameter. The default is DYNAMIC.

You can combine the use of wildcard object selection with explicit object exclusion by using the TABLEEXCLUDE and MAPEXCLUDE parameters.

Rules for using wildcards

Observe the following rules when using wildcards:

- Use wildcards only for object names. Do not use wildcards for schema or database names.
- Target objects must exist in the target database for source objects that exist at startup, and for source objects that are added after startup.
- For source objects, you can use a partial name with a wildcard. For example, the following source specification is valid:

  ```
  hq.t_*;
  ```

- For target objects, you cannot use a wildcard with a partial name, because the asterisk in a wildcarded target name is replaced with the name of the source object. For example, the following statements would be *incorrect*:

  ```
  TABLE hq.t_*, TARGET rpt.t_*;
  MAP hq.t_*, TARGET rpt.t_*;
  ```

  They would produce the following results when resolved, because the wildcard in the target specification is replaced with T_TEST, the name of a source object:

  ```
  TABLE HQ.T_TEST1, TARGET RPT.T_T_TEST1;
  MAP HQ.T_TEST1, TARGET RPT.T_T_TEST1;
  ```

  The following examples show the correct use of wildcarding.

  ```
  TABLE hq.t_*, TARGET rpt.*;
  MAP hq.t_*, TARGET rpt.*;
  ```

  This enables the following correct result:

  ```
  TABLE HQ.T_TEST1, TARGET RPT.T_TEST1;
  MAP HQ.T_TEST1, TARGET RPT.T_TEST1;
  ```

Using Unicode and native encoding in a MAP statement

GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Windows, UNIX, and Linux

operating systems. An escape sequence can be used in the following elements within a TABLE or MAP statement:

- WHERE clause
- COLMAP clause to assign a Unicode character to a Unicode column, or to assign a native-encoded character to a column.
- GoldenGate column conversion functions within a COLMAP clause.

GoldenGate supports the following types of escape sequence:

- \uFFFF Unicode escape sequence
- \377 Octal escape sequence
- \xFF Hexadecimal escape sequence

The following limitations apply:

- This support is limited to UTF-16 code points from U+0000 to U+007F, the equivalent of 7-bit ASCII.
- The source and target columns must both be Unicode.
- The source and target data types must be identical (for example, NCHAR to NCHAR).

To use an escape sequence

Begin each escape sequence with a reverse solidus (code point U+005C), followed by the character code point. (A solidus is more commonly known as the backslash symbol.) Use the escape sequence, instead of the actual character, within your input string in the parameter statement or column-conversion function.

To use the \uFFFF Unicode escape sequence

- Must begin with a lowercase u, followed by exactly four hexadecimal digits.
- Supported ranges:
  - 0 to 9 (U+0030 to U+0039)
  - A to F (U+0041 to U+0046)
  - a to f (U+0061 to U+0066)
- This is the only permissible escape sequence to use for NCHAR and NVARCHAR columns.
- Surrogate pairs are not supported.

Example \u20ac is the Unicode escape sequence for the Euro currency sign.

> **NOTE** For reliable cross-platform support, use the Unicode escape sequence. Octal and hexadecimal escape sequences are not standardized on different operating systems.

To use the \377 octal escape sequence

- Must contain exactly three octal digits.
- Supported ranges:
  - Range for first digit is 0 to 3 (U+0030 to U+0033)
  - Range for second and third digits is 0 to 7 (U+0030 to U+0037)

Example \200 is the octal escape sequence for the Euro currency sign on Microsoft Windows

To use the \xFF hexadecimal escape sequence

- Must begin with a lowercase x followed by exactly two hexadecimal digits.
- Supported ranges:
  - 0 to 9 (U+0030 to U+0039)
  - A to F (U+0041 to U+0046)
  - a to f (U+0061 to U+0066)

Example \x80 is the hexadecimal escape sequence for the Euro currency sign on Microsoft Windows.

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.

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. 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 289
- ASSUMETARGETDEFS on page 117
- "Creating a data-definitions file" in the *GoldenGate for Windows and UNIX Administrator 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 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 illustrates what happens:

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 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.

For all databases except Sybase and SQL Server, column names are changed to upper case for name comparison. For Sybase and SQL Server, 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	Target table USER3.SM01
id	ID
owner	owner
created	id
changed	Creator
creator	comment
modifiedBy	ModifiedBy
comment	creationDate
COMMENT	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`

For more information about column mapping, see the *GoldenGate for Windows and UNIX Administrator Guide*.

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 supported characters in column names, see "Supported character types" on page 207. For rules that apply when using Unicode or native encoded strings or columns, see page 209. <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 a GoldenGate column-conversion function, such as @STREXT (COL1, 1, 3). For descriptions of GoldenGate 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.
`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 212 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 COLMAP ( EUROVAL = "\u20ac0" );
```

Example 3
```
MAP ggs.tran, TARGET ggs.tran2, COLMAP (SECTION = @STRCAT("\u00a7", SECTION
));
```

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 *GoldenGate for Windows and UNIX Administrator 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 GoldenGate processing based on database events.

NOTE 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 336.

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 or stop a process, to perform a transformation, or to report statistics. The event marker system can be put to use for purposes such as:

- To establish a synchronization point at which SQLEXEC or user exit functions can be performed
- To execute a shell command that executes a data validation script
- To activate tracing when a specific account number is detected
- To capture lag history
- To establish a point at which to start batch processes or end-of-day reporting procedures

The event marker feature is supported for the replication of data changes, but not for initial loads.

To use the event marker system

The system requires the following input components:

1. Specify the *event record* that will trigger the action. You can do this by including a FILTER or WHERE clause, or a SQLEXEC query or procedure, in one of the following parameter statements:
   - TABLE statement in an Extract parameter file
   - MAP statement in a Replicat parameter file
   - Special TABLE statement in a Replicat parameter file that enables you to perform EVENTACTIONS actions without mapping a source table to a target table

2. In the same TABLE or MAP statement where you specified the event record, include the EVENTACTIONS parameter with the appropriate option to specify the action that is to be taken by the process.

> **NOTE** 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
  - ABORT
  - CHECKPOINT AFTER
  - FORCESTOP
  - STOP

To control the processing of the event record itself

To prevent the event record itself from being processed in the normal manner, use the IGNORE or DISCARD option. Because IGNORE and DISCARD are evaluated before the record itself, they prevent the record from being processed. Without those options, Extract writes the record to the trail, and Replicat applies the operation that is contained in the record to the target database.

You should take into account the possibility that a transaction could contain two or more records that trigger an event action. In such a case, there could be multiple executions of certain EVENTACTIONS specifications. For example, encountering two qualifying records that trigger two successive ROLLOVER actions will cause Extract to roll over the trail twice, leaving one of the two essentially empty.

Syntax

```
EVENTACTIONS (
[STOP | ABORT | FORCESTOP]
[IGNORE [TRANSACTION [INCLUDEVENT]]
[DISCARD]
[LOG [INFO | WARNING]]
[REPORT]
[ROLLOVER]
[SHELL <command>]
[TRACE <trace file> [TRANSACTION] [PURGE | APPEND]]
[CHECKPOINT [BEFORE | AFTER | BOTH]]
[, ...]
)
```

Action	Description
STOP	Brings the process to a graceful stop when the specified event record is encountered. The process waits for open transactions to be completed before stopping. If the transaction is a Replicat grouped or batched transaction, the current group of transactions are applied before the process stops gracefully. The process restarts at the next record after the event record, so long as that record also signified the end of a transaction. The process logs a message if it cannot stop immediately because a transaction is still open. However, if the event record is encountered within a long-running open transaction, there is no warning message that alerts you to the uncommitted state of the transaction. Therefore, the process may remain running for a long time despite the STOP event. STOP can be combined with other EVENTACTIONS options except for ABORT and FORCESTOP.
ABORT	Forces the process to exit immediately when the specified event record is encountered, whether or not there are open transactions. The event record is not processed. A fatal error is written to the log, and the event record is written to the discard file if DISCARD is also specified. The process will undergo recovery on startup. ABORT can be combined only with CHECKPOINT BEFORE, DISCARD, SHELL, and REPORT.

Action	Description
FORCESTOP	Forces the process to stop gracefully when the specified event record is encountered, but only if the event record is the last operation in the transaction or the only record in the transaction. The record is written normally. If the event record is encountered within a long-running open transaction, the process writes a warning message to the log and exits immediately, as in ABORT. In this case, recovery may be required on startup. If the FORCESTOP action is triggered in the middle of a long-running transaction, the process exits without a warning message. FORCESTOP can be combined with other EVENTACTIONS options except for ABORT, STOP, CHECKPOINT AFTER, and CHECKPOINT BOTH. If used with ROLLOVER, the rollover only occurs if the process stops gracefully.
IGNORE [TRANSACTION [INCLUDEVENT]]	By default, forces the process to ignore the specified event record. No warning or message is written to the log, but the GoldenGate statistics are updated to show that the record was ignored. ◆ Use TRANSACTION to ignore the entire transaction that contains the record that triggered the event. If TRANSACTION is used, the event record must be the first one in the transaction. When ignoring a transaction, the event record is also ignored by default. TRANSACTION can be shortened to TRANS. ◆ Use INCLUDEEVENT with TRANSACTION to propagate the event record to the trail or to the target, but ignore the rest of the associated transaction. IGNORE can be combined with all other EVENTACTIONS options except ABORT and DISCARD.
DISCARD	Causes the process to: ◆ write the specified event record to the discard file. ◆ update the GoldenGate statistics to show that the record was discarded. The process resumes processing with the next record in the trail. When using this option, use the DISCARDFILE parameter to specify the name of the discard file. By default, a discard file is not created. DISCARD can be combined with all other EVENTACTIONS options except IGNORE.

Action	Description
LOG [INFO | WARNING]	Causes the process to log the event when the specified event record is encountered. The message is written to the report file, to the GoldenGate error log, and to the system event log. Use the following options to specify the severity of the message: ◆ INFO specifies a low-severity informational message. This is the default. ◆ WARNING specifies a high-severity warning message. LOG can be combined with all other EVENTACTIONS options except ABORT. If using ABORT, LOG is not needed because ABORT logs a fatal error before the process exits.
REPORT	Causes the process to generate a report file when the specified event record is encountered. This is the same as using the SEND command with the REPORT option in GGSCI. The REPORT message occurs after the event record is processed (unless DISCARD, IGNORE, or ABORT are used), so the report data will include the event record. REPORT can be combined with all other EVENTACTIONS options.
ROLLOVER	Valid only for Extract. Causes Extract to roll over the trail to a new file when the specified event record is encountered. The ROLLOVER action occurs before Extract writes the event record to the trail file, which causes the record to be the first one in the new file unless DISCARD, IGNORE or ABORT are also used. ROLLOVER can be combined with all other EVENTACTIONS options except ABORT. Note: ROLLOVER cannot be combined with ABORT because: ◆ ROLLOVER does not cause the process to write a checkpoint. ◆ ROLLOVER happens before ABORT. Without a ROLLOVER checkpoint, ABORT causes Extract to go to its previous checkpoint upon restart, which would be in the previous trail file. In effect, this cancels the rollover.

Action	Description
`SHELL <command>`	Causes the process to execute the specified shell command when the specified event record is encountered. ◆ <command> specifies the system or shell command to be issued. ◆ If the shell command is successful, the process writes an informational message to the report file and to the event log. Success is based upon the exit status of the command in accordance with the UNIX shell language. In that language, zero indicates success. ◆ If the system call is not successful, the process abends with a fatal error. In the UNIX shell language, non-zero equals failure. SHELL can be combined with all other EVENTACTIONS options.

`TRACE <trace file>`
`[TRANSACTION]`
`[PURGE | APPEND]`
	Causes process trace information to be written to a trace file when the specified event record is encountered.
	By default, tracing is enabled until the process terminates. To set the trace level, use the GoldenGate TRACE or TRACE2 parameter.
	◆ <trace file> specifies the name of the trace file and must appear immediately after the TRACE keyword. You can specify a unique trace file, or use the default trace file that is specified with the standalone TRACE or TRACE2 parameter.
	The same trace file can be used across different TABLE or MAP statements in which EVENTACTIONS TRACE is used. If multiple TABLE or MAP statements specify the same trace file name, but the TRACE options are not used consistently, preference is given to the options in the last resolved TABLE or MAP that contains this trace file.
	◆ Use TRANSACTION to enable tracing only until the end of the current transaction, instead of when the process terminates. For Replicat, transaction boundaries are based on the source transaction, not the typical Replicat grouped or batched target transaction. TRANSACTION can be shortened to TRANS.
	◆ Use PURGE to truncate the trace file before writing additional trace records, or use APPEND to write new trace records at the end of the existing records. APPEND is the default.
	TRACE can be combined with all other EVENTACTIONS options except ABORT.
	To disable tracing to the specified trace file, issue the GGSCI SEND <process> command with the TRACE OFF <filename> option.

Action	Description
`CHECKPOINT [BEFORE | AFTER | BOTH]`	Causes the process to write a checkpoint when the specified event record is encountered. Checkpoint actions provide a context around the processing that is defined in TABLE or MAP statements. This context has a begin point and an end point, thus providing synchronization points for mapping the functions that are performed with SQLEXEC and user exits. ◆ BEFORE BEFORE for an Extract process writes a checkpoint before Extract writes the event record to the trail. BEFORE for a Replicat process writes a checkpoint before Replicat applies the SQL operation that is contained in the record to the target. BEFORE requires the event record to be the first record in a transaction. If it is not the first record, the process will abend. Use BEFORE to ensure that all transactions prior to the one that begins with the event record are committed. CHECKPOINT BEFORE can be combined with all EVENTACTIONS options. ◆ AFTER AFTER for Extract writes a checkpoint after Extract writes the event record to the trail. AFTER for Replicat writes a checkpoint after Replicat applies the SQL operation that is contained in the record to the target. AFTER flags the checkpoint request as an advisory, meaning that the process will only issue a checkpoint at the next practical opportunity. For example, in the case where the event record is one of a multi-record transaction, the checkpoint will take place at the next transaction boundary, in keeping with the GoldenGate data-integrity model. CHECKPOINT AFTER can be combined with all EVENTACTIONS options except ABORT. ◆ BOTH BOTH combines BEFORE and AFTER. The Extract or Replicat process writes a checkpoint before and after it processes the event record. CHECKPOINT BOTH can be combined with all EVENTACTIONS options except ABORT. CHECKPOINT can be shortened to CP.

Example 1 The following example shows how you can configure a process to ignore certain records. When Replicat processes any trail record that has col1 = goldengate, it ignores the record.

```
MAP <owner.table>, TARGET <owner2.table2>, &
FILTER (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 aborts.

```
MAP <owner.table>, TARGET <owner2.table2>, &
FILTER (name = goldengate), &
EVENTACTIONS (DISCARD, ABORT);
```

Example 3 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 *GoldenGate for Windows and UNIX Administrator Guide*.

Using EXCEPTIONSONLY

Use EXCEPTIONSONLY in an exceptions MAP statement intended for error handling. It causes the MAP statement in which it is specified to be executed only if an error occurs for the last record processed in the preceding MAP statement.

To use EXCEPTIONSONLY, set the REPERROR parameter to EXCEPTION for the error to be handled as an exception. Make certain that the exceptions MAP statement specifies the same SOURCE table as in the MAP statement for which the error is anticipated. Set the TARGET table to the name of the exceptions table. For more information about REPERROR, see page 259.

The exceptions MAP statement must follow the MAP statement for which the error is anticipated. For more information about using an exceptions MAP statement, see the *GoldenGate for Windows and UNIX Administrator 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, 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))`
	Enclose text strings within double quotes, as in `FILTER (@STRFIND(NAME, "JOE") > 0).`
	You can use GoldenGate's column-conversion functions in a filter clause, as in:
	`FILTER (@COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`

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 GoldenGate. For example: RAISEERROR 21000.

Using HANDLECOLLISIONS | NOHANDLECOLLISIONS

Use HANDLECOLLISIONS and NOHANDLECOLLISIONS to control whether or not GoldenGate adjusts for transactional changes to source tables that were replicated by 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 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 193.

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 ggs
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 *GoldenGate for Windows and UNIX Administrator 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 196.

Syntax

```
MAP <table spec>, TARGET <table spec>, INSERTALLRECORDS;
```

INSERTAPPEND | NOINSERTAPPEND

Use the INSERTAPPEND and NOINSERTAPPEND parameters to control whether or not Replicat uses an APPEND hint when it applies inserts to Oracle target tables. These parameters are valid only for Oracle databases.

INSERTAPPEND is appropriate for use as a performance improvement when the replicated transactions are large and contain multiple inserts into the same table. The best performance will be achieved when the BATCHSQL parameter is enabled. 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.

In a cascading configuration, do not use INSERTAPPEND for tables that will be a source for the local Extract process. GoldenGate cannot read redo log records that are generated for INSERTS that include an APPEND hint. The local Extract will skip those records, which may cause the target of that Extract to be out-of-sync with the source.

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.

The default for Replicat INSERT statements is NOINSERTAPPEND.

See also INSERTAPPEND | NOINSERTAPPEND on page 195 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. 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 log-based extraction only).

On the target tables, create a unique index on the KEYCOLS-defined key columns. An index improves the speed with which 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.

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 (see page 259). 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>]}
)
[, ...];
```

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.
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.

Argument	Description
EXCEPTION	Handle the error as an exception. In anticipation of possible errors, you can create an exceptions MAP statement that executes only after an error. Use that MAP statement, for example, to map columns from a failed update statement into a "missing update" table. In the parameter file, specify the exceptions MAP statement after the MAP statement for which the error is anticipated. For more information about error handling, see the *GoldenGate for Windows and UNIX Administrator Guide*.
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 269.
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.

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.

Using SQLEXEC

Use SQLEXEC to execute a SQL stored procedure or query from within a MAP statement during GoldenGate processing. SQLEXEC enables 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.

WARNING A query or procedure must be structured correctly when executing a SQLEXEC statement. If Replicat encounters a problem with the query or procedure, then the process will immediately abend, regardless of any error-handling rules that are in place.

Supported databases and data types

The following are the databases that are supported by SQLEXEC and the data types that are supported for input and output parameters:

DB2 LUW and z/OS

- CHAR
- VARCHAR
- DATE
- All numeric data types.
- BLOB data types

Ingres

All data types except LOB data types.

Oracle

- CHAR
- VARCHAR2
- DATE
- The ANSI equivalents of these types.

- All numeric data types.

SQL Server

- CHAR
- VARCHAR
- DATETIME
- All numeric data types.
- Image and text data types where the length is less than 200 bytes.
- TIMESTAMP parameter types are not supported natively, but you can use other data types as parameters and convert the values to TIMESTAMP format within the stored procedure.

Sybase

All data types except TEXT, IMAGE, and UDT.

Teradata

All Teradata data types that are supported by GoldenGate.

For additional instructions for using stored procedures and queries with GoldenGate, see the *GoldenGate for Windows and UNIX Administrator Guide*.

SQLEXEC dependencies and restrictions

- The SQL is executed by the database user under which the 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 226.
- For DB2 on z/OS, 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 GoldenGate users. See the DB2 for z/OS documentation for more information.

When using 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.

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. 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 234.

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.

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 contained on one line, within quotes. 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 234.

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

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 239).

- 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 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 398.

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, for example:

  ```
  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"

Column name	Description
custid	Number
current_residence_state	Char(2)
birth_state	Char(2)

Target table "cust_extended"

Column name	Description
custid	Number

Column name	Description
current_residence_state_long	Varchar(30)
birth_state_long	Varchar(30)

Lookup table "state_lookup"

Column name	Description
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. 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 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 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. GoldenGate continues processing after reporting the error.

Action	Description
RAISE	Handles errors according to rules set by a REPERROR parameter. 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 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 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.
ONCE	Executes the procedure or query once during the course of the 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 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. 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 GoldenGate column-conversion function that is supplying the input. Either a PARAMS clause or NOPARAMS is required.

The following are the databases that are supported by SQLEXEC and the data types that are supported for input and output parameters:

DB2 LUW and z/OS

- CHAR
- VARCHAR
- DATE
- All numeric data types.
- BLOB data types

Ingres

All data types except LOB data types.

Oracle

- CHAR
- VARCHAR2
- DATE
- The ANSI equivalents of these types.
- All numeric data types.

SQL Server

- CHAR

- VARCHAR
- DATETIME
- All numeric data types.
- Image and text data types where the length is less than 200 bytes.
- TIMESTAMP parameter types are not supported natively, but you can use other data types as parameters and convert the values to TIMESTAMP format within the stored procedure.

Sybase

All data types except TEXT, IMAGE, and UDT.

Teradata

All Teradata data types that are supported by GoldenGate.

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.

Component	Description
	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. ◆ <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 a 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, 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.

Action	Description
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 *GoldenGate for Windows and UNIX Administrator 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 are truncated when posting to the target table. These parameters only affect CHAR to VARCHAR column mappings. The default is TRIMSPACES.

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 trims 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 WHERE

Use WHERE to select records based on a conditional statement. To use a Unicode column or a string that contains extended ASCII or unprintable characters, see page 209. For full

instructions on using a WHERE clause, see the *GoldenGate for Windows and UNIX Administrator Guide*.

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 45 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 GoldenGate.
Comparison operators	=, <>, >, <, >=, <=
Conjunctive operators	AND, OR
Grouping parentheses	Use open and close parentheses for logical grouping of multiple elements.

Example The following WHERE clauses demonstrate the use of a Unicode escape sequence in a WHERE clause that can be part of a TABLE or MAP statement. Each escapes the Latin lowercase e with an acute (code point U+00E9) in a different location within the input strings of “Élise,” “Zoé,” and “Véronique.”

```
WHERE (FIRSTNAME <> "\u00e9lise”)
WHERE (FIRSTNAME <> "Zo\u00e9")
WHERE (FIRSTNAME <> "V\u00e9ronique")
```

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 GoldenGate installation directory.

For more information about the marker table and params.sql, see the *GoldenGate for Windows and UNIX Administrator 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. Extract may need to fetch data directly from a table or from the undo tablespace (Oracle 9i and later). 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`

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 119). 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 GoldenGate Technical Support.

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 as of GoldenGate version 10, when Extract recovers from a failure, it will re-send 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 support case with GoldenGate.

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 GoldenGate instances or when also installing the 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`

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 file contains only insert records pertaining to a single table type 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 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 115).

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. GoldenGate's memory 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. After processing the parameters in the other file, GoldenGate resumes processing of the current parameter file.

To use OBEY, first create and save a parameter file containing the parameters that you want to retrieve. Use OBEY in the current parameter file to invoke the other file. OBEY statements cannot be nested within other OBEY statements.

Instead of using OBEY, or in addition to using it, you can use GoldenGate macros to call frequently used parameters. For more information about using macros, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Default None

Syntax
```
OBEY <file name>
```

Argument	Description
`<file name>`	The fully qualified name of the file from which to retrieve parameters or commands.

Example
```
OBEY /home/ggs/myparams
```

OUTPUTFILEUMASK

Valid for GLOBALS

Use the OUTPUTFILEUMASK parameter to specify an octal umask that will be used by GoldenGate processes to create trail files and discard files. This parameter 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 GoldenGate 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 abort. For more information about exceptions maps, see the *GoldenGate for Windows and UNIX Administrator 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 292) 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 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 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 sample parameter file passes through all data from fin.acct, but allows normal processing for fin.sales.

```
EXTRACT fin
RMTHOST sysb, MGRPORT 7809
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 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. GoldenGate first looks for a name specified with the DDLTABLE <table> parameter in the GLOBALS file or, if that parameter does not exist, GoldenGate uses the default name of GGS_DDL_HIST.

NOTE For additional information about purging the DDL history table, see the *GoldenGate for Windows and UNIX Administrator 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 None (Old rows are not purged)

Syntax

```
PURGEDDLHISTORY
{, <max rule>}
[, <min rule>]
```

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.

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.

```
PURGEDDLHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 5
```

PURGEMARKERHISTORY

Valid for Manager

Use the PURGEMARKERHISTORY parameter to control the size of the 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. GoldenGate first looks for a name specified with the MARKERTABLE <table> parameter in the GLOBALS file or, if that parameter does not exist, 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 *GoldenGate for Windows and UNIX Administrator Guide*.

PURGEMARKERHISTORY requires a logon with the USERID parameter and, if required, the SOURCEDB parameter.

Default None (Old rows are not purged)

Syntax

```
PURGEMARKERHISTORY
{, <max rule>}
[, <min rule>]
```

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.

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.

```
PURGEMARKERHISTORY MINKEEPDAYS 3, MAXKEEPDAYS 5
```

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 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 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 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.

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, 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 129). 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.

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 GoldenGate. It will cause PURGEOLDEXTRACTS to function improperly.

Default USECHECKPOINTS

Syntax

```
PURGEOLDEXTRACTS <trail name>
[, USECHECKPOINTS | NOUSECHECKPOINTS]
[, <minkeep rule>]
[, <frequency>]
```

Argument	Description
<trail name>	The trail to purge. Use the 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.

Argument	Description
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 129). 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 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`

Argument	Description
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 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.

Best practice

Do not change RECOVERYOPTIONS from the default unless instructed to do so by a GoldenGate Technical Support analyst. If Extract is permitted to overwrite existing data in the trail, it might be harder for 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 GoldenGate that is being used on the target is older than 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 EXTTRAIL, RMTTRAIL, EXTFILE, and RMTFILE. 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. 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 abort 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
```

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>]
}) |
RESET }
```

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.
`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 within 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 can create an exceptions MAP statement that executes only after an error. Use that MAP statement, for example, to map columns from a failed update statement into a "missing update" table. In the parameter file, specify the exceptions MAP statement after the MAP statement for which the error is anticipated. For more information about error handling, see the *GoldenGate for Windows and UNIX Administrator Guide*.
`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 269.

Argument	Description
TRANSABORT [, MAXRETRIES <n>] [, DELAYSECS <n> | DELAYCSECS <n>]	Abort the transaction and reposition 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.
RESET	Removes all error rules specified with previous REPERROR statements in the parameter file. Restores the default of ABEND on all errors. Replicat rolls back the transaction and terminates processing abnormally. This option is not available for use with REPERROR in a MAP statement.

Example 1 The following example demonstrates how to abort 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;
```

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 from either the table or the undo tablespace (Oracle 9i and later) when the transaction record does not contain enough information to construct a SQL statement or when a FETCHCOLS clause is used (see page 315).

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). The default action taken by Replicat is one of the following: ◆ For Oracle 9i and 10g, Replicat logs the row to the discard file, and continues processing subsequent data. ◆ For Oracle 8i, Replicat ignores the operation 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 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 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. **Column** — **Default** 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 211.)
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.

Default NONE

Syntax REPLACEBADCHAR {<char> | SPACE | NULL | UNPRINTABLE | NONE}

Argument	Description
<char>	Replace with the specified 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.

Argument	Description
NONE	Suppress transformation of double-byte character set values to default characters. This is the default.

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 zero.

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. SPECIALRUN specifies a batch run. See page 290.

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 296.

For more information about using process reports, see the *GoldenGate for Windows and UNIX Administrator 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 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 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, 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 2006-08-31 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 GoldenGate has stopped because of a conflict. By default, NORESTARTCOLLISIONS applies. However, there might be circumstances when you would want 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 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 193.

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 259).

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 batch processing. For online change synchronization, use the RMTTRAIL parameter.

On Solaris systems, the size of an extract file cannot exceed 2GB if it will be processed by Extract or Replicat. The size can be larger if the file will be processed by another application, such as a native bulk load utility.

RMTFILE must be preceded by a RMTHOST statement, and it must precede any TABLE statements.

About file versioning

Because all of the GoldenGate processes are decoupled and thus can be of different GoldenGate versions, each trail file or extract file has a version that is stored in the file header. By default, the version of a trail is the current version of the process that created the file. To set the version of a trail, use the FORMAT option of the EXTTRAIL, EXTFILE, RMTTRAIL, or RMTFILE parameter.

To ensure forward and backward compatibility of files among different GoldenGate process versions, the file header fields are written in a standardized token format. New tokens that are created by new versions of a process can be ignored by older versions, so that backward compatibility is maintained. Likewise, newer GoldenGate versions support older tokens. Additionally, if a token is deprecated by a new process version, a default value is assigned to the token so that older versions can still function properly. The token that specifies the file version is COMPATIBILITY and can be viewed in the Logdump utility and also by retrieving it with the GGFILEHEADER option of the @GETENV function.

A trail or extract file must have a version that is equal to, or lower than, that of the process that *reads* it. Otherwise the process will abend. Additionally, GoldenGate forces the output trail or file of a data pump to be the same version as that of its input trail or file. Upon restart, Extract rolls a trail to a new file to ensure that each file is of only one version (unless the file is empty).

Parameter dependencies

There is a dependency between the RECOVERYOPTIONS parameter and the FORMAT option of EXTTRAIL, RMTTRAIL, EXTFILE, and RMTFILE. 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 None

Syntax

```
RMTFILE <file name>
[, APPEND]
[, PURGE]
[, MAXFILES <number>]
[, MEGABYTES <megabytes>]
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<file name>`	The 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.

Argument	Description
`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).
`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 format depends on the version of the GoldenGate process; older GoldenGate versions contain less, or different, metadata than newer ones. The metadata tells the process whether the data records are of a version that it supports. ◆ FORMAT is a required keyword. ◆ RELEASE specifies a GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current GoldenGate version number. (If you use a 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. If append mode is not enabled for Extract, the file version number automatically defaults to 9 (GoldenGate version 9.x). The Extract mode is controlled by the RECOVERYOPTIONS parameter.

Example 1 `RMTFILE /ggs/dirdat/salesny, MEGABYTES 3, PURGE`

Example 2 `RMTFILE /ggs/dirdat/salesny, MEGABYTES 3, FORMAT RELEASE 10.0`

RMTHOST

Valid for Extract

Use the RMTHOST parameter to identify a remote system and the TCP/IP port number on that system where the Manager process is running, and to control various attributes of the TCP/IP connections made by GoldenGate between source and target systems. This parameter controls compression, data encryption, buffer attributes, 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 ggs123
RMTHOST ny, MGRPORT 7888
RMTTRAIL /ggs/dirdat/aa
TABLE ora.orders;
RMTHOST la, MGRPORT 7888
RMTTRAIL /ggs/dirdat/bb
TABLE ora.orders;
```

Do not use RMTHOST for an Extract created in PASSIVE mode. See page 20 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 RTT is 0.08 seconds and the bandwidth is 100 Mbps (megabits per second), then the optimum buffer size is:

   0.08 * 100Mbps = 8 Mbps

3. Network bandwidth is measured in bits per second, so divide the result by 8 to determine the number of bytes (8 bits to a byte). For example, the preceding sample result translates to 1 megabyte per second; therefore, you would set TCPBUFSIZE to a value of 1000000 (the required unit is bytes).

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 GoldenGate can increase the buffer size configured with TCPBUFSIZE.

NOTE Performance improvements are seen only when the target GoldenGate version is 8.0.4 or higher.

Default None

Syntax

```
RMTHOST
{<host name> | <IP address>}
{, MGRPORT <port> | PORT <port>}
[, COMPRESS]
[, COMPRESSTHRESHOLD]
[, ENCRYPT {NONE | BLOWFISH}]
[, KEYNAME <keyname>]
[, PARAMS <collector parameters>]
[, TCPBUFSIZE <bytes>]
[, TCPFLUSHBYTES <bytes>]
[, TIMEOUT <seconds>]
```

Argument	Description
`{<host name> | <IP address>}`	The DNS name or IP address of the target system. You can use either one to define the host.
`COMPRESS`	Compresses outgoing blocks of records to reduce bandwidth requirements. 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 {NONE | BLOWFISH}`	Encrypts the data stream sent over TCP/IP to the target system. ◆ NONE specifies no encryption (the default). ◆ BLOWFISH specifies Blowfish encryption. Requires using the KEYNAME option. Data is decrypted automatically on the target system, so no decryption parameters are required. For more information about using data encryption, see the *GoldenGate for Windows and UNIX Administrator Guide*.
`KEYNAME <keyname>`	A key name in the ENCKEYS file. GoldenGate uses the key to decrypt the data. Unless a matching key name exists in the ENCKEYS file on the target system, GoldenGate abends. For more information about creating an ENCKEYS file, see the *GoldenGate for Windows and UNIX Administrator Guide*.
`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.

Argument	Description
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 *Reference Guide*.
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 272. If the 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>	Defines how long Collector waits for a connection from Extract. The minimum value is 1 second, but setting the timeout to a very low value is not recommended in a production setting. The maximum value is 1800 seconds (30 minutes). The default value is 300 seconds (5 minutes). An orphaned Collector process will close itself after the defined timeout. This parameter does not affect a static Collector.

Example 1 `RMTHOST 20.20.20.17, MGRPORT 7809, ENCRYPT blowfish, KEYNAME newyork`

Example 2 `RMTHOST newyork, MGRPORT 7809, COMPRESS, COMPRESSTHRESHOLD 750`

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, 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 GoldenGate in a zoned network, see the *GoldenGate for Windows and UNIX Administrator 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.

Default None

Syntax

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

Argument	Description
COMPRESS	Compresses outgoing blocks of records to reduce bandwidth requirements. 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 {NONE | BLOWFISH}	Encrypts the data stream sent over TCP/IP to the target system. ◆ NONE specifies no encryption (the default). ◆ BLOWFISH specifies Blowfish encryption. Requires using the KEYNAME option. Data is decrypted automatically on the target system, so no decryption parameters are required. For more information about using data encryption, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Argument	Description
KEYNAME <keyname>	A key name in the ENCKEYS file. GoldenGate uses the key to decrypt the data. Unless a matching key name exists in the ENCKEYS file on the target system, GoldenGate abends. For more information about creating an ENCKEYS file, see the *GoldenGate for Windows and UNIX Administrator Guide*.
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 *Reference Guide*.
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 packet sizes 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 272. If the 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 an Extract running in PASSIVE mode waits for a connection from Collector. The minimum value is 1 second, but setting the timeout to a very low value is not recommended in a production setting. The maximum value is 1800 seconds (30 minutes). The default value is 300 seconds (5 minutes). An orphaned Collector process will close itself after the defined timeout. This parameter does not affect a static Collector.

Example RMTHOSTOPTIONS ENCRYPT blowfish, KEYNAME newyork, COMPRESS, COMPRESSTHRESHOLD 750, TCPBUFSIZE 100000, TCPFLUSHBYTES 300000

RMTTASK

Valid for Extract

Use the RMTTASK parameter for an initial-load Extract to initiate a Replicat processing task during a 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.
- SPECIALRUN must be used in the initial-load Replicat parameter file.

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 user-defined types (UDT).

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 *GoldenGate for Windows and UNIX Administrator 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.

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 format depends on the version of the GoldenGate process; older GoldenGate versions contain less, or different, metadata than newer ones. The metadata tells the process whether the data records are of a version that it supports. ◆ FORMAT is a required keyword. ◆ RELEASE specifies a GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current GoldenGate version number. (If you use a 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. If append mode is not enabled for Extract, the file version number automatically defaults to 9 (GoldenGate version 9.x). The Extract mode is controlled by the RECOVERYOPTIONS parameter.

Example `RMTTASK REPLICAT, GROUP initrep`

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.

About file versioning

Because all of the GoldenGate processes are decoupled and thus can be of different GoldenGate versions, each trail file or extract file has a version that is stored in the file header. By default, the version of a trail is the current version of the process that created the file. To set the version of a trail, use the FORMAT option of the EXTTRAIL, EXTFILE, RMTTRAIL, or RMTFILE parameter.

To ensure forward and backward compatibility of files among different GoldenGate process versions, the file header fields are written in a standardized token format. New tokens that are created by new versions of a process can be ignored by older versions, so that backward compatibility is maintained. Likewise, newer GoldenGate versions support older tokens. Additionally, if a token is deprecated by a new process version, a default value is assigned to the token so that older versions can still function properly. The token that specifies the file version is COMPATIBILITY and can be viewed in the Logdump utility and also by retrieving it with the GGFILEHEADER option of the @GETENV function.

Parameter dependencies

There is a dependency between the RECOVERYOPTIONS parameter and the FORMAT option of EXTTRAIL, RMTTRAIL, EXTFILE, and RMTFILE. 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 None

Syntax
```
RMTTRAIL <name>
[, FORMAT RELEASE <major>.<minor>]
```

Argument	Description
`<name>`	The 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 format depends on the version of the GoldenGate process; older GoldenGate versions contain less, or different, metadata than newer ones. The metadata tells the process whether the data records are of a version that it supports. ◆ FORMAT is a required keyword. ◆ RELEASE specifies a GoldenGate release version. <major> is the major version number, and <minor> is the minor version number. Valid values are 9.0 through the current GoldenGate version number. (If you use a 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. If append mode is not enabled for Extract, the file version number automatically defaults to 9 (GoldenGate version 9.x). The Extract mode is controlled by the RECOVERYOPTIONS parameter.

Example 1
```
RMTTRAIL /ggs/dirdat/ny
```

Example 2
```
RMTTRAIL /ggs/dirdat/ny, FORMAT RELEASE 10.0
```

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 processing 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.

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 a GoldenGate trail and delivery to another database. Currently, 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 GoldenGate DDL support environment. You can just use the SEQUENCE parameter.

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 * source_RAC_nodes) <=
target_highwater_value
```

Or...

```
source_highwater_value + (source_cache_size * source_increment_size * source_RAC_nodes) >=
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, and the source INCREMENTBY value is 1, and there are two source RAC nodes, the target highwater value should be at least 80:

```
40 + (20*1*2) <80
```

If the target highwater value is less than 80, 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

- GoldenGate online and batch (SPECIALRUN) change synchronization methods support the replication of sequence values.
- 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.
- GoldenGate does not support the replication of sequence values in a bi-directional configuration.
- 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 GoldenGate in a high-availability configuration, see the *GoldenGate for Windows and UNIX Administrator 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 source and target must be set the same way.
- To add SEQUENCE to a configuration in which DDL support is enabled, you must re-install the GoldenGate DDL objects in INITIALSETUP mode.

Error handling

- If Extract cannot resolve a sequence name, it writes a message to the trace file if the TLTRACE parameter is present and 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 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, unless the NOCHECKSEQUENCEVALUE parameter is used (see page 131).
- 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 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.

Object names and owners

Source and target object names must be fully qualified in GoldenGate parameter files with both the schema and name, as in scott.emp.

Case sensitivity

GoldenGate is case-insensitive for Oracle and converts lower and mixed-case owner, database, and object names to upper case. This is true whether or not the database itself is set to support case sensitivity.

Supported characters

GoldenGate supports alphanumeric characters in object names and the column names of key columns and non-key columns. GoldenGate also supports the following non-alphanumeric characters in columns that are not being used by GoldenGate as a key.

Table 46 Supported non-alphanumeric characters in object names and non-key column names[1]

Character	Description
~	Tilde
< >	Greater-than and less-than symbols
/	Forward slash
\	Backward slash
!	Exclamation point
@	At symbol
#	Pound symbol
$	Dollar symbol
%	Percent symbol
^	Carot symbol
( )	Open and close parentheses
_	Underscore
-	Dash
+	Plus sign
=	Equal symbol
|	Pipe

Table 46 Supported non-alphanumeric characters in object names and non-key column names[1]

Character	Description
[ ]	Begin and end brackets
{ }	Begin and end curly brackets (braces)

[1] The type of key that is being used by GoldenGate depends on the definition of a given table and whether there are any overrides by means of a KEYCOLS clause. GoldenGate will use a primary key, if available, or a unique key/index (selection is dependent on the database). In the absence of those definitions, all columns of the table are used, but a KEYCOLS clause overrides all existing key types. For columns that are being used by GoldenGate as a key, the characters in the names must be valid for inclusion in a WHERE clause. This list is all-inclusive; a given database platform may or may not support all listed characters.

Non-supported characters

GoldenGate does not support the following characters in object or column names:

Table 47 Non-supported characters in object and column names[1]

Character	Description
&	Ampersand
*	Asterisk
?	Question mark
:	Colon
;	Semi-colon
,	Comma
‘ ’	Single quotes
“ ”	Double quotes
‘	Accent mark (Diacritical mark)
.	Period
	Space

[1] This list is all-inclusive; a given database platform may or may not support all listed characters.

Using wildcards

The TABLE, SEQUENCE, and MAP parameters permit the use of wildcards to specify multiple objects in one statement. An asterisk (*) matches any number of characters.

By default, wildcarding is resolved in the following manner:

Source objects: If the name of the source object is stated explicitly, the resolution for that object and its target object occurs at process startup. When a source object name is wildcarded, the resolution for that object and its target object occurs when the first row for that source object is processed.

Target objects: When a target object is wildcarded, GoldenGate replaces the wildcard with the name of the corresponding source object. (See "Rules for using wildcards".)

The default behavior enables GoldenGate to capture source objects that are created after processing starts. To change the rules for resolving wildcards, use the WILDCARDRESOLVE parameter. The default is DYNAMIC.

You can combine the use of wildcard object selection with explicit object exclusion by using the TABLEEXCLUDE and MAPEXCLUDE parameters.

Rules for using wildcards

Observe the following rules when using wildcards:

- Use wildcards only for object names. Do not use wildcards for schema or database names.
- Target objects must exist in the target database for source objects that exist at startup, and for source objects that are added after startup.
- For source objects, you can use a partial name with a wildcard. For example, the following source specification is valid:

  ```
  hq.t_*;
  ```

- For target objects, you cannot use a wildcard with a partial name, because the asterisk in a wildcarded target name is replaced with the name of the source object. For example, the following statements would be *incorrect*:

  ```
  TABLE hq.t_*, TARGET rpt.t_*;
  MAP hq.t_*, TARGET rpt.t_*;
  ```

  They would produce the following results when resolved, because the wildcard in the target specification is replaced with T_TEST, the name of a source object:

  ```
  TABLE HQ.T_TEST1, TARGET RPT.T_T_TEST1;
  MAP HQ.T_TEST1, TARGET RPT.T_T_TEST1;
  ```

  The following examples show the correct use of wildcarding.

  ```
  TABLE hq.t_*, TARGET rpt.*;
  MAP hq.t_*, TARGET rpt.*;
  ```

  This enables the following correct result:

  ```
  TABLE HQ.T_TEST1, TARGET RPT.T_TEST1;
  MAP HQ.T_TEST1, TARGET RPT.T_TEST1;
  ```

Default None

Syntax `SEQUENCE <owner>.<sequence>;`

Argument	Description
`SEQUENCE <owner>.<sequence>`	Specifies the owner (schema) and name of the source sequence. See “Object names and owners” on page 282.
`;`	Terminates the SEQUENCE parameter statement. The semi-colon is optional.

Example `SEQUENCE hr.employees_seq;`

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) 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.

Options	Description
GGS_CacheRetryDelay	(SQL Server) 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 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 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 a GoldenGate 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.

To use 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 GoldenGate to treat binary data as a null-terminated string. Contact GoldenGate Technical Support

before using it. NODYNSQL causes Replicat to use literal SQL statements instead of using dynamic SQL with bind variables.

2. From the 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, and DDLGEN

Use the SOURCEDB parameter for databases that require a data source name 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 with SOURCEDB, depending on the authentication that is required by the database:

- For databases that require a database login, 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 c-tree databases, SOURCEDB specifies the server alias.
- For databases that allow authentication at the operating-system level, you can specify SOURCEDB without USERID.

For Manager, use SOURCEDB only when using GoldenGate parameters that cause Manager to interact with the 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>`

Argument	Description
`<data source>`	The name of the data source.

Example 1 `SOURCEDB mydb`

Example 2 `SOURCEDB mydb, USERID ggs, PASSWORD ggs123`

SOURCEDEFS

Valid for Extract data pump and Replicat

Use the SOURCEDEFS parameter to specify the name of a file on the target system or on an intermediary system that contains definitions of the source tables. Source definitions are required when using GoldenGate in a heterogeneous synchronization environment, where source and target table structures are different. To generate the source-definitions file, use the DEFGEN utility. Transfer the file to the intermediary or target system before starting the 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.

Default None

Syntax `SOURCEDEFS <file name>`

Argument	Description
`<file name>`	The fully qualified name of the file containing the source data definitions.

Example 1 `SOURCEDEFS c:\ggs\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:

- A GoldenGate direct load, where Extract sends load data directly to the Replicat process without use of a file.
- A 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 *GoldenGate for Windows and UNIX Administrator 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 140.

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 Extract and Replicat

The implementation of this parameter varies, depending on the process.

SPECIALRUN for Extract

Use the SPECIALRUN parameter in an Extract parameter file to specify a one-time processing run for one of the following configurations:

- A batch change-synchronization run.
- A GoldenGate Rollback session.

For more information about these processes, see the *GoldenGate for Windows and UNIX Administrator Guide*.

SPECIALRUN directs Extract not to create checkpoints. Because a one-time run has a beginning and an end, checkpoints are not required. (For online processing, use the EXTRACT parameter instead of SPECIALRUN.)

When used, SPECIALRUN must be the first parameter in the parameter file. SPECIALRUN requires using the BEGIN and END parameters.

This parameter is not supported for the following databases:

- Ingres
- SQL/MX
- SQL Server
- Sybase

Default None

Syntax `SPECIALRUN, <data source>`

Argument	Description	
`<data source>`	Can be one of the following:	
	`TRANLOG [<bsds>]`	Specifies the transaction log as the data source. Use the <bsds> option for DB2 running on a z/OS system to specify the Bootstrap Data Set 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*. GoldenGate does not perform any validations of the BSDS specification.
	`EXTTRAILSOURCE <trail name>`	Specifies a trail as the data source. Specify the fully qualified name of the trail that was specified with the EXTTRAIL or RMTTRAIL parameter. Use this option with a data pump Extract process.
	`EXTFILESOURCE <file name>`	Specifies an extract file as the data source. Specify the full path name of the file that was specified with the EXTFILE or RMTFILE parameter. Use this option with a data pump Extract process.

Example 1 `SPECIALRUN`

Example 2 `SPECIALRUN TRANLOG`

Example 3 `SPECIALRUN EXTTRAILSOURCE c:\ggs\dirdat\aa`

Example 4 `SPECIALRUN EXTFILESOURCE c:\ggs\dirdat\datafile`

SPECIALRUN 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 the following:

- The initial data load method that loads data from file to Replicat.
- A batch change-synchronization run.

When Replicat is in SPECIALRUN mode, do not start it with the START REPLICAT command in GGSCI. It is either started automatically during the initial load task, or it must be started from the command line in a batch run.

For more information about these configurations, see the *GoldenGate for Windows and UNIX Administrator Guide*.

SPECIALRUN requires the use of the END parameter. Either REPLICAT (see page 265) 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 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 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.

Databases and 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:

DB2 LUW and z/OS

- CHAR
- VARCHAR
- DATE
- All numeric data types.
- BLOB data types

Ingres

All data types except LOB data types.

Oracle

- CHAR
- VARCHAR2
- DATE
- The ANSI equivalents of these types.
- All numeric data types.

SQL Server

- CHAR
- VARCHAR
- DATETIME
- All numeric data types.
- Image and text data types where the length is less than 200 bytes.
- TIMESTAMP parameter types are not supported natively, but you can use other data types as parameters and convert the values to TIMESTAMP format within the stored procedure.

Sybase

All data types except TEXT, IMAGE, and UDT.

Teradata

All Teradata data types that are supported by GoldenGate.

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 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 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 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 GoldenGate, see the *GoldenGate for Windows and UNIX Administrator Guide*.

NOTE For DB2 on z/OS, 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 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. Example: `SQLEXEC “call prc_job_count ()”`
“<sql query>”	Specifies the name of a query to execute. Enclose the query within quotes. 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 defined intervals, for example: `SQLEXEC "call prc_job_count ()" EVERY 30 SECONDS`
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 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 315) 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.
`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.

TABLE for DDLGEN, DEFGEN

Use the TABLE parameter in a DDLGEN or DEFGEN parameter file to identify a source table 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 when using log-based extraction for Oracle. This parameter accepts wildcard (*) arguments for tables only. GoldenGate will automatically increase internal storage to track up to 100,000 wildcard entries.
`DEF <definitions template>`	(DEFGEN only) 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 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

Use the TABLE parameter in an Extract parameter file to specify objects for extraction by GoldenGate. TABLE is valid for:

- Log-based and VAM-based methods of extraction, to support the capture of transactional data changes.
- All initial-load extraction methods, 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. You can:

- Select and filter rows of tables
- Map columns of tables
- Transform data
- Specify key columns
- Execute stored procedures and queries
- Define user tokens
- Trim trailing spaces
- Pass a parameter to a user exit

For indexes, triggers, and materialized views, use TABLE only to specify an object for capture.

NOTE GoldenGate supports the replication of the actual data values of Oracle materialized views. GoldenGate supports the replication of Oracle and Teradata DDL for indexes and triggers, but not the content of those objects. See the *GoldenGate for Windows and UNIX Administrator 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>)]
[, KEYCOLS (<column specification>)]
[, SQLEXEC (<SQL specification>)]
[, SQLPREDICATE "WHERE <where clause>"]
[, TOKENS (<token specification>)]
[, TRIMSPACES | NOTRIMSPACES]
[, WHERE (<where clause>)]
;
```

Table 48 Summary of TABLE syntax components

Component	Description
TABLE <table spec>	Specifies the source table. See the guidelines following this table.

Table 48 Summary of TABLE syntax components (continued)

Component	Description
`TARGET <table spec>`	Specifies a target table to which the source table will be mapped. TARGET is required for TABLE statements when Extract must refer to a target definitions file (specified with the TARGETDEFS parameter) to perform conversions, and 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. Replication from a Windows or UNIX system to a NonStop system require these functions to be performed on the source, however.
	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. See the guidelines following this table for naming conventions.
`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.
`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.

Table 48 Summary of TABLE syntax components (continued)

Component	Description
FILTER	Selects records based on a numeric value. FILTER provides more flexibility than WHERE.
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.
WHERE	Selects records based on conditional operators.

Object names and owners

Source and target object names must be fully qualified in GoldenGate parameter files, as in fin.emp.

Case sensitivity

Whether or not c-tree file and path names are case-sensitive depends on the requirements of the c-tree server host operating system.

If a database is case-sensitive, GoldenGate supports the case sensitivity of database names, owner names, object names, column names, and user names. Case-sensitive names must be specified in GoldenGate parameter files exactly as they appear in the database.

If a database is case-insensitive, or if it supports case-sensitivity but is configured to be case-insensitive, GoldenGate converts all names to upper case. The exception is Oracle 11g, where case-sensitive passwords are supported in GoldenGate input that requires passwords.

To preserve case-sensitivity

Case-sensitive names must be specified in GoldenGate parameter files exactly as they appear in the database. Enclose case-sensitive names in double quotes if the other database (the source or target of the case-sensitive objects) is not case-sensitive.

If replicating from a case-insensitive database to a case-sensitive database, the source object names must be entered in the Replicat MAP statements in upper case, to reflect the fact that they were written to the trail as uppercase by Extract.

For example:

```
MAP SALES.CUSTOMER, TARGET "Sales.Account";
```

NOTE Column names enclosed within quotes are treated as literals. For information about how GoldenGate observes case sensitivity for column names, see "Using COLMAP" on page 305.

Supported characters

GoldenGate supports alphanumeric characters in object names and the column names of key columns and non-key columns. GoldenGate also supports the following non-alphanumeric characters in columns that are not being used by GoldenGate as a key.

Table 49 Supported non-alphanumeric characters in object names and non-key column names[1]

Character	Description
~	Tilde
< >	Greater-than and less-than symbols
/	Forward slash
\	Backward slash
!	Exclamation point
@	At symbol
#	Pound symbol
$	Dollar symbol
%	Percent symbol
^	Carot symbol
( )	Open and close parentheses
_	Underscore
-	Dash
+	Plus sign
=	Equal symbol
|	Pipe

Table 49 Supported non-alphanumeric characters in object names and non-key column names[1]

Character	Description
[ ]	Begin and end brackets
{ }	Begin and end curly brackets (braces)

[1] The type of key that is being used by GoldenGate depends on the definition of a given table and whether there are any overrides by means of a KEYCOLS clause. GoldenGate will use a primary key, if available, or a unique key/index (selection is dependent on the database). In the absence of those definitions, all columns of the table are used, but a KEYCOLS clause overrides all existing key types. For columns that are being used by GoldenGate as a key, the characters in the names must be valid for inclusion in a WHERE clause. This list is all-inclusive; a given database platform may or may not support all listed characters.

Non-supported characters

GoldenGate does not support the following characters in object or column names:

Table 50 Non-supported characters in object and column names[1]

Character	Description
&	Ampersand
*	Asterisk
?	Question mark
:	Colon
;	Semi-colon
,	Comma
‘ ’	Single quotes
“ ”	Double quotes
‘	Accent mark (Diacritical mark)
.	Period
	Space

[1] This list is all-inclusive; a given database platform may or may not support all listed characters.

Using wildcards

The TABLE, SEQUENCE, and MAP parameters permit the use of wildcards to specify multiple objects in one statement. An asterisk (*) matches any number of characters.

By default, wildcarding is resolved in the following manner:

Source objects: If the name of the source object is stated explicitly, the resolution for that object and its target object occurs at process startup. When a source object name is wildcarded, the resolution for that object and its target object occurs when the first row for that source object is processed.

Target objects: When a target object is wildcarded, GoldenGate replaces the wildcard with the name of the corresponding source object. (See "Rules for using wildcards".)

The default behavior enables GoldenGate to capture source objects that are created after processing starts. To change the rules for resolving wildcards, use the WILDCARDRESOLVE parameter. The default is DYNAMIC.

You can combine the use of wildcard object selection with explicit object exclusion by using the TABLEEXCLUDE and MAPEXCLUDE parameters.

Rules for using wildcards

Observe the following rules when using wildcards:

- Use wildcards only for object names. Do not use wildcards for schema or database names.
- Target objects must exist in the target database for source objects that exist at startup, and for source objects that are added after startup.
- For source objects, you can use a partial name with a wildcard. For example, the following source specification is valid:

  ```
  hq.t_*;
  ```

- For target objects, you cannot use a wildcard with a partial name, because the asterisk in a wildcarded target name is replaced with the name of the source object. For example, the following statements would be *incorrect*:

  ```
  TABLE hq.t_*, TARGET rpt.t_*;
  MAP hq.t_*, TARGET rpt.t_*;
  ```

  They would produce the following results when resolved, because the wildcard in the target specification is replaced with T_TEST, the name of a source object:

  ```
  TABLE HQ.T_TEST1, TARGET RPT.T_T_TEST1;
  MAP HQ.T_TEST1, TARGET RPT.T_T_TEST1;
  ```

  The following examples show the correct use of wildcarding.

  ```
  TABLE hq.t_*, TARGET rpt.*;
  MAP hq.t_*, TARGET rpt.*;
  ```

  This enables the following correct result:

  ```
  TABLE HQ.T_TEST1, TARGET RPT.T_TEST1;
  MAP HQ.T_TEST1, TARGET RPT.T_TEST1;
  ```

Using Unicode and native encoding in a TABLE statement

GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Windows, UNIX, and Linux operating systems. An escape sequence can be used in the following elements within a TABLE or MAP statement:

- WHERE clause
- COLMAP clause to assign a Unicode character to a Unicode column, or to assign a native-encoded character to a column.
- GoldenGate column conversion functions within a COLMAP clause.

GoldenGate supports the following types of escape sequence:

- \uFFFF Unicode escape sequence
- \377 Octal escape sequence
- \xFF Hexadecimal escape sequence

The following limitations apply:

- This support is limited to UTF-16 code points from U+0000 to U+007F, the equivalent of 7-bit ASCII.
- The source and target columns must both be Unicode.
- The source and target data types must be identical (for example, NCHAR to NCHAR).

To use an escape sequence

Begin each escape sequence with a reverse solidus (code point U+005C), followed by the character code point. (A solidus is more commonly known as the backslash symbol.) Use the escape sequence, instead of the actual character, within your input string in the parameter statement or column-conversion function.

To use the \uFFFF Unicode escape sequence

- Must begin with a lowercase u, followed by exactly four hexadecimal digits.
- Supported ranges:
  - 0 to 9 (U+0030 to U+0039)
  - A to F (U+0041 to U+0046)
  - a to f (U+0061 to U+0066)
- This is the only permissible escape sequence to use for NCHAR and NVARCHAR columns.
- Surrogate pairs are not supported.

Example \u20ac is the Unicode escape sequence for the Euro currency sign.

NOTE For reliable cross-platform support, use the Unicode escape sequence. Octal and hexadecimal escape sequences are not standardized on different operating systems.

To use the \377 octal escape sequence

- Must contain exactly three octal digits.
- Supported ranges:

- Range for first digit is 0 to 3 (U+0030 to U+0033)
- Range for second and third digits is 0 to 7 (U+0030 to U+0037)

Example \200 is the octal escape sequence for the Euro currency sign on Microsoft Windows

To use the \xFF hexadecimal escape sequence

- Must begin with a lowercase x followed by exactly two hexadecimal digits.
- Supported ranges:
  - 0 to 9 (U+0030 to U+0039)
  - A to F (U+0041 to U+0046)
  - a to f (U+0061 to U+0066)

Example \x80 is the hexadecimal escape sequence for the Euro currency sign on Microsoft Windows.

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.

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. 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 289
- ASSUMETARGETDEFS on page 117
- "Creating a data-Definitions file" in the *GoldenGate for Windows and UNIX Administrator 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 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.

For all databases except Sybase and SQL Server, column names are changed to upper case for name comparison. For Sybase and SQL Server, 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	**Target table USER3.SM01**
id	ID
owner	owner
created	id
changed	Creator
creator	comment
modifiedBy	ModifiedBy
comment	creationDate
COMMENT	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 *GoldenGate for Windows and UNIX Administrator 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 "Supported character types" on page 301. For rules that apply when using Unicode or native encoded strings or columns, see page 304. <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 a GoldenGate column-conversion function, such as @STREXT (COL1, 1, 3). For descriptions of GoldenGate column-conversion functions, see Chapter 4.
`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 305 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 GoldenGate.
- Use COLSEXCEPT to exclude columns from synchronization. All other columns are processed by GoldenGate. For tables with numerous columns, COLSEXCEPT may be more efficient than listing each column with COLS. Do *not* exclude key columns.

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, 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 *GoldenGate for Windows and UNIX Administrator 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 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 or stop a process, to perform a transformation, or to report statistics. The event marker system can be put to use for purposes such as:

- To establish a synchronization point at which SQLEXEC or user exit functions can be performed
- To execute a shell command that executes a data validation script
- To activate tracing when a specific account number is detected
- To capture lag history
- To establish a point at which to start batch processes or end-of-day reporting procedures

The event marker feature is supported for the replication of data changes, but not for initial loads.

To use the event marker system

The system requires the following input components:

1. Specify the *event record* that will trigger the action. You can do this by including a FILTER or WHERE clause, or a SQLEXEC query or procedure, in one of the following parameter statements:
   - TABLE statement in an Extract parameter file
   - MAP statement in a Replicat parameter file
   - Special TABLE statement in a Replicat parameter file that enables you to perform EVENTACTIONS actions without mapping a source table to a target table
2. In the same TABLE or MAP statement where you specified the event record, include the EVENTACTIONS parameter with the appropriate option to specify the action that is to be taken by the process.

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.

To combine multiple actions

- Many, but not all EVENTACTIONS options, can be combined. You probably will need to combine two or more actions to achieve your goals.
- The entire EVENTACTIONS statement is parsed first, and only then are the specified options executed according to which one takes precedence over another. In the following list, the actions that are listed before Process the record will occur before the record is written to the trail or applied to the target (depending on the process). Actions that are listed after Process the record will be executed after the record is processed.
  - TRACE
  - LOG

- CHECKPOINT BEFORE
- IGNORE
- DISCARD
- SHELL
- ROLLOVER
- (Process the record)
- REPORT
- ABORT
- CHECKPOINT AFTER
- FORCESTOP
- STOP

To control the processing of the event record itself

To prevent the event record itself from being processed in the normal manner, use the IGNORE or DISCARD option. Because IGNORE and DISCARD are evaluated before the record itself, they prevent the record from being processed. Without those options, Extract writes the record to the trail, and Replicat applies the operation that is contained in the record to the target database.

You should take into account the possibility that a transaction could contain two or more records that trigger an event action. In such a case, there could be multiple executions of certain EVENTACTIONS specifications. For example, encountering two qualifying records that trigger two successive ROLLOVER actions will cause Extract to roll over the trail twice, leaving one of the two essentially empty.

Syntax

```
EVENTACTIONS (
[STOP | ABORT | FORCESTOP]
[IGNORE [TRANSACTION [INCLUDEVENT]]
[DISCARD]
[LOG [INFO | WARNING]]
[REPORT]
[ROLLOVER]
[SHELL <command>]
[TRACE <trace file> [TRANSACTION] [PURGE | APPEND]]
[CHECKPOINT [BEFORE | AFTER | BOTH]]
[, ...]
)
```

Action	Description
STOP	Brings the process to a graceful stop when the specified event record is encountered. The process waits for open transactions to be completed before stopping. If the transaction is a Replicat grouped or batched transaction, the current group of transactions are applied before the process stops gracefully. The process restarts at the next record after the event record, so long as that record also signified the end of a transaction. The process logs a message if it cannot stop immediately because a transaction is still open. However, if the event record is encountered within a long-running open transaction, there is no warning message that alerts you to the uncommitted state of the transaction. Therefore, the process may remain running for a long time despite the STOP event. STOP can be combined with other EVENTACTIONS options except for ABORT and FORCESTOP.
ABORT	Forces the process to exit immediately when the specified event record is encountered, whether or not there are open transactions. The event record is not processed. A fatal error is written to the log, and the event record is written to the discard file if DISCARD is also specified. The process will undergo recovery on startup. ABORT can be combined only with CHECKPOINT BEFORE, DISCARD, SHELL, and REPORT.
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 [TRANSACTION [INCLUDEVENT]]`	By default, forces the process to ignore the specified event record. No warning or message is written to the log, but the GoldenGate statistics are updated to show that the record was ignored. ◆ Use TRANSACTION to ignore the entire transaction that contains the record that triggered the event. If TRANSACTION is used, the event record must be the first one in the transaction. When ignoring a transaction, the event record is also ignored by default. TRANSACTION can be shortened to TRANS. ◆ Use INCLUDEEVENT with TRANSACTION to propagate the event record to the trail or to the target, but ignore the rest of the associated transaction. IGNORE can be combined with all other EVENTACTIONS options except ABORT and DISCARD.
`DISCARD`	Causes the process to: ◆ write the specified event record to the discard file. ◆ update the 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 GoldenGate error log, and to the system event log. Use the following options to specify the severity of the message: ◆ INFO specifies a low-severity informational message. This is the default. ◆ WARNING specifies a high-severity warning message. LOG can be combined with all other EVENTACTIONS options except ABORT. If using ABORT, LOG is not needed because ABORT logs a fatal error before the process exits.
`REPORT`	Causes the process to generate a report file when the specified event record is encountered. This is the same as using the SEND command with the REPORT option in GGSCI. The REPORT message occurs after the event record is processed (unless DISCARD, IGNORE, or ABORT are used), so the report data will include the event record. REPORT can be combined with all other EVENTACTIONS options.

Action	Description
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 specified event record is encountered. ◆ <command> specifies the system or shell command to be issued. ◆ If the shell command is successful, the process writes an informational message to the report file and to the event log. Success is based upon the exit status of the command in accordance with the UNIX shell language. In that language, zero indicates success. ◆ If the system call is not successful, the process abends with a fatal error. In the UNIX shell language, non-zero equals failure. SHELL can be combined with all other EVENTACTIONS options.
TRACE <trace file> [TRANSACTION] [PURGE | APPEND]	Causes process trace information to be written to a trace file when the specified event record is encountered. By default, tracing is enabled until the process terminates. To set the trace level, use the GoldenGate TRACE or TRACE2 parameter. ◆ <trace file> specifies the name of the trace file and must appear immediately after the TRACE keyword. You can specify a unique trace file, or use the default trace file that is specified with the standalone TRACE or TRACE2 parameter. The same trace file can be used across different TABLE or MAP statements in which EVENTACTIONS TRACE is used. If multiple TABLE or MAP statements specify the same trace file name, but the TRACE options are not used consistently, preference is given to the options in the last resolved TABLE or MAP that contains this trace file.

Action	Description
	◆ Use TRANSACTION to enable tracing only until the end of the current transaction, instead of when the process terminates. For Replicat, transaction boundaries are based on the source transaction, not the typical Replicat grouped or batched target transaction. TRANSACTION can be shortened to TRANS. ◆ Use PURGE to truncate the trace file before writing additional trace records, or use APPEND to write new trace records at the end of the existing records. APPEND is the default. TRACE can be combined with all other EVENTACTIONS options except ABORT. To disable tracing to the specified trace file, issue the GGSCI SEND <process> command with the TRACE OFF <filename> option.
`CHECKPOINT [BEFORE | AFTER | BOTH]`	Causes the process to write a checkpoint when the specified event record is encountered. Checkpoint actions provide a context around the processing that is defined in TABLE or MAP statements. This context has a begin point and an end point, thus providing synchronization points for mapping the functions that are performed with SQLEXEC and user exits. ◆ BEFORE BEFORE for an Extract process writes a checkpoint before Extract writes the event record to the trail. BEFORE for a Replicat process writes a checkpoint before Replicat applies the SQL operation that is contained in the record to the target. BEFORE requires the event record to be the first record in a transaction. If it is not the first record, the process will abend. Use BEFORE to ensure that all transactions prior to the one that begins with the event record are committed. CHECKPOINT BEFORE can be combined with all EVENTACTIONS options. ◆ AFTER AFTER for Extract writes a checkpoint after Extract writes the event record to the trail. AFTER for Replicat writes a checkpoint after Replicat applies the SQL operation that is contained in the record to the target. AFTER flags the checkpoint request as an advisory, meaning that the process will only issue a checkpoint at the next practical opportunity. For example, in the case where the event record is one of a multi-record transaction, the checkpoint will take place at the next transaction boundary, in keeping with the GoldenGate data-integrity model. 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 *GoldenGate for Windows and UNIX Administrator 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). FETCHCOLS and FETCHCOLSEXCEPT ensure that 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.

Fetching works as follows:

- For an Oracle 9i or later source database, 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 GoldenGate uses Flashback Query, see the *GoldenGate for Windows and UNIX Administrator Guide*.
- For an Oracle 8.x or earlier source database, GoldenGate fetches the values directly from the database table. Those values might not correspond to the transaction record data, but in fact could be more current if the columns were updated before the fetch.
- Instead of using FETCHCOLS or FETCHCOLSEXCEPT, it may be more efficient to enable supplemental logging for the desired columns.

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. Values that can be present in the transaction log are those of columns that were either modified or included in supplemental logging.

- 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.

Do not use these options for tables being processed in pass-through mode by a data-pump Extract group (using the PASSTHRU parameter in the parameter file). A database login is not supported by that processing mode.

This option is valid for the Oracle database.

Default FETCHMODCOLS is the default only for LOBs, user-defined types, nested tables, and XMLType. Other columns must be explicitly defined in the column specification.

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, GoldenGate column-conversion functions, or both.

NOTE To filter based on a string, use one of the 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))`
	Text strings within a FILTER clause must be enclosed within double quotes, as in `FILTER (@STRFIND(NAME, "JOE") > 0).`
	You can use GoldenGate's column-conversion functions within a filter clause, as in:
	`FILTER (@COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)`

`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.

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 log-based extraction only).

On the target tables, create a unique index on the KEYCOLS-defined key columns. An index improves the speed with which 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.

Using SQLEXEC

Use SQLEXEC to execute a SQL stored procedure or query from within a TABLE statement

during GoldenGate processing. SQLEXEC enables 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.

SQLEXEC dependencies and restrictions

- The SQL is executed by the user under which the 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 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 319.
- For DB2 on z/OS, 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 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 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.

Databaes and 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:

DB2 LUW and z/OS

- CHAR
- VARCHAR
- DATE
- All numeric data types.
- BLOB data types

Ingres

All data types except LOB data types.

Oracle

- CHAR
- VARCHAR2
- DATE
- The ANSI equivalents of these types.
- All numeric data types.

SQL Server

- CHAR
- VARCHAR
- DATETIME
- All numeric data types.
- Image and text data types where the length is less than 200 bytes.
- TIMESTAMP parameter types are not supported natively, but you can use other data types as parameters and convert the values to TIMESTAMP format within the stored procedure.

Sybase

All data types except TEXT, IMAGE, and UDT.

Teradata

All Teradata data types that are supported by GoldenGate.

For additional instructions for using stored procedures and queries with GoldenGate, see the *GoldenGate for Windows and UNIX Administrator 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. Not required when executing a procedure only 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.

Component	Description
<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 325.

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.
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. 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.

Component	Description
<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 325.

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

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 330).
- 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 a 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 398.

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, for example:

```
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"

Column name	Description
custid	Number
current_residence_state	Char(2)
birth_state	Char(2)

Target table "cust_extended"

Column name	Description
custid	Number
current_residence_state_long	Varchar(30)
birth_state_long	Varchar(30)

Lookup table "state_lookup"

Column name	Description
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.
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. 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 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 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. GoldenGate continues processing after reporting the error.
RAISE	Handles errors according to rules set by a REPERROR parameter. 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 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 a 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.
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 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. 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 GoldenGate column-conversion function that is supplying the input. Either a PARAMS clause or NOPARAMS is required.

The following are the databases that are supported by SQLEXEC and the data types that are supported for input and output parameters:

DB2 LUW and z/OS

- CHAR
- VARCHAR
- DATE
- All numeric data types.
- BLOB data types

Ingres

All data types except LOB data types.

Oracle

- CHAR
- VARCHAR2
- DATE
- The ANSI equivalents of these types.
- All numeric data types.

SQL Server

- CHAR
- VARCHAR
- DATETIME
- All numeric data types.
- Image and text data types where the length is less than 200 bytes.
- TIMESTAMP parameter types are not supported natively, but you can use other data types as parameters and convert the values to TIMESTAMP format within the stored procedure.

Sybase

All data types except TEXT, IMAGE, and UDT.

Teradata

All Teradata data types that are supported by GoldenGate.

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 a 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, 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 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 *GoldenGate for Windows and UNIX Administrator 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 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.

For more information about using tokens, see the *GoldenGate for Windows and UNIX Administrator 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 alphanumeric characters and is not case-sensitive.
`<token data>`	A character string of up to 2000 bytes. The data can be either a constant that is enclosed within double quotes or the result of a GoldenGate column-conversion function.

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 are truncated. These parameters only affect CHAR to VARCHAR column mappings. The default is TRIMSPACES.

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.

Syntax

```
TABLE <table spec>, {TRIMSPACES | NOTRIMSPACES};
```

Example The following keeps the default of trimming trailing spaces for the first two tables, but trims spaces for the last two.

```
TABLE fin.src1;
TABLE fin.src2;
TABLE fin.src3, NOTRIMSPACES;
TABLE fin.src4, NOTRIMSPACES;
```

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. To use a Unicode column or a string that contains extended ASCII or unprintable characters, see page 304. For full instructions on using a WHERE clause, see the *GoldenGate for Windows and UNIX Administrator 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 51 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 GoldenGate.
Comparison operators	`=, <>, >, <, >=, <=`
Conjunctive operators	`AND, OR`
Grouping parentheses	Use open and close parentheses for logical grouping of multiple elements.

Example The following WHERE clauses demonstrate the use of a Unicode escape sequence in a WHERE clause that can be part of a TABLE or MAP statement. Each escapes the Latin lowercase e with an acute (code point U+00E9) in a different location within the input strings of "Élise," "Zoé," and "Véronique."

```
WHERE (FIRSTNAME <> "\u00e9lise")
WHERE (FIRSTNAME <> "Zo\u00e9")
WHERE (FIRSTNAME <> "V\u00e9ronique")
```

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.

Syntax

```
TABLE <table spec>,
[, SQLEXEC (<SQL specification>), BEFOREFILTER]
[, FILTER (<filter specification>)]
[, WHERE (<where clause>)]
{, EVENTACTIONS ({IGNORE | DISCARD} [<action>])}
;
```

For supported characters in table names and for descriptions of the syntax options shown here, see the MAP parameter documentation.

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 *GoldenGate for Windows and UNIX Administrator 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> [NORENAME]`

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.
`NORENAME`	(Oracle and Teradata when DDL support is enabled) Specifies that the objects in this TABLEEXCLUDE statement will not be renamed to a name that is included in the GoldenGate configuration. An example of how this can happen is: TableA is excluded, but tableB is not, then tableA gets renamed to tableB. NORENAME generates a warning if an object that is not in the configuration is renamed to a name that is included in the configuration. This is most likely to occur when wildcards are being used in TABLE statements. NORENAME is helpful, for example, to prevent errors when the renamed object has a structure that is not supported by GoldenGate, or is an object that you do not want to be replicated. For Oracle RAC, NORENAME also eliminates unnecessary processing that otherwise is required to keep track of excluded objects in case their status changes. Using NORENAME also prevents errors if renamed objects that are now included have unsupported structures. An alternative to using NORENAME is to use DDLOPTIONS with the NOCROSSRENAME option. NOCROSSRENAME applies the processing exclusions globally across all TABLE, SEQUENCE, and TABLEEXCLUDE statements in the parameter file. See page 150.

Example In the following example, the TABLE statement retrieves 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 c-tree databases, TARGETDB specifies the server alias.
- For databases that allow authentication at the operating-system level, you can specify TARGETDB without USERID.

Default None

Syntax `TARGETDB <data source>`

Argument	Description
`<data source>`	The name of the data source.

Example 1 `TARGETDB mydb`

Example 2 `TARGETDB mydb, USERID ggs, PASSWORD ggs123`

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 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 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 Extract operates in an Oracle RAC (Real Application Cluster) environment.

Using performance options

GoldenGate queues data in memory before sending it to the target system. 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>]
[TRACE LEVEL <level>]
[TRACE TIMESTAMPS FORMAT]
```

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 Extract thread. Valid values are 16 to 65535. The default is 255. The default should be adequate in most cases. INQUEUESIZE should be at least twice as large as OUTQUEUESIZE.
`IOLATENCY <milliseconds>`	Specifies the amount of time between the database-configured max commit propagation delay and the internal value used by 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>`	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 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 (9 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).

Argument	Description
OUTQUEUESIZE <n>	Specifies the number of queue entries in the output queue of each Extract thread. Valid values are 8 to 65535. The default is 127. The default should be adequate in most cases. OUTQUEUESIZE should be about half of INQUEUESIZE.
TRACE LEVEL <level>	Specifies the level of tracing in the report file. Use a number from 0 through 3, with 3 being more verbose and producing more messages in the report.
TRACE TIMESTAMPS FORMAT	Formats timestamps as date and time strings. More CPU resources are used this way than formatting as a number, which can have a significant effect on GoldenGate performance if tracing is enabled and Extract activity is high. Without TIMESTAMPS FORMAT, timestamps in trace records are printed as big numbers.

TLTRACE

Valid for Extract

Use the TLTRACE parameter to trace the activity of the database transaction log. Two levels of tracing can show either basic or detailed information about DML and DDL operations being processed.

Use this parameter only with guidance from a GoldenGate Technical Support engineer.

Default No tracing

Syntax

```
TLTRACE
[, LEVEL 1 | LEVEL 2]
[, DATA]
[, DEBUG]
[, PAUSE LEVEL 1 | PAUSE LEVEL 2]
[, DDL[INCLUDE] | DDLONLY]
[, [FILE] <file name>]
[, SIZELIMIT <size>]
[, SHOWCHECKPOINTONLY]
```

Options	Description
LEVEL 1 | LEVEL 2	Defines the level of trace detail. ◆ LEVEL 1 logs information about the records being processed. This is the default for TLTRACE. ◆ LEVEL 2 provides more detailed information about the records being processed.
DATA	Valid for Oracle. Displays transaction log data in raw output format.

Options	Description
DEBUG	Valid for Oracle. Logs detailed trace information to isolate and identify problems with Extract. Note: In certain instances, the DEBUG option causes GoldenGate to pause even when the PAUSE option was not specified.
PAUSE LEVEL 1 | PAUSE LEVEL 2	Valid for Oracle. Causes Extract to pause at predefined points. Valid values: ◆ PAUSE LEVEL 1 causes Extract to pause after processing important transaction log records. ◆ PAUSE LEVEL 2 causes Extract to pause after processing each transaction log record.
DDL[INCLUDE] | DDLONLY	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 fully qualified name of a file to which 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:
	`TLTRACE FILE <file name> DDLINCLUDE`
	If no other options will follow the file name, the FILE keyword can be omitted, for example:
	`TLTRACE DDLINCLUDE <file name>`

SIZELIMIT <size>	Controls the size of the trace file, in bytes. Valid values are from 10240 bytes (10K) to the maximum size allowed by the operating system. The actual file size will always be about 4k higher than the limit to allow for additional configuration information.
SHOWCHECKPOINTONLY	Constrains the trace to the checkpoints that are issued to the log.

Example 1 `TLTRACE LEVEL 1`

Example 2 `TLTRACE LEVEL 1, DEBUG, SIZELIMIT 500000`

Example 3 `TLTRACE LEVEL 1, PAUSE LEVEL 2`

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 GoldenGate Technical Support for assistance if the trace reveals significant processing bottlenecks.

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 fully qualified name of a file to which 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 187.

For instructions on configuring bidirectional synchronization, see the *GoldenGate for Windows and UNIX Administrator 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

TRANLOGOPTIONS

Valid for Extract

Use the TRANLOGOPTIONS parameter to control aspects of the way that Extract interacts with the transaction logs. 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.

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])]
[ALTONLINELOGS (<path> [, <path>] [...])]
[ARCHIVEDLOGONLY]
[ASMBUFSIZE <size>]
[ASMUSER SYS@<ASM_instance>, ASMPASSWORD <password>
      [, ENCRYPTKEY DEFAULT | ENCRYPTKEY <keyname>]]
[BUFSIZE <size>]
[CONVERTUCS2CLOBS]
[EXCLUDETRANS <trans name>]
[EXCLUDEUSER <user name>]
[EXCLUDEUSERID <Oracle uid>]
[FILTERTABLE <table_name>]
[IGNOREDATACAPTURECHANGES | NOIGNOREDATACAPTURECHANGES]
[LOGSOURCE <platform>]
[MANAGESECONDARYTRUNCATIONPOINT | NOMANAGESECONDARYTRUNCATIONPOINT]
[MAXWARNEOF <seconds>]
[NOFLUSH]
[PATHMAP <NFS mount point> <log path>]
[PURGEORPHANEDTRANSACTIONS | NOPURGEORPHANEDTRANSACTIONS]
[QUERYRETRYCOUNT <number of retries>] |
[READBUFFER <byte length>]
[READTIMEOUT <milliseconds>]
[REQUIRELONGDATACAPTURECHANGES | NOREQUIRELONGDATACAPTURECHANGES]
[TRANSCLEANUPFREQUENCY <minutes>]
[UNPRIVILEGED]
}
[, ...]
```

Option	Description

ALTARCHIVEDLOGFORMAT
<string>
[INSTANCE <instance_name>]
[THREADID <id>]
	(Oracle) 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_%S.arc`
	When using ALTARCHIVEDLOGFORMAT on RAC, also use the ALTARCHIVEDLOGFORMAT parameter on each node. 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`

ALTARCHIVELOGDEST [PRIMARY] [INSTANCE <instance_name>] <path name>	(Oracle) Points Extract to the archived or backup transaction logs when they reside somewhere other than the default location. Extract first checks the specified location and then checks the default location. `ALTARCHIVELOGDEST [PRIMARY] [INSTANCE <instance_name>] <path name>` ◆ `<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 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.

Option	Description
	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.
`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. 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, because that would allow 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.

Option	Description

`ALTONLINELOGS (`
`<path>`
`[, <path>]`
`[...]`
`)`
	(SQL Server) Specifies alternate location(s) for SQL Server online transaction logs. Multiple online log files can be specified by separating each one with a comma. If a path contains spaces, enclose it in quote marks, as in:
	ALTONLINELOGS ("c:\SQL Server\Data\NorthWnd1.ldf")
	Enclose the entire input specification within parentheses.

`ARCHIVEDLOGONLY`	(Oracle) 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 more information, see the GoldenGate for Oracle installation 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) Controls the size of the buffer that contains data extracted from an ASM instance. 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, for BUFSIZE. *Oracle*: ◆ Minimum: size of one block in the redo log ◆ Maximum: 28672 ◆ Default: 28672

`ASMUSER SYS@<ASM_instance>,`
`ASMPASSWORD <password>`
`[, ENCRYPTKEY DEFAULT |`
`ENCRYPTKEY <keyname>]`
	(Oracle) Specifies the user and password for logging into an ASM instance. This user must be SYS. If the password is case-sensitive, type it that way.
	◆ ENCRYPTKEY DEFAULT is required if the password was encrypted with a default GoldenGate key by means of the ENCRYPT PASSWORD command without optional arguments.
	◆ ENCRYPTKEY <keyname> is required if the password was encrypted with a user-defined encryption key by means of the ENCRYPT PASSWORD ENCRYPTKEY <keyname> command. Specify the same logical key name that is in the ENCKEYS file.

Option	Description
	Note: This parameter does *not* replace the standard USERID parameter. Both are required in an ASM environment. See the *GoldenGate for Windows and UNIX Administrator Guide* for additional ASM requirements and more information about GoldenGate security features.
`BUFSIZE <size>`	(DB2 LUW, Oracle) Controls the size, in bytes, of the internal buffer that holds the results of each read of the transaction log. The API will return as many records as will fit in the buffer, so use this option with caution. The following are the valid ranges and default sizes, in bytes, for BUFSIZE. *Oracle*: Minimum: 8,192 Maximum: 10,000,000 Default: 1,024,000 *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.
`CONVERTUCS2CLOBS`	(Oracle) Directs Extract to use the byte length instead of the character length when fetching CLOB data from a multi-byte database. The default is to use the character length. CONVERTUCS2CLOBS affects CLOBs stored in-row and applies globally, regardless of whether different columns have different character sets.
`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 *GoldenGate for Windows and UNIX Administrator Guide*.

Option	Description
`EXCLUDEUSER <user name>`	(DB2 LUW, DB2 z/OS, Ingres, Oracle 10g and later, 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. **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. 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 167), 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. For Oracle 9i and earlier, you must use a trace table to identify Replicat transactions. See page 343.

Option	Description
`EXCLUDEUSERID <Oracle uid>`	(Oracle 10g and later) 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. 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.
`FILTERTABLE <table_name>`	(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 *GoldenGate for Windows and UNIX Administrator Guide*
`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.
`IGNOREMISSINGTABLES`	(Extract for DB2 LUW) Causes Extract to ignore tables that are missing because of tables that were deleted after it received DML operations. This can occur in the following condition. Because the DB2 LUW log record does not contain the table name, Extract uses the table space ID and table ID to look up the name from the database in order to process TABLE and TABLEEXCLUDE statements. If the table gets deleted after the log record that is being processed occurs, Extract will skip the record for the missing table. IGNOREMISSINGTABLES is the default.

Option	Description
`LOGSOURCE <platform>`	(Oracle) Specifies the operating system 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 You can also use LOGSOURCE with the 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.

Option	Description
MANAGESECONDARYTRUNCATIONPOINT | NOMANAGESECONDARYTRUNCATIONPOINT	(SQL Server 2005 and Sybase) Controls whether or not GoldenGate maintains the secondary truncation point.The default is MANAGESECONDARYTRUNCATIONPOINT. **SQL Server 2005** Use MANAGESECONDARYTRUNCATIONPOINT if GoldenGate will not be running concurrently with SQL Server replication. GoldenGate will maintain the secondary truncation point.Use NOMANAGESECONDARYTRUNCATIONPOINT if GoldenGate will be running concurrently with SQL Server replication. Allows SQL Server replication to manage the secondary truncation point. **Sybase** Use MANAGESECONDARYTRUNCATIONPOINT if GoldenGate will not be running concurrently with Sybase Replication Server. Use NOMANAGESECONDARYTRUNCATIONPOINT to allow the Sybase replication engine to manage the truncation point. Always make certain that the archives are available to Extract until it has finished processing all of the data in them. You can tell which archives are needed by using the INFO EXTRACT command.
MAXWARNEOF <seconds>	(Oracle) 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.
NOFLUSH	DB2 z/OS) Inhibits the flushing of log buffers.
PATHMAP <NFS mount point> <log path>	(Oracle) 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 RAC) Enables or disables purging of orphaned transactions that occur when a 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.
READBUFFER <byte length>	(c-tree) Specifies a length, in bytes, for the buffer maintained by c-tree that contains replication log data. This buffer is used by default as a performance feature. When Extract requests data that is in the buffer, a noticeable performance gain can usually be observed. The default buffer size, and the minimum permitted, is 8 KB. The maximum is 64 KB. A value of zero disables the replication log read buffer.
READTIMEOUT <milliseconds>	(c-tree) Specifies a timeout, in milliseconds, that determines how long the c-tree replication engine waits to send a change record to the transaction buffer (controlled by READBUFFER) if one is not present at its current position. If a change record has not arrived before the timeout interval elapses, an error is returned by the database. The timeout specified by this parameter determines how long Extract must wait for the next record when one is not present. The default is 1000 milliseconds. The minimum is 1 millisecond. The maximum is 30000 milliseconds.

Option	Description
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 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. ◆ REQUIRELONGDATACAPTURECHANGES Extract abends with an error. ◆ NOREQUIRELONGDATACAPTURECHANGES Extract issues a warning but continues processing the data record.
TRANSCLEANUPFREQUENCY <minutes>	(Oracle RAC) Specifies an interval, in minutes, after which 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.
UNPRIVILEGED	(DB2 on z/OS) Changes the extended attribute of Extract to *not authorized* by the APF (Authorized Program Facility). This is the same as issuing the system command extattr -a extract. Clearing the APF-authorization extended attribute may be done without special permissions. This option directs Extract to use a pre-version-8.0 API, which does not support Sysplex data sharing environments.

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")
```

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 aborted 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 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 GoldenGate's 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

Option	Description
`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 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 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 batch 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 are truncated when posting to the target table. These parameters only affect CHAR to VARCHAR column mappings.

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.

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;
```

UPDATEDELETES | NOUPDATEDELETES

Valid for Replicat

Use the UPDATEDELETES parameter to convert delete operations to update operations for all MAP statements specified after it in the parameter file. Use NOUPDATEDELETES to turn off UPDATEDELETES.

Default NOUPDATEDELETES

Syntax UPDATEDELETES | NOUPDATEDELETES

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 164.

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
```

USEDATEPREFIX

Valid for Replicat

Use the USEDATEPREFIX parameter for Teradata databases that use a Teradata-ODBC driver version earlier than 3.02.0.02. This parameter prefixes data values for DATE data types with a DATE literal.

When using USEDATEPREFIX, use the USETIMEPREFIX and USETIMESTAMPPREFIX parameters. When using these parameters, you must specify the NODYNSQL parameter (see page 167).

Teradata-ODBC driver 3.02.0.02 and later does *not* require this parameter or NODYNSQL.

Default Disabled

Syntax `USEDATEPREFIX`

USERID

Valid for Manager, Extract, Replicat, DEFGEN, DDLGEN

Use the USERID parameter to specify the type of authentication for a GoldenGate process to use when logging into a database, and to specify password encryption information. For more information about GoldenGate encryption, see the *GoldenGate for Windows and UNIX Administrator Guide*.

NOTE Password encryption is not supported for SQL/MX.

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 permissions that are required for the user that is specified with USERID, see the GoldenGate installation guide for the database.

c-tree

Use USERID with PASSWORD for all GoldenGate processes that connect to a c-tree database. Also use the SOURCEDB or TARGETDB parameter with USERID to specify the c-tree server alias.

DB2 for LUW

Use USERID with PASSWORD for all 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 *DB2 LUW Installation and Setup Guide*.

DB2 for z/OS database

Use USERID with PASSWORD if the user that is assigned to the GoldenGate process does not have the DB2 privileges that are required for the process to function properly.

MySQL

Use USERID with PASSWORD for all GoldenGate processes that connect to a MySQL database.

Oracle

Use USERID for 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.

Ingres Database

- To use an operating system login, USERID is not needed. Just use SOURCEDB or TARGETDB parameter with USERID to specify the ODBC data source.
- To use a database user name and password, use USERID with PASSWORD. Also use the SOURCEDB or TARGETDB parameter with USERID to specify the ODBC data source.

SQL/MX

- For 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 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.

SQL Server

Use USERID with PASSWORD if the ODBC datasource connection that will be used by the 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 GoldenGate processes that connect to a Sybase database.

Teradata

Use USERID with PASSWORD for 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>]}
[ENCRYPTKEY DEFAULT | ENCRYPTKEY <keyname>] [SYSDBA]
```

Argument	Description
/	(Oracle and Ingres) Directs 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 GoldenGate parameter files if application passwords frequently change.

Argument	Description
`<user id>` `[, PASSWORD <password>]`	Directs GoldenGate to use database-level authentication. ◆ <user id> specifies the name of a database user, a schema (SQL/MX) or a SQL*Net connect string (Oracle). ◆ <password> specifies the password for the user, if required by the database. If the password was encrypted by means of the ENCRYPT PASSWORD command, use 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 GoldenGate parameter files.
`ENCRYPTKEY DEFAULT`	Required if the password was encrypted with a default GoldenGate key by means of the ENCRYPT PASSWORD command without optional arguments.
`ENCRYPTKEY <keyname>`	Required if the password was encrypted with a user-defined encryption key by means of the ENCRYPT PASSWORD ENCRYPTKEY <keyname> command. Specify the same logical key name that is in the ENCKEYS file.
`SYSDBA`	(Oracle) Specifies that the user logs in as sysdba.

Example 1 `USERID /`

Example 2 `USERID ggs, PASSWORD ggs123`

Example 3 `USERID ggs@ora1.ora, PASSWORD ggs123`

Example 4 `USERID data1`

Example 5 `USERID ggs, PASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH, ENCRYPTKEY superx128`

Example 6 `USERID ggs, PASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH, ENCRYPTKEY default`

USETHREADS | NOUSETHREADS

Valid for Manager

Use the USETHREADS and NOUSETHREADS parameter to control whether or not Manager uses a thread on Windows systems or a child process on UNIX systems to perform background tasks. These background tasks include restarting processes, purging SOURCEISTABLE or SPECIALRUN tasks, purging trail files, or purging GoldenGate log and history tables. Because some of these tasks may take some time, it is best to separate the processing from the parent process by means of a thread or child process.

USETHREADS is the default except for DB2 on z/OS. For DB2 on z/OS, NOUSETHREADS must be used to purge the log and history tables properly. NOUSETHREADS causes the parent process to perform all maintenance tasks.

Default `USETHREADS`

Syntax `USETHREADS | NOUSETHREADS`

USETIMEPREFIX

Valid for Replicat

Use the USETIMEPREFIX parameter for Teradata databases that use a Teradata-ODBC driver version earlier than 3.02.0.02. This parameter prefixes data values for TIME data types with a TIME literal.

When using USETIMEPREFIX, use the USEDATEPREFIX and USETIMESTAMPPREFIX parameters. When using these parameters, you must specify the NODYNSQL parameter (see page 167).

Teradata-ODBC driver 3.02.0.02 and later does *not* require this parameter or NODYNSQL.

Default Disabled

Syntax `USETIMEPREFIX`

USETIMESTAMPPREFIX

Valid for Replicat

Use the USETIMESTAMPPREFIX parameter for Teradata databases that use a Teradata-ODBC driver version earlier than 3.02.0.02. This parameter prefixes data values for TIMESTAMP data types with a TIMESTAMP literal.

When using USETIMESTAMPPREFIX, use the USETIMEPREFIX and USEDATEPREFIX parameters. When using these parameters, you must specify the NODYNSQL parameter (see page 167).

Teradata-ODBC driver 3.02.0.02 and later does *not* require this parameter or NODYNSQL.

Default Disabled

Syntax `USETIMESTAMPPREFIX`

VAM

Valid for Extract

Use the VAM parameter to specify that a Vendor Access Module is being used to perform data capture functions for the Extract process.

Default None

Syntax `VAM <library name>, PARAMS ("<param>" [, "<param>"] [, ...])`

Argument	Description
`<library name>`	The name of the library that is supplied by the VAM 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 GoldenGate directory. This program or library communicates with the GoldenGate API. Teradata calls this library the *Teradata Access Module (TAM).*

Argument	Description
`PARAMS <param>`	◆ PARAMS is a required keyword. ◆ <param> is any parameter, enclosed within quotes, that is passed to the VAM, for example the name of the VAM initialization file or a runtime parameter.

Teradata PARAMS options

For the Teradata TAM, the following are the required parameters:

```
inifile, <ini file>, callbackLib, extract.exe
```

Where:

- inifile indicates that the next parameter specifies the initialization file.
- <ini file> is the name of the initialization file. Unless the file resides in the same directory where GoldenGate's 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. Use VARWIDTHNCHAR if the database has an NLS_NCHAR_CHARACTERSET value other than AL16UTF16.

VARWIDTHNCHAR works as follows when used for the different GoldenGate processes that support it:

- Extract: Forces Extract to write NCHAR data to the trail with 2-byte length information. By default (NOVARWIDTHNCHAR), length information is not included.
- DEFGEN: Causes DEFGEN to write the NCHAR metadata to the source-definitions file as a variable-length data type (datatype 64). By default (NOVARWIDTHNCHAR), the metadata for NCHAR is written as a fixed-length datatype (datatype 0). Because this is an N* datatype, the sub-datatype remains at 1 (indicating UTF-16 Big Endian).
- Replicat: Causes Replicat to assume that NCHAR data begins with 2-byte length information.

Default NOVARWIDTHNCHAR (Not active; see preceding descriptions)

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 GoldenGate checks for long-running transactions.

When WARNLONGTRANS is specified, GoldenGate checks for transactions that satisfy the specified threshold, and it reports the first one that it finds to the GoldenGate error log, the Extract report file, and the system log. By default, 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 34).

This parameter is valid for Oracle only.

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 GoldenGate checks for transactions that satisfy WARNLONGTRANS and reports them.
	◆ <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.

Argument	Description
NOUSETHREADS	Specifies that the monitoring will be done by the main process thread. By default, it is done with a separate thread. NOUSETHREADS should only be used if the system does not support multi-threading.
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 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, 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 52 Collector parameters

Parameter	Description
`-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 *GoldenGate for Windows and UNIX Administrator Guide*.

Table 52 Collector parameters (continued)

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 *GoldenGate for Windows and UNIX Administrator Guide*.
`-d <definitions file>`	A local file generated by the DEFGEN utility that contains exported definitions of source tables.
`-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 GoldenGate security, see the *GoldenGate for Windows and UNIX Administrator Guide*.
`-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: **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.

Table 52 Collector parameters (continued)

Parameter	Description
	<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.
`-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 GoldenGate's performance.
`-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 *GoldenGate for Windows and UNIX Administrator 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.
`-x <discard file>`	Specifies a discard file for outputting records that could not be formatted. Use the fully qualified name.

Table 52 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. GoldenGate attempts to replicate the data as-is.
-E	Converts incoming header and data to EBCDIC format from ASCII. By default, GoldenGate does not convert the data.	

CHAPTER 4

Column Conversion Functions

Using GoldenGate column conversion functions, you can manipulate source values into the appropriate format for target columns. GoldenGate functions enable you to manipulate numbers and characters, perform tests, extract parameter values, return environment information, and more.

Using Unicode and native encoding in a column conversion function

GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Windows, UNIX, and Linux operating systems. An escape sequence can be used in the following elements within a TABLE or MAP statement:

- WHERE clause
- COLMAP clause to assign a Unicode character to a Unicode column, or to assign a native-encoded character to a column.
- GoldenGate column conversion functions within a COLMAP clause.

GoldenGate supports the following types of escape sequence:

- \uFFFF Unicode escape sequence
- \377 Octal escape sequence
- \xFF Hexadecimal escape sequence

The following limitations apply:

- This support is limited to UTF-16 code points from U+0000 to U+007F, the equivalent of 7-bit ASCII.
- The source and target columns must both be Unicode.
- The source and target data types must be identical (for example, NCHAR to NCHAR).

To use an escape sequence

Begin each escape sequence with a reverse solidus (code point U+005C), followed by the character code point. (A solidus is more commonly known as the backslash symbol.) Use the escape sequence, instead of the actual character, within your input string in the parameter statement or column-conversion function.

To use the \uFFFF Unicode escape sequence

- Must begin with a lowercase u, followed by exactly four hexadecimal digits.
- Supported ranges:
  - 0 to 9 (U+0030 to U+0039)
  - A to F (U+0041 to U+0046)
  - a to f (U+0061 to U+0066)
- This is the only permissible escape sequence to use for NCHAR and NVARCHAR columns.
- Surrogate pairs are not supported.

Example \u20ac is the Unicode escape sequence for the Euro currency sign.

NOTE For reliable cross-platform support, use the Unicode escape sequence. Octal and hexadecimal escape sequences are not standardized on different operating systems.

To use the \377 octal escape sequence

- Must contain exactly three octal digits.
- Supported ranges:
  - Range for first digit is 0 to 3 (U+0030 to U+0033)
  - Range for second and third digits is 0 to 7 (U+0030 to U+0037)

Example \200 is the octal escape sequence for the Euro currency sign on Microsoft Windows

To use the \xFF hexadecimal escape sequence

- Must begin with a lowercase x followed by exactly two hexadecimal digits.
- Supported ranges:
  - 0 to 9 (U+0030 to U+0039)
  - A to F (U+0041 to U+0046)
  - a to f (U+0061 to U+0066)

Example \x80 is the hexadecimal escape sequence for the Euro currency sign on Microsoft Windows.

Example The following example uses a column-conversion function to concatenate the section-sign character (represented by U+00A7) and the Unicode column SECTION.

```
COLMAP ( SECTION = @STRCAT( "\u00a7", SECTION ) );
```

Summary of column-conversion functions

This summary is organized according to the different types of processing that can be performed with the GoldenGate functions. An alphabetical reference begins on page 379.

Table 53 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.

Table 53 Performing tests (continued)

Function	Description
IF	Selects one of two values depending on whether a conditional statement returns TRUE or FALSE.

Table 54 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 55 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 56 Performing arithmetic calculations

Function	Description
COMPUTE	Returns the result of an arithmetic expression.

Table 57 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.

Table 57 Working with strings (continued)

Function	Description
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 58 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, 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. See page 375 for more information.

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 = @IF (PRICE < 0 AND QUANTITY < 0, PRICE * QUANTITY,
@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 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. 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 59 on page 383 for descriptions of date descriptors. The format descriptor must match the date/time/timestamp format for the target. 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 59 for date descriptions.
<source col>	The name of a source column supplying the preceding input.

Table 59 Date Descriptors

Descriptor	Description	Valid for...
CC	Century	Input/Output
YY	Two-digit year	Input/Output
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

Table 59 Date Descriptors (continued)

Descriptor	Description	Valid for...
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.	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)
```

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 2006.

```
YTD = @DATEDIFF ("DD", "2006-01-01", @DATENOW ())
```

Example 2 The following calculates the numerical day of the year. (@DATEDIFF returns 0 for 2006-01-01):

```
todays_day = @COMPUTE (@DATEDIFF ("DD", "2006-01-01", @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 ()
```

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, 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. See page 375 for more information.

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 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 444)

Table 60. 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. f
GoldenGate information types	
`(“GGENVIRONMENT”, “<option>”)`	Returns GoldenGate environment information.
`(“GGFILEHEADER”, “<option>”)`	Returns the format and properties of a GoldenGate trail file, which is stored in the file header.
`(“GGHEADER”, “<option>”)`	Returns GoldenGate record header information.
`(“RECORD”, “<option>”)`	Returns the sequence number and RBA to indicate the location of a record in a GoldenGate trail file.
Database information types	
`(“DBENVIRONMENT”, “<option>”)`	Returns global database environment information.
`(“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 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.

Environment value	Return value
"OPTYPE"	Returns the operation type that was attempted. For a list of GoldenGate operation types, see the appendix section of the *GoldenGate for Windows and UNIX Administrator 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.

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 GGENVIRONMENT information type

Use the GGENVIRONMENT option of @GETENV to return information about the 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.

Environment value	Return value
"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 a GoldenGate trail record. Every data record within the GoldenGate trail contains a header, which describes the transaction environment of the record. For more information on record headers, see the *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 YYY-MM-DD HH:MI:SS.FFFFFF, for example: `2006-01-24 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).

Environment value	Return value
"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 *GoldenGate for Windows and UNIX Administrator Guide*.
"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 *GoldenGate for Windows and UNIX Administrator 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 a 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 a GoldenGate process to determine whether the records are in a format that the current version of GoldenGate supports.

The trail header fields are stored as tokens, where the token format remains the same across all versions of GoldenGate. If a version of 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 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 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 GoldenGate compatibility level of the trail file. The compatibility level of the current 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 GoldenGate version 10.0 or later, which supports file headers that contain file versioning information. ◆ 0 means that the trail file is of a 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 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 473.
"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 473.
"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 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 GoldenGate process thata 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 a GoldenGate extract file, populated with change data) ◆ DS_LOG_TABLE (source was a GoldenGate log table, used for trigger-based extraction) ◆ 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.

Environment value	Return value
"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.
"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 ◆ 2000 Advanced Server
"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

Environment value	Return value
Information about the database that produced the data in the trail file.	
"DBNAME"	Returns the name of the database, for example findb.
"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 INGRES 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 GoldenGate use.
"LASTCOMPLETECSN"	Returns recovery information for internal GoldenGate use.
"LASTCOMPLETEXIDS"	Returns recovery information for internal GoldenGate use.
"LASTCSN"	Returns recovery information for internal GoldenGate use.

Environment value	Return value
"LASTXID"	Returns recovery information for internal GoldenGate use.
"LASTCSNTS"	Returns recovery information for internal GoldenGate use.

Using the RECORD information type

Use the RECORD option of @GETENV to return the location of a record in a 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.

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.
"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 473.
"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.
"RSN"	Returns the record sequence number.
"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 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 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 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 GoldenGate function. Use numeric operators (such as =, > or <) only for numeric comparisons. For character comparisons, use one of GoldenGate's 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 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, if one exists. Otherwise, the primary key columns are used.

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, 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, 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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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.

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.
`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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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, 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. See page 375 for more information.

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 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, a 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 *GoldenGate for Windows and UNIX Administrator 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, 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. See page 375 for more information.

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 Parameters

This chapter describes the GoldenGate user exit functions and their parameters. For more information about using GoldenGate user exits, see the *GoldenGate for Windows and UNIX Administrator Guide*.

Calling a user exit

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 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 GoldenGate process that called it).

For more information and syntax for the CUSEREXIT parameter, see page 135.

About the installed user exit files

About the usrdecs.h file

The usrdecs.h file is the include file for the user exit API. It contains type definitions, return status values, callback function codes, and a number of other definitions. The usrdecs.h file is installed within the GoldenGate directory. Do not modify this file without assistance from GoldenGate Technical Support.

Upgrading your user exits

The usrdecs.h file is versioned to allow backward compatibility with existing user exits when enhancements or upgrades, such as new functions or structural changes, are added to a new GoldenGate release. The version of the usrdecs.h file is printed in the report file at the startup of Replicat or Extract.

To use new user exit functionality, you must recompile your routines to include the new usrdecs file. Routines that do not use new features do not need to be recompiled.

Viewing examples of how to use the user exit functions

GoldenGate installs the following sample user exit files into the UserExitExamples directory of the GoldenGate installation directory:

- exitdemo.c shows how to initialize the user exit, issue callbacks at given exit points, and modify data. The demo is not specific to any database type.
- exitdemo_passthru.c shows how the PASSTHRU option of the CUSEREXIT parameter can be used in an Extract data pump.
- exitdemo_more_recs.c shows an example of how to use the same input record multiple times to generate several target records.
- exitdemo_lob.c shows an example of how to get read access to LOB data.
- exitdemo_pk_befores.c shows how to access the before and after image portions of a primary key update record, as well as the before images of regular updates (non-key updates). It also shows how to get target row values with SQLEXEC in the Replicat parameter file as a means for conflict detection. The resulting fetched values from the target are mapped as the target record when it enters the user exit.

Each directory contains the .c file as well as makefiles and a readme.txt file.

Function summary

Parameter	Description
EXIT_CALL_TYPE	Indicates when, during processing, the routine is called.
EXIT_CALL_RESULT	Provides a response to the routine.
EXIT_PARAMS	Supplies information to the routine.
ERCALLBACK	Implements a callback routine. Callback routines retrieve record and GoldenGate context information, and they modify the contents of data records.

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 61 User exit calls

Call type	Processing point
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_BEGIN_TRANS	Called just before the start of a Replicat transaction.

Table 61 User exit calls (continued)

Call type	Processing point
EXIT_CALL_END_TRANS	Called just after the last record in a Replicat transaction.
EXIT_CALL_CHECKPOINT	Called just before an Extract or Replicat checkpoint is written.
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_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_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 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_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_FATAL_ERROR	Called during Extract or Replicat processing just before GoldenGate terminates after a fatal error.
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 62 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 GoldenGate processes the record buffer that was returned by the user exit.
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.
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.
EXIT_ABEND_VAL	Instructs the caller to terminate immediately.
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.

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 63 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 a GoldenGate callback routine using the Windows API or to identify the calling program when user exits are used with both Extract and Replicat processing.

Table 63 User exit input (continued)

Exit parameter	Description
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 222. 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 135.) 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 420 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 64.

Table 64 Result codes

Code	Description
EXIT_FN_RET_OK	The callback function succeeded.
EXIT_FN_RET_INVALID_COLUMN	A non-existent column was referred to in the function call.
EXIT_FN_RET_COLUMN_NOT_FOUND	The column was not found in a compressed update record.
EXIT_FN_RET_TABLE_NOT_FOUND	An invalid table name was specified.
EXIT_FN_RET_BAD_COLUMN_DATA	Invalid data was encountered when retrieving or setting column data.
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_INVALID_CALLBACK_FNC_CD	An invalid callback function code was passed to the callback routine.
EXIT_FN_RET_NOT_SUPPORTED	This function is not supported for this process.
EXIT_FN_RET_FETCH_ERROR	The record could not be fetched. View the error message to see the reason.
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_TOKEN_NOT_FOUND	The specified user token could not be found in the record.
EXIT_FN_RET_INCOMPLETE_DDL_REC	An internal error occurred when processing the DDL record. The record is probably incomplete.
EXIT_FN_RET_ENV_NOT_FOUND	The specified environment value could not be found in the record.
EXIT_FN_RET_INVALID_COLUMN_TYPE	The routine is trying to manipulate a data type that is not supported by GoldenGate for that purpose.

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 65 Summary of 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.
FETCH_CURRENT_RECORD	Use the FETCH_CURRENT_RECORD function to obtain the record that exists in the target table that has the same key as the source record being processed, and without locking the record.
FETCH_CURRENT_RECORD_WITH_LOCK	Use the FETCH_CURRENT_RECORD_WITH_LOCK function to obtain the record that exists in the target table with the same key as the source record being processed, while locking the record for update at the same time.
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_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.

Table 65 Summary of GoldenGate function codes (continued)

Function code	Description
GET_COLUMN_VALUE_FROM_INDEX	Use the GET_COLUMN_VALUE_FROM_INDEX function to retrieve 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 retrieve the column value from the data record by using the specified column name.
GET_DDL_RECORD_PROPERTIES	Use the GET_DDL_RECORD_PROPERTIES function to retrieve information about a DDL operation.
GET_ENV_VALUE	Use the GET_ENV_VALUE function to return information about the GoldenGate environment.
GET_ERROR_INFO	Use the GET_ERROR_INFO function to retrieve error information associated with a discard record.
GET_GMT_TIMESTAMP	Use the GET_GMT_TIMESTAMP function to retrieve the operation commit timestamp in GMT format.
GET_MARKER_INFO	Use the GET_MARKER_INFO function to retrieve 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 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 retrieve the length of the data record.
GET_RECORD_TYPE	Use the GET_RECORD_TYPE function to retrieve the type of record being processed
GET_STATISTICS	Use the GET_STATISTICS function to retrieve the current processing statistics for the Extract or Replicat process.
GET_TABLE_COLUMN_COUNT	Use the GET_TABLE_COLUMN_COUNT function to retrieve the total number of columns in a table.

Table 65 Summary of GoldenGate function codes (continued)

Function code	Description
GET_TABLE_METADATA	Use the GET_TABLE_METADATA function to retrieve metadata for the table that associated with the record that is being processed.
GET_TABLE_NAME	Use the GET_TABLE_NAME function to retrieve the name of the source or target table associated with the record being processed.
GET_TIMESTAMP	Use the GET_TIMESTAMP function to retrieve 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 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_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.

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 GoldenGate's internal 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 458).

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 451) or SET_RECORD_BUFFER function (see page 470) 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 422) to re-compress the record before returning it to the GoldenGate process.

This function is valid for processing UPDATE operations only. Deletes, inserts and updates appear in the buffer as full record images.

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 451).
`compressed_len`	The length of the compressed record. Use the GET_RECORD_BUFFER (see page 451) or GET_RECORD_LENGTH (see page 454) 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 GGS internal 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
```

FETCH_CURRENT_RECORD

Valid for Replicat, currently supported for c-tree database

Use the FETCH_CURRENT_RECORD function to obtain the record that exists in the target table that has the same key as the source record being processed, and without locking the record. This makes the record buffer available to be read into the user exit by completing a call to one of the following record retrieval functions specified for the target image:

- GET_RECORD_BUFFER
- GET_COLUMN_VALUE_FROM_INDEX
- GET_COLUMN_VALUE_FROM_NAME.

Fetching from the target table is only supported by Replicat, and only if a mapped target buffer is available. Replicat fetches the current record by key.

Syntax

```
#include “usrdecs.h”
short result_code;
error_info_def error_info;
ERCALLBACK (FETCH_CURRENT_RECORD, &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.

Input	Description
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_OK
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_NOT_SUPPORTED
EXIT_FN_RET_FETCH_ERROR
```

FETCH_CURRENT_RECORD_WITH_LOCK

Valid for Extract and Replicat, currently supported for c-tree database

Use the FETCH_CURRENT_RECORD_WITH_LOCK function to obtain the record that exists in the target table with the same key as the source record being processed, while locking the record for update at the same time. This makes the record buffer available to be read into the user exit by completing a call to one of the following record retrieval functions specified for the target image:

- GET_RECORD_BUFFER
- GET_COLUMN_VALUE_FROM_INDEX
- GET_COLUMN_VALUE_FROM_NAME.

Fetching from the target table is only supported by Replicat, and only if a mapped target buffer is available. Replicat fetches the current record by key.

Syntax
```
#include “usrdecs.h”
short result_code;
error_info_def error_info;
ERCALLBACK (FETCH_CURRENT_RECORD_WITH_LOCK, &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_OK
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_NOT_SUPPORTED
EXIT_FN_RET_FETCH_ERROR
```

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_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.

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).

Output	Description
	◆ 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.
gg_data_type	The GoldenGate data type of the column.
gg_sub_data_type	The 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 GoldenGate.

key_column_index	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.
	`CREATE TABLE ABC`
	`(`
	`cust_code VARCHAR2(4),`
	`name VARCHAR2(30),`
	`city VARCHAR2(20),`
	`state CHAR(2),`
	`PRIMARY KEY (city, cust_code)`
	`USING INDEX`
	`);`
	Executing the callback function for each column in the logical column order returns the following:
	Column name — **Key index value returned**
	cust_code — 1
	name — -1
	city — 0
	state — -1
	
	◆ 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.

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.

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.

Input	Description
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 GoldenGate data type of the column.
gg_sub_data_type	The 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 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`
	`);`

Output	Description
	The return is as follows: **Column name** / **Key index value returned** cust_code / 1 name / -1 city / 0 state / -1 ◆ 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.
`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_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.

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.

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. The user exit can specify whether the column value is returned in ASCII or GoldenGate internal 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;
short ascii_or_internal;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
} 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 ascii_or_internal, the column value is null-terminated and the maximum length should equal the maximum length of the column value.
`column_index`	The column index of the column value to be returned.

`ascii_or_internal`	One of the following indicating whether to return the column value in ASCII or GGS internal format.
	`EXIT_FN_ASCII_FORMAT`
	`EXIT_FN_INTERNAL_FORMAT`
	Internal format includes a two-byte null indicator and a two-byte variable data length when applicable.

`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 ascii_or_internal is specified as EXIT_FN_ASCII_FORMAT, the column value is returned as a null-terminated ASCII string; otherwise, the column value is returned in GGS internal 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. The actual length does not include a null terminator when ascii_or_internal is specified as EXIT_FN_ASCII_FORMAT.
`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.

Output	Description
`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 ascii_or_internal was specified as EXIT_FN_ASCII_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. 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. The user exit can specify whether the column value is returned in ASCII or GoldenGate internal format.

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;
short ascii_or_internal;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
} 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 ascii_or_internal) 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.

`ascii_or_internal`	One of the following indicating whether to return the column value in ASCII or GGS internal format.
	`EXIT_FN_ASCII_FORMAT`
	`EXIT_FN_INTERNAL_FORMAT`
	Internal format includes a two-byte null indicator and a two-byte variable data length when applicable.

`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 ascii_or_internal is specified as EXIT_FN_ASCII_FORMAT, the column value is returned as a null-terminated ASCII string; otherwise, the column value is returned in GGS internal 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 ascii_or_internal is specified as EXIT_FN_ASCII_FORMAT.
`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.

Output	Description
`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 ascii_or_internal was specified as EXIT_FN_ASCII_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. 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_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.

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.

Input	Description
ddl_type_max_length	Specifies 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	Specifies 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.
object_max_length	Specifies the maximum length of the name of the object that is returned by *object_name.
owner_max_length	Specifies the maximum length of the name of the owner that is returned by *owner_name.
ddl_text_max_length	Specifies 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 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 386.

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 386.
`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 386. 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.

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. Use of this function requires a re-compile with the Version 2 usrdecs.h.

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, in the format of: YYYY-MM-DD HH:MI:SS.FFFFFF

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.

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.
`program`	A null-terminated string indicating the program that processed the marker.

Output	Description
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 451).

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 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.
`ascii_or_internal`	An indicator for the format in which the column value was passed. Currently, only the GoldenGate internal format is supported, as represented by: EXIT_FN_INTERNAL_FORMAT

Output

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 470) to modify the data retrieved with GET_RECORD_BUFFER. However, it requires an understanding of the data record in GoldenGate internal format. As an alternative, you can set column values in the data record with the SET_COLUMN_VALUE_BY_INDEX function (see page 465) or the SET_COLUMN_VALUE_BY_NAME function (see page 467).

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 424) to decompress the record for possible manipulation and then use the COMPRESS_RECORD function (page 422) 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.
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 internal 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_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.

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.	
`group`	Can be one of the following:	
	`EXIT_STAT_GROUP_STARTUP`	Retrieves statistics since the 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.

Output	Description
`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 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.

Input	Description
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.
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.
num_key_columns	The number of columns in the key that is being used by 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 name of the source or target table associated with the record being processed.

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 null-terminated table name.
`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 the 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.

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.
`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.
`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.

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 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.

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;
short ascii_or_internal;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
} 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. 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.

`ascii_or_internal`	One of the following indicating whether the new column value was passed in ASCII or GGS internal format.
	`EXIT_FN_ASCII_FORMAT`
	`EXIT_FN_INTERNAL_FORMAT`
	Internal format must include a two-byte null indicator and a two-byte variable data length when applicable.

`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.

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;
short ascii_or_internal;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
} 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. 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.

`ascii_or_internal`	One of the following indicating whether the new column value was passed in ASCII or GoldenGate internal format.
	`EXIT_FN_ASCII_FORMAT`
	`EXIT_FN_INTERNAL_FORMAT`
	Internal format must include a two-byte null indicator and a two-byte variable data length when applicable.

`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 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)

Input	Description
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 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.
`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_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.

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.

Output None

Return Values
```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

APPENDIX 1

About the GoldenGate commit sequence number

When working with GoldenGate, you might need to refer to a *Commit Sequence Number*, or CSN. The CSN can be required to position Extract in the transaction log, to reposition Replicat in the trail, or for other purposes. It is returned by some conversion functions and is included in reports and certain GGSCI output.

A CSN is an identifier that GoldenGate constructs to identify a transaction for the purpose of maintaining transactional consistency and data integrity. It uniquely identifies a particular point in time in which a transaction commits to the database.

Each kind of database management system generates some kind of unique serial number of its own at the completion of each transaction, which uniquely identifies that transaction. A CSN captures this same identifying information and represents it internally as a series of bytes, but the CSN is processed in a platform-independent manner. A comparison of any two CSN numbers, each of which is bound to a transaction-commit record in the same log stream, reliably indicates the order in which the two transactions completed.

The CSN value is stored as a token in any trail record that identifies the beginning of a transaction. This value can be retrieved with the @GETENV column conversion function and viewed with the Logdump utility.

Table 66 GoldenGate CSN values per database[1]

Database	CSN Value

Ingres	`<LSN-high>:<LSN-low>`
	Where:
	◆ <LSN-high> is the newest log file in the range. Up to 4 bytes padded with leading zeroes.
	◆ <LSN-low> is the oldest log file in the range. Up to 4 bytes padded with leading zeroes.
	The valid range of a 4-byte integer is 0 to 4294967295.
	The two components together comprise the Ingres LSN.
	Example:
	`1206396546:43927`

Table 66 GoldenGate CSN values per database[1] (continued)

Database	CSN Value

Oracle	`<system change number>`
	Where:
	◆ <system change number> is the Oracle SCN value.
	Example:
	`6488359`

Sybase	`<time_high>.<time_low>.<page>.<row>`
	Where:
	◆ <time_high> and <time_low> represent an instance ID for the log page. It is stored in the header of each database log page. <time_high> is 2-bytes and <time_low> is 4-bytes, each padded with leading zeroes.
	◆ <page> is the database logical page number, padded with zeroes.
	◆ <row> is the row number, padded with zeroes.
	Taken together, these components represent a unique location in the log stream. The valid range of a 2-byte integer for a timestamp-high is 0 - 65535. For a 4-byte integer for a timestamp-low, it is: 0 - 4294967295.
	Example:
	`00001.0000067330.0000013478.00026`

DB2 LUW	`<LSN>`
	Where:
	◆ <LSN> is the variable-length, decimal-based DB2 log sequence number.
	Example:
	`1234567890`

DB2 z/OS	`<RBA>`
	Where:
	◆ <RBA> is the 6-byte relative byte address of the commit record within the transaction log.
	Example:
	`1274565892`

Table 66 GoldenGate CSN values per database[1] (continued)

Database	CSN Value

SQL Server	Can be any of these, depending on how the database returns it:
	◆ Colon separated hex string (8:8:4) padded with leading zeroes and 0X prefix
	◆ Colon separated decimal string (10:10:5) padded with leading zeroes
	◆ Colon separated hex string with 0X prefix and without leading zeroes
	◆ Colon separated decimal string without leading zeroes
	◆ Decimal string
	Where:
	◆ The first value is the virtual log file number, the second is the segment number within the virtual log, and the third is the entry number.
	Examples:
	`0X00000d7e:0000036b:01bd`
	`0000003454:0000000875:00445`
	`0Xd7e:36b:1bd`
	`3454:875:445`
	`3454000000087500445`

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

SQL/MX	`<sequence number>.<RBA>`
	Where:
	◆ <sequence number> is the 6-digit decimal NonStop TMF audit trail sequence number padded with leading zeroes.
	◆ <RBA> is the 10-digit decimal relative byte address within that file, padded with leading zeroes.
	Together these specify the location in the TMF Master Audit Trail (MAT).
	Example:
	`000042.0000068242`

Teradata	`<sequence ID>`
	Where:
	◆ <sequence ID> is a generic VAM fixed-length printable sequence ID.
	Example:
	`0x08000000000000000D700000021`

[1] All database platforms except Oracle, DB2 LUW, and DB2 z/OS have fixed-length CSNs, which are padded with leading zeroes as required to fill the fixed length. CSNs that contain multiple fields can be padded within each field, such as the Sybase CSN.

Glossary

The following explains terminology contained in this manual.

Term	Definition
abend	*Abnormal end*. The failure or unexpected termination of a process running on a computer system.
after image	The values of a row in a database after an insert or update is performed.
alias Extract	An Extract group that operates on a target system that resides within a more secure network zone than the source system. The purpose of the alias Extract is to initiate TCP/IP connections from the target to the less-trusted source. Once a connection is established, data is processed and transferred across the network in the usual manner by a passive Extract group that operates on the source system.
append mode	The default method of writing to the trail, whereby Extract appends re-read data to the trail file after a failure, instead of overwriting the old data.
Archived Log Only mode (ALO)	A mode of operation for Extract, where the process is configured to read exclusively from the archived transaction logs on a production or standby database system.
audit trail	A file on a NonStop Server system that stores modifications made to a database for the purpose of replication and recovery.
batch run	A one-time processing run that has a distinct beginning and an end, as opposed to continuous processing that does not have a specific end point, such as online change synchronization.
before image	The values that exist in a row in a database before a SQL operation is performed on that row.
bidirectional synchronization	Permits load distribution across multiple databases and servers where, in most cases, different users can change the same sets of data and those changes are synchronized by GoldenGate.
BLOB	See *LOB*.
caller	The GoldenGate process that executes a user exit routine.

Term	Definition
cascading synchronization	A GoldenGate configuration in which data is sent from a source system to one or more intermediary systems and, from those systems, to one or more other systems in a synchronized state.
change synchronization	The process of synchronizing data changes made to a database on one system with a similar set of data on one or more other systems.
checkpoints	Internal indicators that record the current read and write position of a GoldenGate process. Checkpoints are used by the Extract and Replicat processes for online change synchronization to ensure data accuracy and fault tolerance.
checkpoint file	A file on disk that stores the checkpoint generated by GoldenGate processes.
checkpoint table	A table created in the target database that maintains Replicat checkpoints, used optionally in conjunction with a standard checkpoint file on disk.
CLOB	See *LOB*.
CMDSEC file	A GoldenGate file that stores rules for GGSCI command permissions.
Collector	The process that receives data from the Extract process over TCP/IP and writes it to a trail or extract file on the target system.
collisions	Errors that occur when data changes that are replicated by GoldenGate are applied to a target table, but the target row is either missing or is a duplicate.
column	One among a set of attributes assigned to an entity that is described by a database table. For example, there can be columns for the name, address, and phone number of the entity called "employees."
column-conversion functions	Built-in GoldenGate processing functions that perform comparisons, tests, calculations, and other processing for the purpose of selecting and manipulating data.
column map	See *map*.
compressed update	A method of logging SQL update operations by which only column values that changed as the result of the update are logged to the transaction log.
conflict resolution	Instructions used in bidirectional synchronization that provide processing and error-handling rules in the event that the same SQL operation is applied to the same row in two or more databases at (or about) the same time.

Term	Definition
consolidated synchronization	The process of replicating different data from two or more databases to one central database, such as in data warehousing.
conversion	See *transformation*.
data definitions file	See *source definitions file* and *target definitions file*.
data pump	A secondary Extract process that reads from an extract file or trail. The trail is populated by a primary Extract process that reads from the data source.
data source	The container of the data changes that are to be processed by GoldenGate. A data source can be: ◆ the transaction log of a database ◆ a Vendor Access Module
data source name (DSN)	A DSN defines an ODBC connection to a database. A DSN consists of a database name, the database directory, the database ODBC driver name, database authentication information, and other information depending on the database. External applications, such as GoldenGate, require a DSN, because a DSN enables an application to connect to a database without having to encode the required information within the application program. The three types of DSN are ◆ A system DSN can be used by any entity that has access to the machine. It is stored within the system configuration. ◆ A user DSN can only be used by a specific user. It is stored within the system configuration. ◆ A file DSN is stored in a text file with a .dsn extension. It can be shared among different systems where the required ODBC driver is installed.
data type	An attribute of a piece of data that identifies what kind of data it is and what kinds of operations can be performed on it. For example, an integer data type is a number, and a character data type contains letters.
DDL	*Data Definition Language*. Data that defines the structure of a database, including rows, columns, tables, indexes, and database specifics such as file locations, users, privileges, and storage parameters.
DDLGEN	A GoldenGate utility that generates table-creation instructions based on the source tables. These instructions are used to create target tables.
DEFGEN	A GoldenGate utility that generates a data definitions file.

Term	Definition
Director	Also known as GoldenGate Director, a graphical user interface that enables users to monitor and manage GoldenGate processes. The components of Director are: **Director Administrator**: A utility used by administrators to define users and instances of GoldenGate. **Director Server**: A software module that gathers data about the GoldenGate processes. **Director Client**: Software installed on a user's system as an interface to Director. **Director Web**: A browser-based user interface to Director (requires no software to be installed).
discard file	A GoldenGate file containing information about SQL operations that failed. This file is created when a record cannot be processed, but only if the DISCARDFILE parameter exists in the parameter file to specify the location for the file.
DML	*Data Manipulation Language*. Retrieves and manipulates data in a database. In the case of SQL, the actions are "select", "insert", "update", and "delete".
DSN	See data source name (DSN).
dynamic Collector	A Collector process that the Manager process starts automatically, as opposed to a static Collector.
EMSCLNT	A GoldenGate utility that distributes GoldenGate system error messages that originate on Windows and other supported operating systems to the EMS (Event Management Subsystem) server on the NonStop Server.
ENCKEYS file	A GoldenGate lookup file that stores encryption keys.
encryption	A method of encoding data into a format that is unreadable to anyone except those who posses a password or decryption code to decipher it.
error log	A file that shows processing events, messages, errors, and warnings generated by GoldenGate. Its name is ggserr.log and it is located in the root GoldenGate directory.
event marker system	A system that customizes GoldenGate to take a specific action during processing based on a record that qualifies for filtering criteria. For example, you can skip the record or stop the GoldenGate process when the record is encountered. See also event record.

Term	Definition
event record	A record in the transaction log that satisfies specific filter criteria and is used to trigger a specific action during processing. See also event marker system.
exceptions map	A special MAP parameter used specifically for error handling, which executes only after an error and sends error data to an exceptions table.
exceptions table	A database table to which information about failed SQL operations is written as the result of an exceptions map. Used for error handling.
Extract	The GoldenGate program that reads data either from a data source, from source tables, or from a local trail or file. Extract processes the data for delivery to the target system. A *primary Extract* reads the data source or database tables, and a *data-pump Extract* reads a local trail that is populated by a primary Extract.
extract file	A file written by GoldenGate where data is stored temporarily awaiting further processing during a batch run or initial load.
extraction	The processing of reading data from database tables or from a data source in preparation for further processing and/or transmission to a target database.
fetch	A query to the database issued by the Extract process when processing a record from the transaction log. A fetch is required if the data values that are needed to complete the SQL operation are not present in the record.
file header	See header.
filtering	The use of rules to select and exclude data for extraction or replication.
function	A segment of code that can be executed within an application or routine. See also *column-conversion functions*.
GGSCI	*GoldenGate Software Command Interface*. The primary interface for issuing commands that configure, control, and monitor GoldenGate.
GLOBALS file	A text file in the root GoldenGate directory that contains parameters which apply to the GoldenGate instance as a whole, as opposed to runtime parameters that are specific to a process such as Extract or Replicat.
GoldenGate Rollback	A utility that uses before images to undo changes made to a database.
group	Also known as *process group*. A group consists of a GoldenGate process (either Extract or Replicat) and the parameter file, the checkpoint file, and any other files associated with that process.

Term	Definition
header	A header can be: ◆ **A record header**: an area at the beginning of a record in a GoldenGate trail file that contains information about the transaction environment for that record. ◆ **A file header**: an area at the beginning of each file in a trail, or at the beginning of an extract file. This header contains information about the file itself, such as the GoldenGate version.
heterogeneous	A data environment where data is being exchanged among different types of applications, different types of databases, or different operating systems, or among a combination of those things.
homogeneous	A data environment where data is being exchanged among identical types of applications, databases, and operating systems.
initial load	The duplication of source data into a target database to make the two databases identical.
intermediary system	A system on the network that serves as a transfer station between the source and target systems. This system can be host to additional processing activities, such as transformation.
key	A column or columns in a table that are being used as a unique identifier for the rows in that table. GoldenGate uses the key to find the correct row in the target database and for fetches from the source database. For GoldenGate, a key can be the primary key, a unique key, a substitute key, or all of the columns of a table in the absence of a defined identifier.
KEYCOLS	A clause in a TABLE or MAP statement that defines a column or columns for GoldenGate to use as a unique identifier to locate any given row in a table.
KEYGEN	A GoldenGate utility that generates encryption keys.
lag	Extract lag is the difference between the time that a record was processed by Extract and the timestamp of that record in the data source. Replicat lag is the difference between the time that the last record in a trail was processed by Replicat and the timestamp of the record in the trail.
latency	The difference in time between when a change is made to source data and when that change is reflected in the target data.

Term	Definition
LOB	*Large Object*. A data type in a database that represents an unstructured object that is too large to fit into a character field, such as a Microsoft Word document or a video or sound file. Subsets of LOB are CLOB (Character Large Object) and BLOB (Binary Large Object), which contain character data and binary data, respectively.
log-based extraction	A method of extracting data changes from the database transaction log.
logical name	A name for a stored procedure that represents an instance of the *execution* of the procedure, as opposed to its actual name. For example, logical names for a procedure named "lookup" might be "lookup1," "lookup2," and so forth.
LUW	*Linux, UNIX, Windows*. An acronym that describes an application that runs on any of these platforms, such as DB2 LUW.
macro	A computer program that automates a task, such as the implementation of parameters and commands.
map	An association between a set of source data and a set of target data. A map can include data selection and conversion criteria. These maps are specified in a Replicat MAP parameter.
MAP statement	A Replicat parameter that specifies the relationship between a source table and a target table and the processing rules for those tables.
Manager	The control program for GoldenGate processing.
marker	A record that is inserted into the audit trail on a NonStop Server to identify application-specific events in the context of Extract and Replicat processing. See also event marker system.
object	For the purpose of this documentation, the term *object* refers to any logical component of a database that is visible to, and can be created by, its users for the purpose of storing data (for example, tables), defining ownership and permissions (for example, roles), executing an action on another object (for example, triggers), and so forth.
object record	A file containing attributes of the tables and other database objects that are configured for processing by GoldenGate, such as column IDs and data types.
ODBC	*Open Database Connectivity*. Acronym for a standard interface that enables applications to connect to different types of databases in a uniform manner. The goal of ODBC is to make the process of connecting to a database independent of programming languages, database systems, and operating systems.
online Extract	An Extract group that is configured for online change synchronization.

Term	Definition
online change synchronization	A GoldenGate processing method in which Extract and Replicat processes run continuously to synchronize data changes unless they are stopped by a GoldenGate user. Online processes maintain checkpoints in the trail.
online processing	See *online change synchronization*.
online Replicat	A Replicat group that is configured for online change synchronization.
operation	A single unit of work. This typically refers to a SQL change made to data or a change made to the structure of an object in the database, but can also refer to any work done by a computer process.
owner	A logical namespace in a database to which database objects are assigned as part of the organizational hierarchy. Because the ownership of database objects is managed differently by different database types, the term *owner* is used in this documentation to denote whichever entity is recognized by the database as the qualifier of an object name, typically a user or schema name. For example, in a qualified Oracle table name of scott.emp, the owner is scott.
overwrite mode	A method of writing data to the trail that was used in GoldenGate versions prior to version 10.0. In this mode, Extract overwrites existing data upon recovery, instead of appending it to the end of the trail file.
parameter	An input or output value for a computer program, such as the code of an application like GoldenGate, a stored procedure, a macro, script, or other processing instructions.
parameter file	A file containing parameters that control the behavior of a GoldenGate process. The default location for parameter files is the dirprm directory in the GoldenGate installation directory.
passive Extract	An Extract process that operates on the source system when an alias Extract is being used on the target. This GoldenGate configuration is required when security rules do not permit TCP/IP connections to be initiated from the source system (as a typical Extract would do) because the target is inside a more secure network zone. The passive Extract is the data pump, when one is being used; otherwise, it is the primary Extract.
pass-through data pump	A data pump that is configured with the PASSTHRU parameter to bypass the need to look up data definitions. This enables faster processing and enables a pump to be used on an intermediary system that has no database.
pass-through Extract	See *pass-through data pump*.

Term	Definition
primary Extract	An Extract group that reads from the data source or directly from the database tables. A primary Extract can write to a local trail, which is then read by a data pump Extract, or it can send the data across TCP/IP to the target system.
primary key	An integrity constraint consisting of a column or columns that uniquely identify all possible rows that exist in a table, current and future. There can be only one primary key for a table. A primary key contains an implicit NOT NULL constraint.
process report	A report generated for Extract, Replicat, and Manager that provides information about the process configuration and runtime statistics and events. The default location for process reports is the dirrpt directory of the GoldenGate installation directory.
record	A unit of information in a transaction log or trail that contains information about a single SQL operation performed on a row in a database. The term *record* is also used to describe the information contained in a specific row of a table.
record header	See header.
remote file	An extract file on a remote system.
remote trail	A trail on a remote system.
Replicat	The GoldenGate process that applies data to target tables or moves it to another application or destination.
report	See *process report*.
replication	The process of recreating source database operations and applying them to a target database.
report file	See *process report*.
rollover	The closing of one file in a sequence of files, such as a trail, and the opening of a new file in the sequence.
routine	A segment of code that is executed within an application such as GoldenGate, which calls functions that retrieve and return values and provide responses. See also *user exit*.
row	Information about a single instance of an entity, such as an employee, that is stored within a database table. For example, a row stores information about "John Doe" in relation to the broader collection of rows that stores information about John and the other employees in a company. Also commonly known as a *record*.

Term	Definition
source	The location of the original data that GoldenGate will be extracting, as in *source database* and *source system*.
source definitions file	A file containing the definitions of the source tables, which is transferred to the target system. This file is used by the Replicat process for data conversion when the source and target tables are dissimilar.
special run	See *batch run*.
statement	An elementary instruction in a computer programming language, for example a SQL statement, parameter statement, or command statement.
static Collector	A Collector process that is started manually by a GoldenGate user, instead of being started automatically by the Manager process.
stored procedure	A group of SQL, PL/SQL, or Java statements that are stored in the database and called on demand by a process or application to enforce business rules, supplement application logic, or perform other work as needed.
substitute key	A unique identifier that consists of any columns in a table that can uniquely identify the rows in that table. A substitute key is not defined in the definition of a table; it is created by creating a KEYCOLS clause in a TABLE or MAP statement.
synchronization	The process of making or keeping two or more sets of data consistent with one another. To be consistent, one set might be identical to the other, or one set might be a reorganized, reformatted, or expanded version of the other, while retaining the essence of the information itself.
table	A logical unit of storage in a database that consists of rows and columns, which together identify the instances of a particular entity (for example, "employees") and the attributes of that entity, such as name, address, and so forth.
TABLE statement	An Extract parameter that specifies a source table or tables whose data is to be extracted from the database.
TAM (Teradata Access Module)	An interface between the Change Data Capture (CDC) component of a Teradata database and the Extract process. It allows GoldenGate to communicate with the Teradata replication components.
target	The destination for the data that is processed by GoldenGate, as in *target database* and *target system*.

Term	Definition
target definitions file	A file containing the definitions of the target tables. This file is transferred to the source system and is used by the Extract process for data conversion when the source and target tables are dissimilar.
task	A special type of batch run in which the Extract process communicates directly with the Replicat process over TCP/IP instead of using a Collector process or trail.
token	A user-defined piece of information that is stored in the header portion of a record in the GoldenGate trail file. Token data can be used to customize the way that GoldenGate delivers information.
trace table	A special table created for use by GoldenGate in an Oracle database. The table is used in conjunction with parameter settings to prevent replicated data from being sent back to the source in a bidirectional synchronization configuration.
trail	A series of files on disk where GoldenGate stores data temporarily in preparation for further processing. GoldenGate records checkpoints in the trail for online change synchronization.
transaction	A group of one or more SQL operations executed within a begin and end statement by a single user on a database.
transaction log	A set of files that records all of the SQL change operations performed on a database for the purpose of data recovery or replication.
transformation	Also called *conversion*. The process of manipulating source data to the format required by target tables or applications, for example converting dates or performing arithmetic calculations. You can do transformation by means of the GoldenGate column-conversion functions.
unidirectional synchronization	A configuration where data changes are replicated in one direction, source-to-target. Changes cannot be made to that same data and then sent back to the source, as is the case in a bidirectional configuration.
unique key	An integrity constraint consisting of a column or columns that uniquely identify all possible rows that exist in a table, current and future. Differs from a primary key in that it does not have an implicit NOT NULL constraint. There can be more than one unique key on a table.
universal data format	A proprietary data format that GoldenGate uses to store data in a trail or extract file. Universal data format allows data to be exchanged rapidly and accurately among heterogeneous databases.
user exit	A user-created program written in C programming code that is called during GoldenGate processing to perform custom processing such as to convert data, to respond to database events, and to repair invalid data.

Term	Definition
VAM (Vendor Access Module)	An API interface that is used by a GoldenGate process module, such as Extract or Replicat, to communicate with certain kinds of databases.
VAM trail	A series of files, similar to a transaction log, that are created automatically and aged as needed. Data operations from concurrent transactions are recorded in time sequence, as they occur, but not necessarily in transaction order. Used to support the Teradata maximum protection commit protocol.
wildcard	A placeholder for an unknown or unspecified character or set of characters. A wildcard is a means of specifying multiple names in a parameter or command statement. GoldenGate supports the asterisk (*) wildcard, which represents any number of unknown characters.

Index

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z