Oracle® GoldenGate

Parameters and Functions Reference Guide

23ai
F70296-07
June 2024

ORACLE

Oracle GoldenGate Parameters and Functions Reference Guide, 23ai

F70296-07

Contents

3 Table and Column Mapping Functions

5 Oracle GoldenGate Programs

Preface

This guide contains reference information, with usage and syntax guidelines, for:

- Oracle GoldenGate configuration parameters.
- Oracle GoldenGate column-conversion functions.
- Oracle GoldenGate user exit functions.
- Oracle GoldenGate parameters, and functions for heterogeneous databases

For details on Admin Client commands, see *Command Line Interface Reference for Oracle GoldenGate*.

Audience

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

Documentation Accessibility

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

Access to Oracle Support

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

Related Information

The Oracle GoldenGate Product Documentation is available from the following location:

Oracle GoldenGate Documentation

Oracle GoldenGate for Distributed Applications and Analytics

Oracle GoldenGate for Distributed Applications and Analytics

For OCI GoldenGate, refer to:

OCI GoldenGate

For details on Oracle Database High Availability, see:

Oracle Database High Availability

Conventions

The following text conventions are used in this document:

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

1

Oracle GoldenGate

Learn about the parameters and functions for Oracle GoldenGate Microservices Architecture.

> **Note:**
>
> To know about the commands and parameters for Oracle GoldenGate for HP Nonstop, see the *Reference Guide for Oracle GoldenGate for HP NonStop (Guardian)*.

2
Oracle GoldenGate Parameters

This chapter contains summaries of the Oracle GoldenGate parameters that control processing, followed by detailed descriptions of each parameter in alphabetical order.
Topics:

Summary of GLOBALS Parameters

Here's a list of GLOBALS parameters.

Table 2-1 GLOBALS Parameters list and description

Parameter	Description
ALLOWINVISIBLEINDEXKEYS	Allows Extract and Replicat to use columns that are part of an Oracle invisible index as a unique row identifier
ALLOWNULLABLEKEYS | NOALLOWNULLABLEKEYS	Changes the key selection logic so that it does not consider a nullable unique key as a viable candidate for uniquely identifying a row.
ALLOWNONVALIDATEDKEYS	Allows Extract, Replicat, and other Oracle GoldenGate commands to use a non-validated primary key or an invalid key as a unique identifier. This parameter overrides the key selection criteria that is used by Oracle GoldenGate
CHECKPOINTTABLE	Specifies the name of a default checkpoint table that can be used by all Replicat groups in one or more Oracle GoldenGate instances.
CRYPTOENGINE	Use this parameter to select which cryptographic library the Oracle GoldenGate processes use to provide implementation of security primitives.
DDLTABLE	Specifies the name of the DDL history table, if other than the default of `GGS_DDL_HIST`.
ENABLEMONITORING	Enables the monitoring of Oracle GoldenGate instances from Oracle GoldenGate Monitor and collects trend data for Performance Metrics Service.
ENABLE_HEARTBEAT_TABLE | DISABLE_HEARTBEAT_TABLE	Specifies whether the Oracle GoldenGate process will be handling records from `GG_HEARTBEAT` table or not.
EXCLUDEWILDCARDOBJECTSONLY	Forces the inclusion of non-wildcarded source objects specified in `TABLE` or `MAP` parameters when an exclusion parameter contains a wildcard that otherwise would exclude that object.
GGSCHEMA	Specifies the name of the schema that contains Oracle GoldenGate database objects.

Table 2-1 (Cont.) GLOBALS Parameters list and description

Parameter	Description
HEARTBEATTABLE	Specifies a non-default name of the heartbeat table. The table name `GG_HEARTBEAT` is the default. This name used to denote the heartbeat table is used to create a seed and history table, `GG_HEARTBEAT_SEED` and `GG_HEARTBEAT_HISTORY` respectively.
LOGOUT_RECV_TIMEOUT	Specifies the amount of time OCI client waits for a response from database server when releasing the connection.
MASTERKEYNAME	Controls the name of the masterkey that Oracle GoldenGate processes in a deployment will use to retrieve the key from the wallet. I
MAXGROUPS	Specifies the maximum number of process groups that can run in an instance of Oracle GoldenGate.
NAMECCSID	Specifies the CCSID (coded character set identifier) of the database object names stored in the SQL catalog tables.
NAMEMATCH parameters	Controls the behavior of fallback name mapping. Fallback name mapping is enabled by default when the source database is casesensitive and the target database support both case-sensitive and case-insensitive object names, such as Oracle and Db2 LUW.
NODUPMSGSUPPRESSION	Prevents the automatic suppression of duplicate informational and warning messages in the report file, the error log, and the system log files.
OUTPUTFILEUMASK	Specifies an octal umask for Oracle GoldenGate processes to use when creating all files.
SESSIONCHARSET	Sets the database session character set for all database connections that are initiated by Oracle GoldenGate processes in the local Oracle GoldenGate instance.
TRAILBYTEORDER	This is automatically handled by OracleGoldenGate.
USEIPV4 | USEIPV6	Forces the use of Internet Protocol version 4 (IPv4) by Oracle GoldenGate for TCP/IP connections.

Summary of Extract Parameters

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

Table 2-2 Extract Parameters: General

Parameter	Description
ABORTDISCARDRECS	Controls the number of discarded records after which Extract aborts.

Table 2-2 (Cont.) Extract Parameters: General

Parameter	Description
TCPSOURCETIMER | NOTCPSOURCETIMER	Adjusts timestamps of records transferred to other systems when those systems reflect different times.
UPDATERECORDFORMAT	Controls whether before and after images are stored in one trail record or two.

Table 2-3 Extract Parameters: Processing Method

Parameter	Description
EXTRACT	Defines an Extract group as an online process.
SOURCEISTABLE	Extracts entire records from source tables.

Table 2-4 Extract Parameters: Selecting, Converting, and Mapping Data

Parameter	Description
COMPRESSDELETES | NOCOMPRESSDELETES	Controls whether Oracle GoldenGate writes only the key or all columns to the trail for delete operations.
COMPRESSUPDATES | NOCOMPRESSUPDATES	Causes only primary key columns and changed columns to be logged for updates.
EXCLUDEHIDDENCOLUMNS	The parameter disables all the Oracle Database hidden columns including the timestamp columns created using automatic CDR.
EXCLUDETAG	Specifies Replicat or data pump changes to be excluded from trail files.
FETCHOPTIONS	Controls certain aspects of the way that Oracle GoldenGate fetches data.
LOGALLSUPCOLS	Logs the columns that are required to support Conflict Detection and Resolution and Integrated Replicat.
SEQUENCE	Specifies sequences for synchronization.
TABLE | MAP	Specifies tables for extraction and controls column mapping and conversion.
TABLEEXCLUDE	Excludes source tables from the extraction process.
TARGETDEFS	Specifies a file containing target table definitions for target databases that reside on the NonStop platform.
TRAILCHARSETASCII	Specifies the ASCII character set for data captured from DB2 on z/OS, when both ASCII and EBCDIC tables are present.

Table 2-4 (Cont.) Extract Parameters: Selecting, Converting, and Mapping Data

Parameter	Description
TRAILCHARSETEBCDIC	Specifies the EBCDIC character set for data captured from DB2 on z/OS, when both ASCII and EBCDIC tables are present.

Table 2-5 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.

Table 2-6 Extract Parameters: Tuning

Parameter	Description
BR	Controls the Bounded Recovery feature of Extract.
CACHEMGR	Controls the virtual memory cache manager.
FLUSHSECS | FLUSHCSECS	Determines the amount of time that record data remains buffered before being written to the trail.
TRANLOGOPTIONS	Supplies capture processing options.
WARNLONGTRANS	Defines a long-running transaction and controls the frequency of checking for and reporting them.

Table 2-7 Extract Parameters: Maintenance

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

Table 2-8 Extract Parameters: Security

Parameter	Description
DECRYPTTRAIL	Required to decrypt data on the target server.
ENCRYPTTRAIL | NOENCRYPTTRAIL	Controls encryption of data in a trail or Extract file.

Summary of Replicat Parameters

Replicat parameters are listed in the following table.

Parameter	Description
TARGETDDLNOTIFY | NOTARGETDDLNOTIFY	Controls whether or not Replicat uses DDL notification to synchronize its target table metadata cache.

ABORTDISCARDRECS

Valid For

Initial Load Extract

Description

Use `ABORTDISCARDRECS` to abort Extracts configured with a `DISCARDFILE` after it has discarded *`N`* number of records.

Default

Zero (0) (Do not abort Extract and any number of discards.)

Syntax

```
ABORTDISCARDRECS
```

ALLOCFILES

Valid For

Extract and Replicat

Description

Use the `ALLOCFILES` parameter to control the incremental number of memory structures that are allocated after the initial memory allocation specified by the `NUMFILES` parameter is reached. 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. The valid range of minimum value is 1

See NUMFILESfor more information.

Default

500

Syntax

```
ALLOCFILES number
```

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

Example

```
ALLOCFILES 1000
```

ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP

Valid For

Extract and Replicat

> **Note:**
>
> Not valid for Oracle Database when running with integrated Replicat or Parallel Replicat.

Description

Use the `ALLOWDUPTARGETMAP` and `NOALLOWDUPTARGETMAP` parameters to control whether or not the following are accepted in a parameter file:

- In an Extract parameter file: duplicate `TABLE` parameters for the same source object if the `COLMAP` option is used in any of them. By default, Extract abends on duplicate `TABLE` statements when `COLMAP` is used.
- In a Replicat parameter file: duplicate `MAP` statements for the same source and target objects. By default, duplicate `MAP` statements cause Replicat to abend.

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

Default

```
NOALLOWDUPTARGETMAP
```

Syntax

```
ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP
```

Examples

Example 1
The following Extract parameter file is permissible with `ALLOWDUPTARGETMAP` enabled.

```
EXTRACT extcust
USERIDALIAS tiger1
ALLOWDUPTARGETMAP
EXTTRAIL dirdat/aa
TABLE ogg.tcustmer;
EXTTRAIL dirdat/bb
TABLE ogg.tcustmer, TARGET ogg.tcustmer, COLMAP (USEDEFAULTS, col1=id, col2=name);
```

Example 2
The following Replicat parameter file is permissible with `ALLOWDUPTARGETMAP` enabled.

```
REPLICAT repcust
USERIDALIAS tiger1
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');
```

Also see About Parallel Replicat.

ALLOWINVISIBLEINDEXKEYS

Valid For

GLOBALS

Description

Use the `ALLOWINVISIBLEINDEXKEYS` parameter in the `GLOBALS` file to allow Extract and Replicat to use columns that are part of an Oracle invisible index as a unique row identifier.

> **Note:**
>
> To enable trigger-based DDL replication to use Oracle invisible indexes, set the following parameter to `TRUE` in the `params.sql script`:
>
> ```
> define allow_invisible_index_keys = 'TRUE'
> ```
>
> This functionality is automatically enabled for integrated capture and Replicat.

Default

None

Syntax

```
ALLOWINVISIBLEINDEXKEYS
```

ALLOWNULLABLEKEYS | NOALLOWNULLABLEKEYS

Valid For

GLOBALS

`ALLOWNULLABLEKEYS` is not valid for integrated Replicat.

Description

Use NOALLOWNULLABLEKEYS to change the key selection logic so that it does not consider a nullable unique key as a viable candidate for uniquely identifying a row. When disabled, the nullable unique keys are viable candidates. The default value for NOALLOWNULLABLEKEYS is set to true.

Allowing Oracle GoldenGate to use a nullable key can cause data corruption, as Oracle treats each row with a NULL value as a key column and as a separate unique value. It is recommended to use NOALLOWNULLABLEKEYS unless you are absolutely sure that the key column does not contain any NULL values.

Be careful when using this parameter because it impacts the contents of the trail file and all installations must be in sync when using this parameter.

Upon upgrade to Oracle GoldenGate 19c, it is recommended that you query DBA_LOGSTDBY_NOT_UNIQUE view. If SCHEMATRANDATA is not being used, then for each table in DBA_LOGSTDBY_NOT_UNIQUE view, add KEYCOLS that mirror key columns returned by INFO TRANDATA, DELETE TRANDATA, or ADD TRANDATA for table to select or use a key with non-NULL columns.

Default

NOALLOWNULLABLEKEYS

Syntax

```
ALLOWNULLABLEKEYS | NOALLOWNULLABLEKEYS
```

ALLOWNONVALIDATEDKEYS

Valid For

GLOBALS

Description

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

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

To enable ALLOWNONVALIDATEDKEYS in a configuration where DDL replication is not active, stop all processes, add ALLOWNONVALIDATEDKEYS to the GLOBALS parameter file, and then restart the processes. To disable ALLOWNONVALIDATEDKEYS again, remove it from the GLOBALS file and then restart the processes.

To enable ALLOWNONVALIDATEDKEYS functionality in a configuration where DDL support is active, take the following steps.

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

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

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

To disable ALLOWNONVALIDATEDKEYS functionality when DDL support is active, take the following steps.

1. Remove ALLOWNONVALIDATEDKEYS from the GLOBALS parameter file.
2. Update the record that you added to the GGS_SETUP table to 0.

```
UPDATE owner.GGS_SETUP SET value='0' WHERE
property='ALLOWNONVALIDATEDKEYS';
COMMIT;
```

Restart all of the Oracle GoldenGate processes.

Default

None (Disabled)

Syntax

```
ALLOWNONVALIDATEDKEYS
```

ALLOWNOOPUPDATES | NOALLOWNOOPUPDATES

Valid For

Replicat

Description

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

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

When NOALLOWNOOPUPDATES is used, Replicat only abends if the source and target tables do not have a key defined, or the Replicat does not use KEYCOLS. In such cases, wherein the target table has no unique key defined and an update operation is carried out on any of the source columns, an error similar to the following occurs:

```
Encountered an update for target table TELLER, which has no unique key defined.
KEYCOLS can be used to define a key. Use ALLOWNOOPUPDATES to process the update
```

without applying it to the target database. Use APPLYNOOPUPDATES to force the update to be applied using all columns in both the SET and WHERE clause.

You can use the parameter APPLYNOOPUPDATES to force the UPDATE to be applied. APPLYNOOPUPDATES overrides ALLOWNOOPUPDATES. If both are specified, Replicat applies updates for which there are key columns for the source and target tables.

If ALLOWNOOPUPDATES is specified when the HANDLECOLLISIONS or INSERTMISSINGUPDATES parameter is being used, and if Oracle GoldenGate has all of the target key values, Oracle GoldenGate applies an UPDATE by using all of the columns of the table in the SET clause and the WHERE clause (invoking APPLYNOOPUPDATES behavior). This is necessary so that Oracle GoldenGate can determine whether the row is present or missing. If it is missing, Oracle GoldenGate converts the UPDATE to an INSERT.

Default

NOALLOWNOOPUPDATES

Syntax

ALLOWNOOPUPDATES | NOALLOWNOOPUPDATES

APPLYNOOPUPDATES | NOAPPLYNOOPUPDATES

Valid For

Replicat

Description

Use APPLYNOOPUPDATES to force a no-op UPDATE operation to be applied by using all of the columns in the SET and WHERE clauses. See ALLOWNOOPUPDATES | NOALLOWNOOPUPDATES for a description of *no-op*.

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

Default

NOAPPLYNOOPUPDATES

Syntax

APPLYNOOPUPDATES | NOAPPLYNOPUPDATES

APPLY_PARALLELISM | MAX_APPLY_PARALLELISM | MIN_APPLY_PARALLELISM

Valid For

Parallel Replicat

Description

Parallel Replicat has two forms of operation. It can run with a constant number of applier threads or it can dynamically adjust the number of appliers based on the transaction mix. These parameters control this behavior. You can adjust this behavior to increase or decrease the number of apply threads and the initial number of connections using the parameters APPLY_PARALLELISM, MAX_APPLY_PARALLELISM, and MIN_APPLY_PARALLELISM.

If these parameters are not set, then the default behavior of Parallel Replicat is to use exactly four appliers to apply the changes to the target database. APPLY_PARALLELISM controls the number of applier processes. If APPLY_PARALLELISM is set, the number of appliers will not dynamically increase or decrease based on transaction mix. If APPLY_PARALLELISM is set, there can be times where the number of concurrent transactions that can be applied by the Parallel Replicat is below the number of applier threads, which can result in idle applier threads.

MAX_APPLY_PARALLELISM and MIN_APPLY_PARALLELISM are used in conjunction when you want to allow the Parallel Replicat process to dynamically adjust the number of applier threads based on the transaction mix. You can set a minimum (MIN_APPLY_PARALLELISM) and maximum (MAX_APPLY_PARALLELISM) number of applier threads to define the ranges in which the Replicat automatically adjusts its parallelism. The initial number of connections will be in the middle of the two parameter values.

APPLY_PARALLELISM is mutually exclusive with MAX_APPLY_PARALLELISM and MIN_APPLY_PARALLELISM. If MAX_APPLY_PARALLELISM and MIN_APPLY_PARALLELISM are used, then do not set APPY_PARALLELSIM. If APPLY_PARALLELISM is set, then do not use MAX_APPLY_PARALLELISM and MIN_APPLY_PARALLELISM.

See Additional Parameter Options for Integrated Replicat

Syntax

```
APPLY_PARALLELISM value
MIN_APPLY_PARALLELISM value
MAX_APPLY_PARALLELISM value
```

Example

```
APPLY_PARALLELISM 4

MIN_APPLY_PARALLELISM 2

MAX_APPLY_PARALLELISM 10
```

ASCIITOEBCDIC

Valid For

Extract and Replicat

Description

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

This parameter must be used in the TRANLOG Extract. It is not valid for Extract data pumps

Default

None

Syntax

```
ASCIITOEBCDIC
```

ASSUMETARGETDEFS

Valid For

Replicat for trail file formats prior to 12c (12.2.0.1)

Description

Use the `ASSUMETARGETDEFS` parameter when the source and target objects specified in a `MAP` statement have identical column structure, such as when synchronizing a hot site. It directs Oracle GoldenGate to assume that the data definitions of the source and target objects are identical, and to refer to the target definitions when metadata is needed for the source data.

When source and target tables have dissimilar structures, do not use `ASSUMETARGETDEFS`. Create a data-definitions file for the source object, and specify the definitions file with the `SOURCEDEFS` parameter. See SOURCEDEFS for more information. Do not use `ASSUMETARGETDEFS` and `SOURCEDEFS` in the same parameter file.

Default

None

Syntax

```
ASSUMETARGETDEFS [OVERRIDE]
```

`OVERRIDE`
By default, the table definitions from the metadata records override the definitions from any `ASSUMETARGETDEFS` file.
Specify `OVERRIDE` to request Replicat to use the definitions from the target database as the definitions for the trail records.

BATCHSQL

Valid For

Replicat

Description

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

`BATCHSQL` is valid for:

- DB2 for i (except V5R4 or i6.1)
- DB2 LUW

- DB2 on z/OS
- Oracle
- PostgreSQL
- SQL Server
- Teradata
- Times Ten

How BATCHSQL Works

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

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

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

Controlling the Number of Cached Statements

The `MAXSQLSTATEMENTS` parameter controls the number of statements that are cached. See "MAXSQLSTATEMENTS" for more information. Old statements are recycled using a least-recently-used algorithm. The batches are executed based on a specified threshold (see "Managing Memory").

Usage Restrictions

SQL statements that are treated as exceptions include:

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

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

- First Replicat tries to use normal mode: one SQL statement at a time within the transaction boundaries that are set with the GROUPTRANSOPS parameter. See "GROUPTRANSOPS" for more information.
- 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.

Table 2-9 Replicat Modes Comparison

Source Transactions *(Assumes same table and column list)*	Replicat Transaction in Normal Mode	Replicat Transaction in BATCHSQL Mode	Replicat Transactions in Source Mode
Transaction 1: INSERT DELETE **Transaction2:** INSERT DELETE **Transaction 3:** INSERT DELETE	INSERT DELETE INSERT DELETE INSERT DELETE	INSERT (x3) DELETE (x3)	**Transaction 1:** INSERT DELETE **Transaction 2:** INSERT DELETE **Transaction 3:** INSERT DELETE

When to Use BATCHSQL

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

Managing Memory

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

```
BATCHESPERQUEUE
BYTESPERQUEUE
OPSPERBATCH
OPSPERQUEUE
```

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

Default

Disabled (Process in normal Replicat mode)

Syntax

```
BATCHSQL
[BATCHERRORMODE | NOBATCHERRORMODE]
[BATCHESPERQUEUE n]
```

```
[BATCHTRANSOPS n]
[BYTESPERQUEUE n]
[OPSPERBATCH n]
[OPSPERQUEUE n]
[THREADS (threadID[, threadID][, ...][, thread_range[, thread_range][, ...])]
[TRACE]
```

BATCHERRORMODE | NOBATCHERRORMODE
Sets the response of Replicat to errors that occur during BATCHSQL processing mode.

BATCHERRORMODE
Causes Replicat to try to resolve errors without leaving BATCHSQL mode. It converts inserts that fail on duplicate-record errors to updates, and it ignores missing-record errors for deletes. When using BATCHERRORMODE, use the HANDLECOLLISIONS parameter to prevent Replicat from abending.

NOBATCHERRORMODE
The default, causes Replicat to disable BATCHSQL processing temporarily when there is an error, and then retry the transaction first in normal mode and then, if normal mode fails, in source mode (same transaction boundaries as on the source).

BATCHESPERQUEUE *n*
Controls the maximum number of batches that one memory queue can contain. After BATCHESPERQUEUE is reached, a target transaction is executed.

- Minimum value is 1.
- Maximum value is 1000.
- Default is 50.

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.

- Minimum value is 1.
- Maximum value is 100000.
- Default is 1000 for nonintegrated Replicat (all database types) and 50 for an integrated Oracle Replicat.

BYTESPERQUEUE *n*
Sets the maximum number of bytes that one queue can contain. After BYTESPERQUEUE is reached, a target transaction is executed.

- Minimum value is 1000000 bytes (1 megabyte).
- Maximum value is 1000000000 bytes (1 gigabyte).
- Default is 2000000 bytes (20 megabytes).

OPSPERBATCH *n*
Sets the maximum number of row operations that one batch can contain. After OPSPERBATCH is reached, a target transaction is executed.

- Minimum value is 1.
- Maximum value is 100000.

- Default is 1200.

OPSPERQUEUE *n*
Sets the maximum number of row operations in all batches that one queue can contain. After OPSPERQUEUE is reached, a target transaction is executed.

- Minimum value is 1.
- Maximum value is 100000.
- Default is 1200.

THREADS (*threadID*[, *threadID*][, ...][, *thread_range*[, *thread_range*][, ...])
Valid for BATCHESPERQUEUE, BATCHTRANSOPS, and BYTESPERQUEUE. Applies these options to the specified thread or threads of a coordinated Replicat.

***threadID*[, *threadID*][, ...]**
Specifies a thread ID or a comma-delimited list of threads in the format of threadID, threadID, threadID.

[, *thread_range*[, *thread_range*][, ...]
Specifies a range of threads in the form of threadIDlow-threadIDhigh or a comma-delimited list of ranges in the format of threadIDlow-threadIDhigh, threadIDlow-threadIDhigh.

A combination of these formats is permitted, such as threadID, threadID, threadIDlow-threadIDhigh.

TRACE
Enables detailed tracing of BATCHSQL activity to the console and to the report file. Do not set tracing without the guidance of an Oracle Support analyst.

NUMTHREADS
Enables detailed tracing of BATCHSQL activity to the console and to the report file. Do not set tracing without the guidance of an Oracle Support analyst.

- Minimum value is 0.
- Maximum value is 50.

MAXTHREADQUEUEDEPTH
Enables detailed tracing of BATCHSQL activity to the console and to the report file. Do not set tracing without the guidance of an Oracle Support analyst.

- Minimum value is 0.
- Maximum value is 50.
- Default is 10.

CHECKUNIQUEKEYS
Enables detailed tracing of BATCHSQL activity to the console and to the report file. Do not set tracing without the guidance of an Oracle Support analyst.

ERRORHANDLING
Enables detailed tracing of BATCHSQL activity to the console and to the report file. Do not set tracing without the guidance of an Oracle Support analyst.

BYPASSCHECK
Enables detailed tracing of `BATCHSQL` activity to the console and to the report file. Do not set tracing without the guidance of an Oracle Support analyst.

Example

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

BEGIN

Valid For

Replicat

Description

You can use the `BEGIN` parameter only with the `SPECIALRUN` parameter. This parameter allows you to direct Replicat to start processing at the first record in the Oracle GoldenGate trail that has a timestamp greater than, or equal to, the time specified with `BEGIN`. All subsequent records, including records where the timestamp is less than the specified time, are processed. Use `BEGIN` when `SPECIALRUN` is specified for the same Replicat group.

Default

None

Syntax

```
BEGIN date[time]
```

date[time]
Specifies a time at which to begin processing. Valid values are a date and optional time in the format of `yyyy-mm-dd[ hh:mi[:ss[.ccccccc]]]` based on a 24-hour clock. Seconds and centiseconds are optional.

Example

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

BLOBMEMORY

This parameter is an alias for `LOBMEMORY`.

BINARY_JSON_FORMAT

Writes out JSON columns in Oracle binary format and GLOBAL parameter. It is supported only with Extract on Oracle Database 21c and higher.

Syntax

BR

Valid For

Extract (Oracle only)

Description

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

Default

```
BR BRINTERVAL 4, BRDIR BR
```

Syntax

```
BR
[, BRDIR directory]
[, BRINTERVAL number {M | H}]
[, BRKEEPSTALEFILES]
[, BROFF]
[, BROFFONFAILURE]
[, BRFSOPTION { MS_SYNC | MS_ASYNC }]
```

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

BRINTERVAL *number* {M | H}
Specifies the time between Bounded Recovery checkpoints. This is known as the *Bounded Recovery interval*. This interval is an integral multiple of the standard Extract checkpoint interval, as controlled by the CHECKPOINTSECS parameter. However, it need not be set exactly. Bounded Recovery will adjust any legal BRINTERVAL parameter internally as it requires.
The minimum interval is 20 minutes. The maximum is 96 hours. The default interval is 4 hours.

> **Note:**
>
> BRINTERVAL should only be changed after consulting with Oracle Support.

BRKEEPSTALEFILES
Causes old Bounded Recovery checkpoint files to be retained. By default, only current checkpoint files are retained. Extract cannot recover from old Bounded Recovery checkpoint files. Retain old files only at the request of an Oracle support analyst.

BROFF
Turns off Bounded Recovery for the run and for recovery. Consult Oracle before using this option. In most circumstances, when there is a problem with Bounded Recovery, it turns itself off.

BROFFONFAILURE
Disables Bounded Recovery after an error. By default, if Extract encounters an error during Bounded Recovery processing, it reverts to normal recovery, but then enables Bounded Recovery again after recovery completes. `BROFFONFAILURE` turns Bounded Recovery off for the runtime processing.

BRRESET
`BRRESET` is a start up option that forces Extract to use normal recovery for the current run, and then turn Bounded Recovery back on after the recovery is complete. Its purpose is for the rare cases when Bounded Recovery does not revert to normal recovery if it encounters an error. Bounded Recovery is enabled during runtime. Consult Oracle Support before using this option.
To use this option, you must start Extract from the command line. To run Extract from the command line, use the following syntax:

```
extract paramfile name.prm reportfile name.rpt
```

Where:

- `paramfile` *`name`*`.prm` is the relative or fully qualified name of the Extract parameter file. The command name can be abbreviated to `pf`.
- `reportfile` *`name`*`.rpt` is the relative or fully qualified name of the Extract report file, if you want it in a place other than the default. The command name can be abbreviated to `rf`.

BR BRFSOPTION {MS_SYNC | MS_ASYNC}
Performs synchronous/asynchronous writes of the mapped data in Bounded Recovery. Consult Oracle Support before using this option.

MS_SYNC
Bounded Recovery writes of mapped data are synchronized for I/O data integrity completion.

MS_ASYNC
Bounded Recovery writes of mapped data are initiated or queued for servicing.

Example

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

CACHEMGR

Valid For

Extract for all databases.

Description

Use the CACHEMGR parameter to specify a non-default file system location for the temporary files needed to hold uncommitted transaction data. The CACHEMGR parameter can also be used to control the amount of virtual memory and temporary disk space that is available for caching uncommitted transaction data. Both of these latter uses are discouraged.

> **Caution:**
>
> Do not change this parameter without consulting Oracle Support. CACHEMGR is internally self-configuring and self-adjusting. It is rare that this parameter requires modification. Doing so unnecessarily may result in performance degradation. It is best to acquire empirical evidence before opening an Oracle Service Request and consulting with Oracle Support.
> However, you can specify the directory for the temporary files without assistance

Oracle GoldenGate only replicates committed transactions. Until a COMMIT is received, any transactional data is stored in an area of virtual memory known as a **cache**. This cache is managed by the CACHEMGR. If the amount of transaction data becomes too great for the virtual memory, then the CACHEMGR writes some of the cached data to temporary files on disk.

Your systems should have sufficient operating system swap and page file space. Oracle recommends a minimum of 512GB.

Identifying the Paging Directory

By default, Oracle GoldenGate maintains the transaction data that it swaps to disk a sub-directory of the Oracle GoldenGate installation directory. CACHEMGR assumes that all of the free space on the file system is available. This directory may fill up quickly if there is a large transaction volume with large transaction sizes. To prevent I/O contention and possible disk-related failures, dedicate a disk to this directory. You can assign directory location with the CACHEDIRECTORY option of the CACHEMGR parameter. A size can also be assigned. However, this is discouraged and should only be done after consulting Oracle Support.

Guidelines for Using CACHEMGR

- This parameter is valid for all databases supported by Oracle GoldenGate.
- 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.

Default

None

Syntax

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

CACHEDIRECTORY *path [size]*
Specifies the name of the directory to which Oracle GoldenGate writes transaction data to disk temporarily when necessary. The default without this parameter is the `dirtmp` sub-directory of the Oracle GoldenGate installation directory. Any directory for temporary files can be on an Oracle Database file system, but cannot be on a direct I/O or concurrent I/O mounted file system that does not support the `mmap()` or `MapViewOfFile()` system calls, like AIX.
On Microservices Architecture (MA), the temporary file location is `$OGG_VAR_HOME/temp`.
You can specify more than one directory by using a `CACHEDIRECTORY` clause for each one. The maximum number of directories is 100.
The value can be specified in bytes or in terms of gigabytes, megabytes, or kilobytes in any of the following forms:
`GB | MB | KB | G | M | K | gb | mb | kb | g | m | k`

CACHESIZE *size*
Sets a soft limit for the amount of virtual memory (**CACHESIZE**) that is available for caching transaction data. You can internally adjust the `CACHESIZE` using `CACHEMGR` as necessary.
If you feel that the default `CACHEMGR` configuration and internal self-adjustment is adversely affecting your system performance, then you should open a Service Request with Oracle Support. It is best to have acquired empirical data showing the problem symptoms in question to aid in configuring a new default.

Example

```
CACHEMGR CACHEDIRECTORY /net/d4atd/ggs/temp
```

CATALOGEXCLUDE

Valid For

Extract, Replicat, DEFGEN

Description

Use the `CATALOGEXCLUDE` parameter to explicitly exclude source objects in the specified container or catalog from the Oracle GoldenGate configuration when the container or catalog name is being specified with a wildcard in `TABLE` or `MAP` statements. This parameter is valid when the database is an Oracle container databasef, where fully qualified three-part names are being used.

The positioning of `CATALOGEXCLUDE` in relation to parameters that specify files or trails determines its effect. Parameters that specify trails or files are: `EXTFILE`, `RMTFILE`, `EXTTRAIL`, `RMTTRAIL`. The parameter works as follows:

- When a `CATALOGEXCLUDE` specification is placed before any `TABLE` or `SEQUENCE` parameters, and also before the parameters that specify trails or files, it applies globally to all trails or files, and to all `TABLE` and `SEQUENCE` parameters.
- When a `CATALOGEXCLUDE` specification is placed after a parameter that specifies a trail or file, it is effective only for that trail or file and only for the `TABLE` or `SEQUENCE` parameters that are associated with it. Multiple trail or file specifications can be made in a parameter file, each followed by a set of `TABLE`, `SEQUENCE`, and `CATALOGEXCLUDE` specifications.

`CATALOGEXCLUDE` is evaluated before evaluating the associated `TABLE` or `SEQUENCE` parameter. Thus, the order in which they appear does not make a difference.

See also the EXCLUDEWILDCARDOBJECTSONLY parameter.

Default

None

Syntax

```
CATALOGEXCLUDE {container}
```

container
The source Oracle container that is to be excluded. A wildcard can be used. Follow the rules for using wildcards described in Using Wildcards in Database Object Names.

Examples

Example 1
This example omits the `pdb1` pluggable database. If integrated Extract is registered with containers `pdb1`, `pdb2` and `pdb3`, you can use the following `CATALOGEXCLUDE` syntax to allow Oracle GoldenGate to skip DML that occurs in the catalog `pdb1`, even though it matches the wildcard syntax.

```
EXTRACT capt
USERIDALIAS ggeast
RMTTRAIL east/aa
CATALOGEXCLUDE pdb1
TABLE *.*.*;
```

CHARMAP

Valid For

Replicat

Description

Use the `CHARMAP` parameter to specify that the character mapping file overrides the character code point mapping.

Default

The encoding of the parameter file is operating system default character set.

Syntax

```
CHARMAP filename
```

The character mapping file format is as follows:

```
-- Sample character mapping file.
--    Can use -- or COMMENT as comment line.
--    Can use CHARSET parameter to specify file encoding.
--
-- Source character set
SOURCECHARSET     shiftjis
--
-- Target character set
TARGETCHARSET     ja16euc
--
-- Character map definition by one code point.
--   left hand is source and right hand target code point.
```

```
       \xa2c1     \x89\xa2\xb7        -- override \xa2c1 to \x89\xa2\xb7
       --
       -- Character map definition by range. Number of source and target characters must
be the same.
      \x61 - \x7a        \x41 - \x5a
```

Example

In the following example, the source and target character sets are different, and a character conversion definition is given using a character mapping file:

```
CHARMAP charmapconv.txt

REPLACEBADCHAR FORCECHECK
```

This enables strict character set conversion and check code point even if the source and target are the same.

Add the following to your character mapping file:

```
SOURCECHARSET AL32UTF8
    TARGETCHARSET UTF-16
    \xef\xbf\xbd \x0020
```

Example

This example uses `CHARMAP` with `REPLACEBADCHAR` to change the target character even when the source and target character sets are the same.

```
REPLACEBADCHAR FORCECHECK

CHARMAP charmapconv.txt
```

This enables strict character set conversion and checks the code point even if the source and target are the same.

By enabling character set conversion for the same character sets, you may encounter some performance degradation.

Add the following to the character mapping file:

```
SOURCECHARSET windows-932
    TARGETCHARSET windows-932
    \x61 - \x7a \x41 - \x5a
```

CHECKPARAMS

Valid For

Extract and Replicat

Description

Use the `CHECKPARAMS` parameter to test the syntax of a parameter file. To start the test:

1. Edit the parameter file to add `CHECKPARAMS`.
2. (Optional) To verify the tables, add the `NODYNAMICRESOLUTION` parameter.
3. Start the process. Without processing data, Oracle GoldenGate audits the syntax. If `NODYNAMICRESOLUTION` exists, Oracle GoldenGate connects to the database to verify that the tables specified with `TABLE` or `MAP` exist. If there is a syntax failure, the process abends

with error 190. If the syntax succeeds, the process stops and writes a message to the report file that the parameters processed successfully.

4. Do one of the following:
   - If the test succeeds, edit the file to remove the `CHECKPARAMS` parameter and the `NODYNAMICRESOLUTION` parameter, if used, and then start the process again to begin processing.
   - If the test fails, edit the parameter file to fix the syntax based on the report's findings, and then remove `NODYNAMICRESOLUTION` and start the process again.

`CHECKPARAMS` can be positioned anywhere within the parameter file.

Default

None

Syntax

```
CHECKPARAMS
```

CHECKPOINTSECS

Valid For

Extract and Replicat

Description

Use the `CHECKPOINTSECS` parameter to control how often Extract and Replicat make their routine checkpoints.

- Decreasing the value causes more frequent checkpoints. This reduces the amount of data that must be reprocessed if the process fails, but it could cause performance degradation because data is written to disk more frequently.
- Increasing the value causes less frequent checkpoints. This might improve performance, but it increases the amount of data that must be reprocessed if the process fails. When using less frequent Extract checkpoints, make certain that the transaction logs remain available in case the data has to be reprocessed.

> **Note:**
>
> In addition to its routine checkpoints, Replicat also makes a checkpoint when it commits a transaction.

Avoid changing `CHECKPOINTSECS` unless you first open an Oracle service request.

Default

10 seconds

Syntax

```
CHECKPOINTSECS seconds
```

seconds
The number of seconds to wait before issuing a checkpoint.

Example

```
CHECKPOINTSECS 20
```

CHECKPOINTTABLE

Valid For

GLOBALS

Description

Use the `CHECKPOINTTABLE` parameter in a `GLOBALS` parameter file to specify the name of a default checkpoint table that can be used by all Replicat groups in one or more Oracle GoldenGate instances. All Replicat groups created with the `ADD REPLICAT` command will default to this table unless it is overridden by using the `CHECKPOINTTABLE` option of that command.

To create the checkpoint table, use the `ADD CHECKPOINTTABLE` command. Oracle supports and recommends that a checkpoint table is created for Integrated Replicat.

See Add a Checkpoint Table for more information about creating a checkpoint table. Also see `ADD CHECKPOINTTABLE`

Default

None

Syntax

```
CHECKPOINTTABLE [container.] owner.table
```

[container.]owner.table
The owner and name of the checkpoint table. Additionally, for an Oracle container database, specify the correct pluggable database (container).

Example

```
CHECKPOINTTABLE finance.ggs.chkpt
```

CHUNK_SIZE

Valid For

Parallel Replicat

Description

`CHUNK_SIZE` defines the memory size for transactions. It controls how large a transaction must be for parallel Replicat to consider it as large. When parallel Replicat encounters a transaction larger than this memory size, it will serialize it, resulting in decreased performance. However, increasing this value will also increase the amount of memory consumed by parallel Replicat.

See About Pararallel Replicat in *Oracle GoldenGate Microservices Documentation*.

Default

1 GB.

You can set the value in bytes (pure number), KB, MB, or GB.

CMDTRACE

Valid For

Extract and Replicat

Description

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]
```

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
```

COLCHARSET

Valid For

Extract, Replicat, and DEFGEN

Description

Use COLCHARSET clause to specify particular column character set or disable character set conversion. This parameter overrides the column character set for the specified column.

The character set specified by the COLCHARSET parameter overrides the character set in the trail file, the character set specified by the SOURCECHARSET OVERRIDE parameter and the character set specified by the CHARSET parameter.

The character set specified by the COLCHARSET Replicat parameter overrides the column level character set specified in the source table definition file.

If the COLCHARSET is specified for DEFGEN file format less than level four, the parameter is ignored and warning message is issued. The column level character set attribute for the older table definition file format is not output.

The COLCHARSET parameter overrides the source column level character set and change the Replicat character set conversion behavior by assuming the source column character set as specified character set.

Default

None

Syntax

```
COLCHARSET character_set (column [, ...])
```

character_set
Any supported character set.

column
The name of a column. To specify multiple columns, create a comma-delimited list.

Examples

Example 1
The following example specifies multiple columns.

```
TABLE SchemaName.TableName, COLCHARSET( WE8MSWIN1252, col0, col2 );
```

Example 2
The following example specifies a different character set.

```
MAP SchemaName.*, TargetName *.*,
  COLCHARSET( WE8MSWIN1252, col1 ),
  COLCHARSET( WE8ISO8859P1, col2 )
```

Example 3
The following example specifies different character set.

```
MAP SchemaName.*, TargetName *.*,
  COLCHARSET( WE8MSWIN1252, col1 ),
  COLCHARSET( WE8ISO8859P1, col2 )
```

Example 4
The following example specifies a wildcard.

```
MAP SchemaName.*, TargetName *.*, COLCHARSET( WE8MSWIN1252, col* )
```

Example 5
The following example disables character set conversion on particular column.

```
MAP SchemaName.*, TargetName *.*, COLCHARSET(PASSTHRU, col )
```

COLMATCH

Valid For

Extract and Replicat

Description

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 Parameters that Control Mapping and Data Integration for more information about mapping columns.

Default

None

Syntax

```
COLMATCH
{NAMES target_column = source_column |
PREFIX prefix |
SUFFIX suffix |
RESET}
```

NAMES *target_column = source_column*
Specifies the name of a target and source column, for example CUSTOMER_CODE and CUST_CODE. If the database requires double quotes to enforce case-sensitivity, specify the column name that way. For example: NAMES "ABC" = "ABC2". For other case-sensitive databases, specify the column name as it is stored in the database, for example: NAMES ABC = abc.

PREFIX *prefix* | SUFFIX *suffix*
Specifies a column name prefix or suffix to ignore. If the database requires double quotes to enforce case-sensitivity, specify the prefix or suffix that way if it is case-sensitive. For other case-sensitive databases, specify the prefix or suffix as it is stored in the database
For example, to map a target column named "ORDER_ID" to a source column named "P_ORDER_ID", specify:

```
COLMATCH PREFIX "P_"
```

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.

Examples

Example 1

```
COLMATCH NAMES "CUSTOMER_CODE" = "CUST_CODE"
```

Example 2

```
COLMATCH NAMES Customer_Code = "Cust_Code"
```

Example 3

```
COLMATCH PREFIX P_
```

Example 4

```
COLMATCH SUFFIX _K
```

Example 5

```
COLMATCH RESET
```

COMPRESSDELETES | NOCOMPRESSDELETES

Valid For

Extract

Description

Use the COMPRESSDELETES and NOCOMPRESSDELETES parameters to control the way that columns are written to the trail record for DELETE operations.

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 support the following databases:

- Oracle
- DB2 LUW
- DB2 z/OS
- DB2 for i
- MySQL
- SQL Server
- PostgreSQL

Default

```
COMPRESSDELETES
```

Syntax

```
{COMPRESSDELETES | NOCOMPRESSDELETES [FETCHMISSINGCOLUMNS]}
```

COMPRESSDELETES
Causes Extract to write only the primary key to the trail for DELETE operations. This is the default. The key provides enough information to delete the correct target record, while restricting the amount of data that must be processed.

NOCOMPRESSDELETES [FETCHMISSINGCOLUMNS]
NOCOMPRESSDELETES sends all the columns to the trail. This becomes the default when a table definition does not include a primary key or unique index, or when a substitute key is defined with the KEYCOLS option of TABLE. The KEYCOLS option writes the specified columns to the trail whether or not a real key exists. See KEYCOLS (*columns*) for more information about the KEYCOLS option.
For SQL Server, columns of IMAGE, NTEXT, and TEXT data types are written as NULL values for delete operations. For more information regarding this restriction, review the Large Object Data Types content at the following Microsoft document:
https://docs.microsoft.com/en-us/sql/relational-databases/system-tables/cdc-capture-instance-ct-transact-sql?view=sql-server-ver15
NOCOMPRESSDELETES is also required when using the Conflict Detection and Resolution (CDR) feature. See Automatic Conflict Detection and Resolution and Manual Conflict Detection and Resolution in the *Oracle GoldenGate Microservices Documentation*.
FETCHMISSINGCOLUMNS is valid for Oracle Database only. It causes the values of data types that are only supported by fetching to be fetched from the database on DELETE operations. These data types are LOB, UDT, LONG, and some XMLType columns. For detailed information about columns that are supported by fetching (rather than directly captured from the redo stream), see Downstream Extract. The columns that are fetched will appear in the trail file as part of the DELETE record. If NOCOMPRESSDELETES is used for Oracle Database data without the FETCHMISSINGCOLUMNS option, only the LOB data that can be read from the logs (without fetching) will be included in the DELETE operation in the trail.

COMPRESSUPDATES | NOCOMPRESSUPDATES

Valid For

Extract

Description

Use the COMPRESSUPDATES and NOCOMPRESSUPDATES parameters for Extract to control the way columns are written to the trail record for UPDATE operations.

COMPRESSUPDATES and NOCOMPRESSUPDATES apply globally for all TABLE statements in a parameter file or they can be used as on-off switches for individual TABLE statements.

GETUPDATEBEFORES is used to decide whether to get before images or not, and COMPRESSUPDATES decides if you want all columns or selected columns. By default, only the after images of update operations are written to the trail, but by using NOCOMPRESSUPDATES along with GETUPDATEBEFORES writes the before image changes as well, which can be used when implementing Conflict Detection and Resolution rules.

These parameters support the following databases:

- DB2 LUW

- DB2 z/OS
- DB2 for i
- MySQL
- SQL Server
- PostgreSQL

For Oracle, refer to the LOGALLSUPCOLS parameter.

Default

```
COMPRESSUPDATES
```

Syntax

```
COMPRESSUPDATES | NOCOMPRESSUPDATES
```

COMPRESSUPDATES
Causes Extract to write only the primary key and the changed columns of a row to the trail for update operations, and is the default value. This provides enough information to update the correct target record (unless conflict resolution is required), while restricting the amount of data that must be processed.
Additionally, if a substitute key is defined with the KEYCOLS option of the TABLE parameter, those columns are written to the trail, whether or not a primary or unique key is defined. See "KEYCOLS (*columns*)" for more information.

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. NOCOMPRESSUPDATES also is required when using the Conflict Detection and Resolution (CDR) feature. See Manual Conflict Detection and Resolution for more information about CDR.
For PostgreSQL, when using NOCOMPRESSUPDATES, LOB column data will only be written in the after image of the record if the column was modified.

COMMIT_SERIALIZATION

Valid For

Parallel Replicat

Description

This parameter forces each transaction to be applied in the exact same order as it was committed on the source database. This is recommended for environments that do any kind of transformation on scheduling columns, or for Active-Active deployments where conflict detection and resolution is being used.

Default

None

COORDSTATINTERVAL

Valid For

Replicat in coordinated mode

Description

Use the `COORDSTATINTERVAL` parameter to set the amount of time, in seconds, between requests for statistics sent by the Replicat coordinator thread to the apply threads. If a thread does not return statistics within an internal heartbeat interval, Replicat logs a warning message. The heartbeat interval is not configurable and is always six times the `COORDSTATINTERVAL` interval. At the default `COORDSTATINTERVAL` interval of 10 seconds, for example, the heartbeat default is one minute (60 seconds).

Default

The minimum value is 0; the maximum value is 2147483647. The default value is 10 seconds

Syntax

```
COORDSTATINTERVAL interval
```

interval
The interval, in seconds, between requests for thread statistics. Valid values are 0 or any positive number.

COORDTIMER

Valid For

Replicat in coordinated mode

Description

Use the `COORDTIMER` parameter to set a base amount of time, in seconds, that the threads and coordinator wait for each other to start. A thread will wait for this base time interval before retrying a connection to the coordinator and it will do this a certain number of times. The coordinator waits for the length of this base time interval and it is reset after every thread is successfully registered. The overall time the coordinator waits before abending is dependent on this timer and it is variable depending on the register time of the threads.

A value of 0 disables this timing procedure. If timing is disabled, the coordinator thread may wait indefinitely for the threads to start, and Replicat will enter a suspended state. In this case, the internal Replicat heartbeat timer is disabled regardless of the COORDSTATINTERVAL setting.

Default

The minimum value is 0; the maximum value is 2147483647. The default value is 180 seconds (three minutes)

Syntax

```
COORDTIMER wait_time
```

wait_time
The amount of time, in seconds, that the coordinator thread waits for the apply threads to start. Valid values are 0 or any positive number.

CRYPTOENGINE

Valid For

GLOBALS

Description

Use the `CRYPTOENGINE` to select which cryptographic library the Oracle GoldenGate processes use to provide implementation of security primitives.

Syntax

```
CRYPTOENGINE (CLASSIC | FIPS140 | NATIVE)
```

`CRYPTOENGINE`
Selects which cryptographic library will the OGG processes use to provide implementation of security primitives.

`CLASSIC`
Uses the Oracle NNZ security framework without FIPS-140 enhancements.

`FIPS140`
Uses the Oracle NNA security framework, but enhanced with the FIPS-140-2 compliant version of the RSA MES shared libraries.

`NATIVE`
For the platforms where this is available, it will use a native library that makes more efficient use of the CPU cryptographic primitives, resulting in higher product throughput when using trail and TCP encryption. Currently, Intel's IPP library version 9.0 is used for Linux.x64 and Windows.x64. All other platforms fall back to `CLASSIC` behavior.

Example

To enable Oracle GoldenGate to use FIPS140-2 compliant encryption, use the following:

```
CRYPTOENGINE FIPS140
```

If this parameter is modifed, added or removed, (like any GLOBALS parameter) all Oracle GoldenGate processes must be restarted, including Manager.

CUSEREXIT

Valid For

Extract when fetching from a multitenant container database (CDB) and Replicat

Description

Use the `CUSEREXIT` parameter to call a custom exit routine written in C programming code from a Windows DLL or UNIX shared object at a defined exit point within Oracle GoldenGate processing. Your user exit routine must be able to accept different events and information from

the Extract and Replicat processes, process the information as desired, and then return a response and information to the caller (the Oracle GoldenGate process that called it).

User exits can be used as an alternative to, or in conjunction with, the data transformation functions that are available within the Oracle GoldenGate solution.

> **Note:**
>
> When using a coordinated Replicat to call a user exit routine, you are responsible for writing the user exits in a thread-safe manner.

For help with creating and implementing user exits, see Using User Exits to Extend Oracle GoldenGate Capabilities.

Default

None

Syntax

```
CUSEREXIT {DLL | shared_object} routine
[, INCLUDEUPDATEBEFORES]
[, PARAMS 'string']
```

{DLL | shared_object}
The name of the Windows DLL or UNIX shared object that contains the user exit function.

routine
The name of the exit routine to be executed.

INCLUDEUPDATEBEFORES
Passes the before images of column values to a user exit. When using this parameter, you must explicitly request the before image by setting the `requesting_before_after_ind` flag to `BEFORE_IMAGE_VAL` within a callback function that supports this flag. Otherwise, only the after image is passed to the user exit. By default, Oracle GoldenGate only works with after images. When using `INCLUDEUPDATEBEFORES` for a user exit that is called from Replicat, always use the `GETUPDATEBEFORES` parameter for the primary Extract process, so that the before image is captured, written to the trail, and causes a `process_record` event in the user exit. In a case where the primary Extract also has a user exit, `GETUPDATEBEFORES` causes both the before image and the after image to be sent to the user exit as separate `EXIT_CALL_PROCESS_RECORD` events.
If the user exit is called from a primary Extract (one that reads the transaction log), only `INCLUDEUPDATEBEFORES` is needed for that Extract. `GETUPDATEBEFORES` is not needed in this case, unless other Oracle GoldenGate processes downstream will need the before image to be written to the trail. `INCLUDEUPDATEBEFORES` does not cause before images to be written to the trail.

PARAMS *'string'*
Passes the specified string at startup. Can be used to pass a properties file, startup parameters, or other string. Enclose the string within single 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`.

Examples

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, &
  PARAMS 'init.properties'
```

Example 5

```
CUSEREXIT cuserexit.dll MyUserExit, &
  INCLUDEUPDATEBEFORES, PARAMS 'Some text to start with during startup'
```

DBOPTIONS

Valid For

Extract and Replicat

Description

Use the `DBOPTIONS` parameter to specify database options. This is a global parameter, applying to all `TABLE` or `MAP` statements in the parameter file. Some options for the `DBOPTIONS` parameters apply only to Extract or Replicat.

The `DBOPTIONS` parameter can be placed anywhere in the parameter file irrespective of other parameters.

Default

None

Syntax

```
DBOPTIONS
[ALLOWLOBDATATRUNCATE | NOALLOWLOBDATATRUNCATE]
[ALLOWUNUSEDCOLUMN | NOALLOWUNUSEDCOLUMN]
[ALLOWNONSTANDARDINTERVALDATA]
[BINDCHARFORBITASCHAR]
[CATALOGCONNECT | NOCATALOGCONNECT]
[CONNECTIONPORT port]
[DECRYPTPASSWORD shared_secret ENCRYPTKEY {DEFAULT | key_name}]
[DEFERREFCONST]
[DISABLECOMMITNOWAIT]
[DISABLELOBCACHING]
[ENABLE_INSTANTIATION_FILTERING]
[EMPTYLOBSTRING 'string']
[FETCHBATCHSIZE records]
```

```
[FETCHCHECKFREQ seconds]
[FETCHLOBS | NOFETCHLOBS]
[FETCHRETRYCOUNT number]
[FETCHTIMEOUT seconds | NOFECHTIMEOUT]
[FORCE_XML_ESCAPE_CONVERSION]
[HOST {DNS_name | IP_address}]
[INTEGRATEDPARAMS(parameter[, ...])]
[LIMITROWS | NOLIMITROWS]
[LOBBUFSIZE bytes]
[LOBWRITESIZE bytes]
[MAXLOBKEYLEN datalength]
[SESSIONPOOLMAX max_value |
[SESSIONPOOLMIN min_value][SESSIONPOOLINCR increment_value]
[SETTAG [tag_value | NULL] ]
[SHOWINFOMESSAGES]
[SHOWWARNINGS]
[SKIPTEMPLOB | NOSKIPTEMPLOB]
[SOURCE_DB_NAME src_dbase_global_name]
[SPTHREAD | NOSPTHREAD]
[SQLMODE]
[SUPPRESSTEMPORALUPDATES]
[SUPPRESSTRIGGERS | NOSUPPRESSTRIGGERS]
[TDSPACKETSIZE bytes]
[TRANSNAME trans_name][USEDATABASEENCODING]
[XMLBUFSIZE bytes]
```

ALLOWLOBDATATRUNCATE | NOALLOWLOBDATATRUNCATE

Valid for Replicat for Db2 LUW and MySQL. `ALLOWLOBDATATRUNCATE` prevents Replicat from abending when replicated `LOB` data is too large for a target `CHAR`, `VARCHAR`, `BINARY` or `VARBINARY` column and is applicable to target LOB columns only. or replicat of Db2 LUW, `ALLOWLOBDATATRUNCATE` prevents Replicat from abending when replicated `LOB` data is too large for a target `LOB` column. The `LOB` data is truncated to the maximum size of the target column without any further error messages or warnings.

`NOALLOWLOBDATATRUNCATE` is the default and causes Replicat to abend with an error message if the replicated LOB is too large.

ALLOWUNUSEDCOLUMN | NOALLOWUNUSEDCOLUMN

Valid for Extract for Oracle. Controls whether Extract abends when it encounters a table with an unused column.

The default is `ALLOWUNUSEDCOLUMN`. When Extract encounters a table with an unused column, it continues processing and generates a warning.When using this parameter, either the same unused column must exist on the target or a source definitions file for the table must be specified to Replicat, so that the correct metadata mapping can be performed.

`NOALLOWUNUSEDCOLUMN` causes Extract to abend on unused columns.

ALLOWNONSTANDARDINTERVALDATA

Valid for PostgreSQL.

Use `DBOPTIONS ALLOWNONSTANDARDINTERVALDATA` in the Extract parameter file to capture the mixed sign interval data (or any other format of interval data, which is not supported by Oracle GoldenGate) as a string (not as standard interval data). When this option is used, the format of the interval data that gets written to the trail and gets applied into the target `CHAR` column is as follows:

year-component-sign years-months days-component-sign days hour-component-sign hours:minutes:seconds.fractional_seconds

For example, `+1026-9 +0 +0:0:22.000000` should be interpreted as 1026 years 9 months 0 days 0 hours 0 minutes 22 seconds. `-0-0 -0 -8` should be interpreted as 0 years 0 months 0 days -8 hours. `+1-3 +0 +3:20` should be interpreted as 1 years 3 months 0 days 3 hours 20 minutes.
In case of Replicat, if the source interval data was captured using `DBOPTIONS ALLOWNONSTANDARDINTERVALDATA` and written as a string to the trail, the corresponding source column is allowed to be mapped to either a `char` or a `binary` type column on the target.

BINDCHARFORBITASCHAR
Valid for DEFGEN, Extract, and Replicat for DB2 for i. Allows columns that are defined as `CHAR` or `VARCHAR` with CCSID 65535, or `CHAR` and `VARCHAR FOR BIT DATA` to be treated as if the field had a normal translatable encoding. The encoding is picked up from the job CCSID. When this option is in effect, DEFGEN does not indicate that the field is binary in the `defs` file.

CATALOGCONNECT | NOCATALOGCONNECT
Valid for Extract and Replicat for ODBC databases.
By default, Oracle GoldenGate creates a new connection for catalog queries, but you can use `NOCATALOGCONNECT` to prevent that.

CONNECTIONPORT *port*
Valid for Replicat for multi-daemon MySQL. Specifies the TCP/IP port of the instance to which Replicat must connect. The minimum value is 1 and the default value is 3306.

DEFERREFCONST | NODEFERREFCONST
Valid for nonintegrated Replicat for Oracle. Default option for parallel integrated Replicat. Sets constraints to `DEFERRABLE` to delay the checking and enforcement of cascade delete and cascade update referential integrity constraints by the Oracle target database until the Replicat transaction is committed. At that point, if there are constraint violations, an error is generated. Integrated Replicat does not require disabling of referential constraints on the target system.
You can use `DEFERREFCONST` instead of disabling the constraints on the target tables or setting them to `DEFERRED`. When used, `DEFERREFCONST` defers both `DEFERABLE` and `NOT DEFERABLE` constraints. `DEFERREFCONST` applies to every transaction that is processed by Replicat. `DEFERREFCONST` parameter works on Oracle 11.2.0.2 and up.
If used with an Oracle Database release that does not support this functionality, `DEFERREFCONST` is ignored without returning a notification to the Oracle GoldenGate log. To handle errors on the commit operation, you can use `REPERROR` at the root level of the parameter file and specify the `TRANSDISCARD` or `TRANSEXCEPTION` option.

> **Note:**
>
> Do not to use with `DEFERREFCONST` coordinated Replicat because there is no way to guarantee that related rows in parent and child tables are processed by same thread

Use the `NODEFERREFCONST` option to disable the `DEFERREFCONST` option.

DISABLECOMMITNOWAIT
Valid for Replicat for Oracle.
Disables the use of asynchronous `COMMIT` by Replicat. An asynchronous `COMMIT` statement includes the `NOWAIT` option.

When DISABLECOMMITNOWAIT is used, Replicat issues a standard synchronous COMMIT (COMMIT with WAIT option).

DISABLELOBCACHING
Valid for nonintegrated Replicat for Oracle.
Disables Oracle's LOB caching mechanism. By default, Replicat enables Oracle's LOB caching mechanism.

ENABLE_INSTANTIATION_FILTERING
Valid for Oracle.
Enables automatic per table instantiation CSN filtering on tables instantiated using the SET_INSTANTIATION_CSN command.

FETCHBATCHSIZE *records*
Valid for Extract for Oracle, DB2 for i, DB2 z/OS, PostgreSQL, and SQL Server.
Enables array fetches for initial loads to improve performance, rather than one row at a time. It is used for Initial Load Extract.
Valid values for Oracle, DB2 for i, DB2 z/OS, and SQL Server are 0 through 1000000 records per fetch. Valid values for DB2 LUW are 1 through 1000000 records per fetch; zero (0) is *not* a valid value.
The default is 1000. Performance slows when batch size gets very small or very large. If the table contains LOB data, Extract reverts to single-row fetch mode, and then resumes batch fetch mode afterward.

FETCHCHECKFREQ *seconds*
Valid for integrated Extract for Oracle.
Specifies the number of seconds that Extract waits between each fetch check for the ADG to catch up. A low number improves latency though increases the number of queries of current_scn from v$database. The default is 3 seconds; the maximum is 120 seconds.

FETCHLOBS | NOFETCHLOBS
Valid for Extract for DB2 for z/OS and DB2 for LUW.
Suppresses the fetching of LOBs directly from the database table when the LOB options for the table are set to NOT LOGGED. With NOT LOGGED, the value for the column is not available in the transaction logs and can only be obtained from the table itself. By default, Oracle GoldenGate captures changes to LOBs from the transaction logs. The default is FETCHLOBS.

FETCHRETRYCOUNT *number*
Valid for Extract for Oracle.
Specifies the number of times that Extract tries before it reports ADG progress or the reason for no progress when waiting for the ADG to catch up. This value is multiplied with FETCHCHECKFREQ to determine approximately how often the ADG progress is reported. The default value for FETCHRETRYCOUNT is 5 and the valid range of values is 0 - 1000.

FETCHTIMEOUT *seconds* | NOFETCHTIMEOUT
Valid for Extract for Oracle.
Specifies the number of seconds that Extract will wait after which it will abend when ADG makes no progress. No progress can be because the MRP is not running or because it is not applying redo changes. When this occurs, the ADG database should be examined. The default is 30 seconds; valid values are 0 - 4294967295 (ub4 max value) seconds.
NOFETCHTIMEOUT means never timeout (the same as FETCHTIMEOUT 0) and seconds cannot be specified with it.

FORCE_XML_ESCAPE_CONVERSION
For trail fie formats of Oracle GoldenGate19c, if FORCE_XML_ESCAPE_CONVERSION is enabled, Replicat will escape the linefeed characters for the character types in the ANYDATA columns. If

this parameter is enabled for a trail file with a format of `19.1` or higher, it is ignored because Extract already performs the linefeed escape.
This parameter only affects `ANYDATA` columns when `NOUSENATIVEOBJSUPPORT` is turned on for Extract.

> **Note:**
>
> This parameter option doesn't affect `ANYDATA` columns retrieved from the database by Logminer in native mode.

`HOST {DNS_name | IP_address}`
Valid for Replicat for multi-daemon MySQL.
Specifies the DNS name or IP address of the system that hosts the instance to which Replicat must connect.

`INTEGRATEDPARAMS(parameter[, ...])`
Valid for Integrated Replicat for Oracle.
Passes settings for parameters that control the database inbound server within the target Oracle database.
You can use the `commit_serialization` option with `INTEGRATEDPARAMS` for integrated Replicat but not for parallel Replicat in integrated mode. Setting internal database parameters for Extract is done using `TRANLOGOPTIONS INTEGRATEDPARAMS`.
For more information about integrated Replicat and a list of supported inbound server parameters, see Select a Replicat Type for the Deployment in *Oracle GoldenGate Microservices Documentation*.

`LIMITROWS | NOLIMITROWS`
Valid for Replicat for MariaDB, MySQL, Oracle, SingleStore, SQL Server, Sybase, and TimesTen.
`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.
`LIMITROWS` is the default. `LIMITROWS` and `NOLIMITROWS` apply globally to all `MAP` statements in a parameter file.
For MySQL, `LIMITROWS` uses a `LIMIT 1` clause in the `UPDATE` or `DELETE` statement.
For Oracle targets, `LIMITROWS` (the default) must be used. It uses either `WHERE ROWNUM = 1` or `AND ROWNUM = 1` in the `WHERE` clause.
For SQL Server, `LIMITROWS` uses a `SET ROWCOUNT 1` clause before the `UPDATE` or `DELETE` statement.
`NOLIMITROWS` permits multiple rows to be updated or deleted by the same Replicat SQL statement.

`LOBBUFSIZE bytes`
Valid for Extract for Db2 LUW, Db2 for i, Db2 z/OS, Oracle, PostgreSQL, and SQL Server.
For Oracle, it determines the memory buffer size in bytes to allocate for each embedded LOB attribute that is in an Oracle object type.
For non-Oracle databases, it specifies the buffer size to fetch LOB data during initial load.
Valid values are from 1024 and 104857600 bytes. The default is 1048576 bytes.
For Oracle, if the length of embedded LOB exceeds the specified `LOBBUFSIZE` size, an error message similar to the following is generated:

```
GGS ERROR    ZZ-0L3  Buffer overflow, needed: 2048, allocated: 1024.
```

`LOBWRITESIZE bytes`
Valid for nonintegrated Replicat for Oracle.

Specifies a fragment size in bytes 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 if `DBOPTIONS NOSKIPTEMPLOB` is specified, or 1MB if `DBOPTIONS SKIPTEMPLOB` is specified. Valid values are from 2,048 bytes to 2,097,152 bytes (2MB).
By default, Replicat enables Oracle's LOB caching mechanism. To disable Oracle's LOB caching, use the `DISABLELOBCACHING` option of `DBOPTIONS`.

MAXLOBKEYLEN *datalength*
Valid for Extract for PostgreSQL.

Sets the LOB datatype columns to be processed as part of the key. The valid values of *datalength* parameter can be from 1 to 8000. The LOBs that can be part of the key are:

- Bytea
- Char
- Citext
- Text
- Varchar

SESSIONPOOLMAX *max_value*
Valid for Extract in integrated mode for Oracle.
Sets a maximum value for the number of sessions in the OCI Session Pool, which is used by Extract for fetching from a container database. The default value is 10 sessions. Must be specified before the `USERIDALIAS` parameter; otherwise will be ignored and the default will be used.

SESSIONPOOLMIN *min_value*
Valid for Extract in integrated mode for Oracle.
Sets a minimum value for the number of sessions in the OCI Session Pool, which is used by Extract for fetching from a container database. The default value is 2 sessions. Must be specified before the `USERIDALIAS` parameter; otherwise will be ignored and the default will be used.

SESSIONPOOLINCR *increment_value*
Valid for Extract in integrated mode for Oracle.
Sets a value for the number of incremental sessions that can be added to the OCI Session Pool, which is used by Extract for fetching from a container database. The default value is 2 sessions. Must be specified before the `USERIDALIAS` parameter; otherwise will be ignored and the default will be used.

SETTAG [*tag_value* | NULL
Valid for Replicat for Oracle.
Sets the value for an Oracle redo tag that will be used to identify the transactions of the associated Replicat in the redo log. A redo tag can also be used to identify transactions other than those of Replicat. This parameter is recommended over `EXCLUDEUSER` and `TRACETABLE`. Use this option to prevent cycling (loop-back) of Replicat the individual records in a bi-directional configuration or to filter other transactions from capture. The default `SETTAG` value is 00 and is limited to 2K bytes. A valid value is any single Oracle Streams tag. A tag value can be up to 2000 hexadecimal digits (0-9 A-F) long.

Transactions in the redo that are marked with the specified tag can be filtered by an Extract that has the TRANLOGOPTIONS parameter with the EXCLUDETAG option set to the *tag_value*. Use tag-based filtering to prevent cycling (loop-back) of Replicat transactions in a bi-directional configuration or to filter other transactions from capture. For more information, see TRANLOGOPTIONS.
You can disable the tagging of DDL by using the DDLOPTIONS parameter with the NOTAG option.

hex_value
A hexadecimal value from 0 through F. The default value is 00. The following are valid examples:

```
DBOPTIONS SETTAG 00112233445566778899AABBCCDDEEFF
DBOPTIONS SETTAG 00112233445566778899aabbccddeeff
DBOPTIONS SETTAG 123
```

NULL
Disables tag-based filtering for the associated Replicat.

SKIPTEMPLOB | NOSKIPTEMPLOB
Valid for Replicat for Oracle Database versions 11*g* and 12*c*. Controls how LOBs are applied to a target Oracle database. The default of SKIPTEMPLOB .
SKIPTEMPLOB improves performance by directly writing LOB data to the target LOB column. Replicat creates a SQL statement with an empty LOB value and returns the LOB locator to the bind variable. After the SQL statement is executed successfully, the LOB data is written directly to the LOB column using the returned LOB locator.
NOSKIPTEMPLOB uses a temporary LOB in the SQL statement. Replicat declares a bind variable within SQL statement and associates a temporary LOB, then writes to the temporary LOB. The Oracle Database applies the LOB column data from the temporary LOB.
SKIPTEMPLOB applies to INSERT and UPDATE operations that contain LOB data. It does not apply if the table has a functional index with a LOB column, if the LOB data is NULL, empty, or stored inline. It does not apply to partial LOB operations.
SKIPTEMPLOB causes Replicat to generate/perform 1 DML+ n LOB_WRITE (piece-wise) operations when updating/inserting a row with LOB columns. However, SKIPTEMPLOB should not be used with FETCHPARTIALLOB (an Extract Parameter) because it results in excessive fetching.
NOSKIPTEMPLOB is provided for backward compatibility; otherwise the default of SKIPTEMPLOB should be retained.

SOURCE_DB_NAME *src_dbase_global_name*
Valid for Oracle.
Indicates the Global Name of the Trail Source Database. It is used to query the relevant instantiation information when DBOPTIONS ENABLE_INSTANTIATION_FILTERING is enabled. This option is optional for instantiation filtering in a 12.2. trail file with metadata enabled.
When the source has no DOMAIN, do not specify a DOMAIN for the downstream database.

SPTHREAD | NOSPTHREAD
Valid for Extract and Replicat. Not valid for Oracle and MySQL.
Creates a separate database connection thread for using SQLEXEC to execute stored procedures. The default is NOSPTHREAD.

SQLMODE
With this option enabled, the sql_mode variable is set to to 'ANSI_QUOTES' (set sql_mode = 'ANSI_QUOTES'). Treat the double quotes (") as an identifier quote character (like the ` quote

character) and not as a string quote character. You can still use ` to quote identifiers with this mode enabled. With ANSI_QUOTES enabled, you cannot use double quotes (") to quote literal strings, because it is interpreted as an identifier.
For more information, see Server SQL Modes.

SUPPRESSTEMPORALUPDATES
Valid for DB2 LUW 10.1 FixPack 2 and greater replication of temporal table.
Use SUPPRESSTEMPORALUPDATES to replicate system-period and bitemporal tables along with associated history tables. Oracle GoldenGate replicates the row begin, row end, and transaction start id columns along with the other columns of the table. You must ensure that the database instance has the execute permission to run the SYSPROC.SET_MAINT_MODE_RECORD_NO_TEMPORALHISTORY¿ stored procedure at the apply side. By default, Oracle GoldenGate does not replicate row begin, row end, and transaction start id columns. To preserve the original values of these columns, implement one of the followings options.

- Add extra timestamp columns in the target temporal table and map the columns accordingly.
- Use a non-temporal table at the apply side and map the columns accordingly.

Replication in Non-Oracle Environment:
In non-Oracle environments where there is no temporal tables at the apply side, you need to set the row begin, row end and transaction start id columns value. These source columns will have timestamp values that the target database may not support. You should first use the map conversion functions to convert these values into the format that target database supports, and then map the columns accordingly. For example, MySQL has a DATETIME range from `1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999¿'. You cannot replicate a DB2 LUW timestamp value of `0001-01-01-00.00.00.000000000000¿ to MySQL. To replicate such values you must convert this value into the MySQL DATETIME format. For example, if a system-period or bitemporal table has the following timestamp column:

```
SYS_START
------------------------------
0001-01-01-00.00.00.000000000000
```

Then to replicate this column into MySQL, you would use the function colmap() as follows:

```
map <source_schema>.<source_table>, target <target_schema>.<target_table>
colmap(sys_start= @IF( ( @NUMSTR( @STREXT(sys_start,1,4))) > 1000, sys_start,
'1000-01-01 00.00.00.000000'));
```

Initial Load of Temporal Table:
Oracle GoldenGate supports initial load of temporal table as usual.
Take into account the following considerations with temporal table:

- Replication between system-period and application-period temporal table is not supported.
- Replication from a non-temporal table to a temporal table is not supported.
- Replication of system-period, bi-temporal tables, and SUPPRESSTEMPORALUPDATES with the INSERTALLRECORDS parameter is not supported.
- If any unique index is created for application-period temporal table using BUSINESS_TIME WITHOUT OVERLAPS for the target table, then the same unique index must be created for the source table.
- Bidirectional replication between temporal tables is advised only with the default.

- CDR is supported only with SUPPRESSTEMPORALUPDATES. There is no CDR support in bidirectional replication.
- By default, there are inconsistencies in row begin, row end, and transaction start id columns of the temporal tables when the source and target databases operate with different time zones. These timestamp columns of system-period and bitemporal tables are automatically populated by the respective database managers and will have values as per the respective time zones of the databases.
- Using the default with GETUPDATEBEFORES is in the replicate parameter file, you cannot use the row begin, row end, and transaction start id columns in any delta calculations. For example, taking before and after image of such columns in any kind of calculations is not possible. These columns can be used in delta calculations using SUPPRESSTEMPORALUPDATES.

SUPPRESSTRIGGERS | NOSUPPRESSTRIGGERS
Valid for Integrated Replicat and Classic Replicat for Oracle.
Controls whether or not triggers are fired during the Replicat session. Provides an alternative to manually disabling triggers. (Integrated Replicat does not require disabling of triggers on the target system.)
SUPPRESSTRIGGERS is the default and prevents triggers from firing on target objects that are configured for replication with Oracle GoldenGate. SUPPRESSTRIGGERS is valid for Oracle Database 12*c*, 11*g* (11.2.0.2), and later 11*g* R2 releases. SUPPRESSTRIGGERS is not valid for 11*g* R1.
To allow a specific trigger to fire, you can use the following SQLEXEC statement in the Replicat parameter file, where *trigger_owner* is the owner of the trigger and *trigger_name* is the name of the trigger.

```
SQLEXEC 'DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY('"S1"','"MY_TRIGGER"',FALSE);'
```

> **Note:**
>
> Once this SQLEXEC is executed with FALSE, the trigger will continue to fire until the command is run again with a setting of TRUE.

NOSUPPRESSTRIGGERS allows target triggers to fire. To use [NO]SUPPRESSTRIGGERS, the Replicat user must have the privileges granted through the OGG_APPLY user role available from Oracle Database 23ai and higher or the dbms_goldengate_auth.grant_admin_privilege package for Oracle Database 21c and lower. This procedure is part of the Oracle database installation. See the database documentation for more information.
The USERIDALIAS parameter must precede a DBOPTIONS statement that contains SUPPRESSTRIGGERS or NOSUPPRESSTRIGGERS.

TRANSNAME *trans_name*
Valid for Replicat for SQL Server.
Allows an individual Replicat to use a specific transaction name that is specified in the parameter file. The *trans_name* is the name of the transaction that the Replicat uses for target DML transactions and overrides the default ggs_repl transaction name when used.

USEDATABASEENCODING
By default, the DB2 for i Extract converts all text data to UTF-8 for non-DBCS data and UTF-16 for DBCS data. Using this option causes the Extract to store all text data in the trail in its native character encoding for non-DBCS data. Currently, DBCS (GRAPHIC/VARGRAPHIC/DBCLOB) data continues to be converted to UTF-16 whether this parameter is provided or not.

For CCSID values that are not supported by Oracle GoldenGate, the Extract converts the data to UTF-8 for non-DBCS data and UTF-16 for DBCS data to ensure compatibility for all Replicats.

`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 104857600 bytes.

Examples

Example 1

```
DBOPTIONS HOST 127.0.0.1, CONNECTIONPORT 3307
```

Example 2

```
DBOPTIONS DECRYPTPASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH ENCRYPTKEY DEFAULT
```

Example 3

```
DBOPTIONS TDSPACKETSIZE 2048
```

Example 4

```
DBOPTIONS FETCHBATCHSIZE 2000
```

Example 5

```
DBOOPTION XMLBUFSIZE 2097152
```

DDL

Valid For

Extract and Replicat

> **Note:**
>
> DDL replication is only supported between Oracle to Oracle databases and between MySQL to MySQL databases.

Description

Use the `DDL` parameter to:

- enable DDL support
- filter DDL operations
- configure a processing action based on a DDL record

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

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

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

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

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

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

- The filtering options of the `DDL` parameter are valid for a primary Extract that captures from the transaction source, but not for a data-pump Extract.
- When combined, multiple filter option specifications are linked logically as `AND` statements.
- All filter criteria specified with multiple options must be satisfied for a DDL statement to be replicated.
- When using complex filtering criteria in a `DDL` parameter statement, it is recommended that you test your configuration in a test environment before using it in production.
- See Example 1, Example for more information.

> **Note:**
>
> Do not use the `DDL` parameter for an Extract data pump. These process types do not permit the mapping or conversion of DDL and will propagate DDL records automatically in pass-through mode. DDL that is performed on a source table (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 additional information about how to use Oracle GoldenGate DDL support, see Oracle: DDL Replication and MySQL: DDL Replication in *Oracle GoldenGate Microservices Documentation*.

Syntax

```
DDL [
{INCLUDE | EXCLUDE}
  [, MAPPED | UNMAPPED | OTHER | ALL]
  [, OPTYPE type]
  [, OBJTYPE 'type']
  [, SOURCECATALOG catalog | ALLCATALOGS]
  [, ALLOWEMPTYOBJECT]
  [, ALLOWEMPTYOWNER]
  [, OBJNAME name]
  [, INSTR 'string']
  [, INSTRWORDS 'word_list']
  [, INSTRCOMMENTS 'comment_string']
  [, INSTRCOMMENTSWORDS 'word_list']
  [, STAYMETADATA]
  [, EVENTACTIONS (action)
```

```
]
[...]
```

DDL Filtering Options

The following are the syntax options for filtering and operating upon the DDL that is replicated by Oracle GoldenGate. These options apply to the INCLUDE and EXCLUDE clauses of the DDL parameter and other parameters that support DDL replication.

INCLUDE | EXCLUDE
Use INCLUDE or 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 the other filtering options of the DDL parameter.
If you use EXCLUDE, you must create a corresponding INCLUDE clause. For example, the following is invalid:

```
DDL EXCLUDE OBJNAME "hr".*
```

However, you can use either of the following:

```
DDL INCLUDE ALL, EXCLUDE OBJNAME "hr"."*"
DDL INCLUDE OBJNAME fin.* EXCLUDE OBJNAME "fin.ss"
```

An EXCLUDE takes priority over any INCLUDEs that contain the same criteria. You can use multiple inclusion and exclusion clauses.
Do not include any Oracle GoldenGate installed DDL objects in a DDL parameter, in a TABLE parameter, or in a MAP parameter, nor in a TABLEEXCLUDE or MAPEXCLUDE parameter. Make certain that wildcard specifications in those parameters do not include Oracle GoldenGate-installed DDL objects. These objects must not be part of the Oracle GoldenGate configuration, but the Extract process must be aware of operations on them, and that is why you must *not* explicitly exclude them from the configuration with an EXCLUDE, TABLEEXCLUDE, or MAPEXCLUDE parameter statement.

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.

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

thus preventing the need to stop and start the Oracle GoldenGate processes. The following special conditions apply to `DDL EXCLUDE ALL`:

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

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

```
DDL INCLUDE OPTYPE ALTER
```

`OBJTYPE 'type'`
Use `OBJTYPE` to apply `INCLUDE` or `EXCLUDE` to a specific type of database object. For *`type`*, use any object type that is valid for the database, such as `TABLE`, `INDEX`, and `TRIGGER`. For an Oracle materialized view and materialized views log, the correct types are `snapshot` and `snapshot log`, respectively. Enclose the name of the object type within single quotes. For example:

```
DDL INCLUDE OBJTYPE 'INDEX'
DDL INCLUDE OBJTYPE 'SNAPSHOT'
```

For Oracle object type `USER`, do not use the `OBJNAME` option, because `OBJNAME` expects `owner.object` or *`container.owner.object`* whereas `USER` only has a schema.

`SOURCECATALOG catalog | ALLCATALOGS`
Use these options to specify how unqualified object names in an `OBJNAME` clause are resolved to the correct container. Use these options when the source database is an Oracle container database.
`SOURCECATALOG` specifies a default container for all of the object names that are specified in the same `INCLUDE` or `EXCLUDE` clause. To take effect, `SOURCECATALOG` must be specified before the `OBJNAME` specification. See "SOURCECATALOG" for more information including using statements that contain two-part names, where three-part object names are required to fully identify an object.
`ALLCATALOGS` specifies that all of the containers of the database should be considered when resolving object names that are specified in the same `INCLUDE` or `EXCLUDE` clause. `ALLCATALOGS` can be placed before or after the `OBJNAME` specification.
The following is the order of precedence that is given when there are different catalog specifications in a parameter file:

1. `ALLCATALOGS` in an `INCLUDE` or `EXCLUDE` clause overrides all `SOURCECATALOG` specifications in the `INCLUDE` or `EXCLUDE` clause and at the root of the parameter file, and it overrides the container specification of a fully qualified object name in the `OBJNAME` clause.
2. An explicit catalog specification in the `OBJNAME` clause overrides all instances of `SOURCECATALOG` (but not `ALLCATALOGS`).
3. `SOURCECATALOG` in an `INCLUDE` or `EXCLUDE` clause overrides the global `SOURCECATALOG` parameter that is specified at the root of the `TABLE` or `MAP` statement.

4. The global SOURCECATALOG parameter takes effect for any unqualified object names in OBJNAME clauses if the INCLUDE or EXCLUDE clause does not specify SOURCECATALOG or ALLCATALOGS.
5. In the absence of any of the preceding parameters, all catalogs are considered.

ALLOWEMPTYOBJECT
Use ALLOWEMPTYOBJECT to allow an OBJNAME specification to process DDL that contains no object name. For example:

```
DDL INCLUDE OBJNAME sch.* ALLOWEMPTYOBJECT
```

ALLOWEMPTYOWNER
Use ALLOWEMPTYOWNER to allow an OBJNAME specification to process DDL that contains no owner name. For example:

```
DDL INCLUDE OBJNAME pdb.sch.* ALLOWEMPTYOWNER
```

OBJNAME *name*
Use OBJNAME to apply INCLUDE or EXCLUDE to the fully qualified name of an object. To specify two-part and three-part object names and wildcards correctly, see Using Wildcards in Command Arguments.
Enclose case-sensitive object names within double quote marks.
Case-insensitive example:

```
DDL INCLUDE OBJNAME accounts.*
```

Case-sensitive example:

```
DDL INCLUDE OBJNAME accounts."cust"
```

Do not use OBJNAME for the Oracle USER object, because OBJNAME expects *owner.object* or *container.owner.object*, whereas USER only has a schema.
When using OBJNAME with MAPPED in a Replicat parameter file, the value for OBJNAME must refer to the name specified with the TARGET clause of the MAP statement. For example, given the following MAP statement, the correct value is OBJNAME fin2.*.

```
MAP fin.exp_*, TARGET fin2.*;
```

In the following example, a CREATE TABLE statement executes as follows on the source:

```
CREATE TABLE fin.exp_phone;
```

That same statement executes as follows 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 USERIDALIAS parameter.
For DDL that creates derived objects, such as a trigger, the value for OBJNAME must be the name of the base object, not the name of the derived object.
For example, to include the following DDL statement, the correct value is hr.accounts, not hr.insert_trig.

```
CREATE TRIGGER hr.insert_trig ON hr.accounts;
```

For RENAME operations, the value for OBJNAME must be the new table name. For example, to include the following DDL statement, the correct value is hr.acct.

```
ALTER TABLE hr.accounts RENAME TO acct;
```

INSTR '*string*'
Use INSTR to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within the command syntax itself, but not within comments. For example, the following excludes DDL that creates an index.

```
DDL INCLUDE ALL EXCLUDE INSTR 'CREATE INDEX'
```

Enclose the string within single quotes. The string search is not case sensitive.
INSTR does not support single quotation marks (' ') that are within the string, nor does it support NULL values.

INSTRCOMMENTS '*comment_string*'
(Valid for Oracle) Use INSTRCOMMENTS to apply INCLUDE or EXCLUDE to DDL statements that contain a specific character string within a comment, but not within the DDL command itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.
For example, the following excludes DDL statements that include the string 'source only' in the comments.

```
DDL INCLUDE ALL EXCLUDE INSTRCOMMENTS 'SOURCE ONLY'
```

In this example, DDL statements such as the following are not replicated.

```
CREATE USER john IDENTIFIED BY john /*source only*/;
```

Enclose the string within single quotes. The string search is not case sensitive. You can combine INSTR and INSTRCOMMENTS to filter on a string in the command syntax and in the comments of the same DDL statement.
INSTRCOMMENTS does not support single quotation marks (' ') that are within the string, nor does it support NULL values.

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:

```
DDL INCLUDE OPTYPE ALTER OBJTEYP 'TABLE' INSTRWORDS 'ALTER CONSTRAINT " xyz"'
```

This example matches the following DDL statements:

```
ALTER TABLE ADD CONSTRAINT xyz CHECK

ALTER TABLE DROP CONSTRAINT xyz
```

INSTRWORDS does not support single quotation marks (' ') that are within the string, nor does it support NULL values.

INSTRCOMMENTSWORDS '*word_list*'
(Valid for Oracle) Works the same way as INSTRWORDS, but only applies to comments within a DDL statement, not the DDL syntax itself. By using INSTRCOMMENTS, you can use comments as a filtering agent.

INSTRCOMMENTSWORDS does not support single quotation marks (' ') that are within the string, nor does it support NULL values.
You can combine INSTRWORDS and INSTRCOMMENTSWORDS to filter on a string in the command syntax and in the comments of the same DDL statement.

STAYMETADATA
(Valid for Oracle). Prevents metadata from being captured by Extract or applied by Replicat.
When Extract first encounters DML on a table, it retrieves the metadata for that table. When DDL is encountered on that table, the old metadata is invalidated. The next DML on that table is matched to the new metadata so that the target table structure always is up-to-date with that of the source.
However, if you know that a particular DDL operation will not affect the table's metadata, you can use STAYMETADATA so that the current metadata is not retrieved or replicated. This is a performance improvement that has benefit for such operations as imports and exports, where such DDL as truncates and the disabling of constraints are often performed. These operations do not affect table structure, as it relates to the integrity of subsequent data replication, so they can be ignored in such cases. For example ALTER TABLE ADD FOREIGN KEY does not affect table metadata.
An example of how this can be applied selectively is as follows:

```
DDL INCLUDE ALL INCLUDE STAYMETADATA OBJNAME xyz
```

This example states that all DDL is to be included for replication, but only DDL that operates on object xyz will be subject to STAYMETADATA.
STAYMETADATA also can be used the same way in an EXCLUDE clause.
STAYMETADATA must be used the same way on the source and target to ensure metadata integrity.
When STAYMETADATA is in use, a message is added to the report file. DDL reporting is controlled by the DDLOPTIONS parameter with the REPORT option.
This same functionality can be applied globally to all DDL that occurs on the source by using the @ddl_staymetadata scripts:

- @ddl_staymetadata_on globally turns off metadata versioning.
- @ddl_staymetadata_off globally enables metadata versioning again.

This option should be used with the assistance of Oracle GoldenGate technical support staff, because it might not always be apparent which DDL affects object metadata. If improperly used, it can compromise the integrity of the replication environment.

EVENTACTIONS (*action*)
Causes the Extract or Replicat process take a defined action based on a DDL record in the transaction log or trail, which is known as the *event record*. The DDL event is triggered if the DDL record is eligible to be written to the trail by Extract or a data pump, or to be executed by Replicat, as determined by the other filtering options of the DDL parameter.
You can use this system to customize processing based on database events.
For *action*, see EVENTACTIONS under the MAP and TABLE parameters.
Guidelines for using EVENTACTIONS on DDL records:

- CHECKPOINTBEFORE: Since each DDL record is autonomous, the DDL record is guaranteed to be the start of a transaction; therefore, the CHECKPOINT BEFORE event action is implied for a DDL record.

- IGNORE: This option is not valid for DDL records. Because DDL operations are autonomous, ignoring a record is equivalent to ignoring the entire transaction.

EVENTACTIONS does not support the following DDL objects because they are derived objects:

- indexes
- triggers
- synonyms
- RENAME on a table and ALTER TABLE RENAME

Examples

Example 1 Combining DDL Parameter Options

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

```
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), but only if those tables are owned by users and their names start with tab,
- INCLUDE all DDL operation types for all tables that are mapped with a TABLE or MAP statement (MAPPED scope),
- EXCLUDE all DDL operation types for all tables that are MAPPED in scope, but only if those tables are owned by temporary and only if their names begin with tab.

Example 2 Including an Event Action

The following example specifies an event action of REPORT for all DDL records.

```
DDL INCLUDE ALL EVENTACTIONS (REPORT)
```

Example 3 Using an Event Action on a Subset of DDL

The following example shows how EVENTACTIONS can be used on a subset of the DDL. All DDL is to be replicated, but only the DDL that is executed on explicitly named objects qualifies to trigger the event actions of REPORT and LOG.

```
DDL INCLUDE ALL &
    INCLUDE OBJNAME sales.t* EVENTACTIONS (REPORT) &
    INCLUDE OBJNAME fin.my_tab EVENTACTIONS (LOG) &
```

Example 4

The following example demonstrates the different ways to specify catalog names for DDL that is issued on objects in a source Oracle container database.

- This includes pdb1.sch1.obj1 and pdb2.sch2.obj2 for DDL processing.

```
SOURCECATALOG pdb1
DDL INCLUDE OBJNAME sch1.obj1 INCLUDE SOURCECATALOG pdb2 OBJNAME sch2.obj2
```

- This includes all objects with the name `sch.obj` in any catalog for DDL processing.

  ```
  DDL INCLUDE ALLCATALOGS OBJNAME sch.obj
  ```

- This also includes all objects with the name `sch.obj` in any catalog for DDL processing, because `ALLCATALOGS` overrides any other catalog specification.

  ```
  DDL INCLUDE ALLCATALOGS OBJNAME pdb.sch.obj
  ```

Example 5
The following shows the combined use of `ALLOWEMPTYOBJECT` and `ALLOWEMPTYOWNER`.

```
DDL INCLUDE pdb.*.* ALLOWEMPTYOWNER ALLOWEMPTYOBJECT
```

DDLERROR

Valid For

Extract and Replicat

Description

Use the `DDLERROR` parameter to handle DDL errors on the source and target systems. Options are available for Extract and Replicat.

DDLERROR for Extract

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 number_of_skips] [RETRYDELAY seconds] [SKIPTRIGGERERROR
number_of_errors]
```

`RESTARTSKIP` *`number_of_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 are 1 to 100000.
To write information about skipped operations to the Extract report file, use the `DDLOPTIONS` parameter with the `REPORT` option.

`SKIPTRIGGERERROR` *`number_of_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 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.

DDLERROR for Replicat

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

Default

Abend

Syntax

```
DDLERROR
{error | DEFAULT} {response}
{INCLUDE inclusion_clause | EXCLUDE exclusion_clause}
[IGNOREMISSINGOBJECTS | ABENDONMISSINGOBJECTS]
[RETRYDELAY seconds]
```

{*error* | DEFAULT} {*response*}

error
Specifies an explicit DDL error for this DDLERROR statement to handle.

DEFAULT
Specifies a default *response* to any DDL errors for which there is not an explicit DDLERROR statement.

response
The action taken by Replicat when a DDL error occurs. Can be one of the following:

ABEND
Roll back the operation and terminate processing abnormally. ABEND is the default.

DISCARD
Log the offending operation to the discard file but continue processing subsequent DDL.

IGNORE
Ignore the error.

{INCLUDE *inclusion_clause* | EXCLUDE *exclusion_clause*}
Identifies the beginning of an inclusion or exclusion clause that controls whether specific DDL is handled or not handled by the DDLERROR statement. See "DDL Filtering Options" for syntax and usage.

[IGNOREMISSINGOBJECTS | ABENDONMISSINGOBJECTS]
Controls whether or not Extract abends when DML is issued on objects that could not be found on the target. This condition typically occurs when DDL that is not in the replication configuration is issued directly on the target, or it can occur when there is a discrepancy between the source and target definitions.

IGNOREMISSINGOBJECTS
Causes Replicat to skip DML operations on missing tables.

ABENDONMISSINGOBJECTS
Causes Replicat to abend on DML operations on missing tables.

[RETRYDELAY *seconds*]
Specifies the delay in seconds between attempts to retry a failed operation. The default is 10 seconds.

Examples

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 1234 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 ABEND
```

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 1234 ABEND
DDLERROR 5678 IGNORE
```

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

```
DDLERROR 1234 ABEND INCLUDE OBJNAME tab*
DDLERROR 5678 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

Description

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 in the parameter file.

Default

See the argument descriptions

Syntax

```
DDLOPTIONS
[, DEFAULTUSERPASSWORD password [algorithm [ENCRYPTKEY DEFAULT | ENCRYPTKEY key_name]
```

```
[, CAPTUREGLOBALTEMPTABLE ]
[, DEFAULTUSERPASSWORDALIAS alias [DOMAIN domain] ]
[, EXCLUDETAG [tag | + | NULL]
[, IGNOREMAPPING [, INCLUDETAG tag | +]
[, MAPDERIVED | NOMAPDERIVED]
[, MAPSCHEMAS]
[, MAPSESSIONSCHEMA source_schema TARGET target_schema]
[, NLSLENGTHSEMANTICS CHAR | BYTE | DEFAULT]
[, NOAPPLYGLOBALTEMPTABLE]
[, NOTAG]
[, PASSWORD algorithm ENCRYPTKEY {key_name | DEFAULT}]
[, REMOVECOMMENTS {BEFORE | AFTER}]
[, REPLICATEPASSWORD | NOREPLICATEPASSWORD]
[, REPORT | NOREPORT]
[, UPDATEMETADATA]
[, USEPASSWORDVERIFIERLEVEL {10|11}]
[, _USEOWNERFORSESSION]
```

DEFAULTUSERPASSWORD *password* [*algorithm* ENCRYPTKEY {*key_name* | DEFAULT}]
Valid for Replicat. (Oracle only)
Can be used instead of the DEFAULTUSERPASSWORDALIAS option if an Oracle GoldenGate credential store is not being used. Specifies a different password for a replicated {CREATE | ALTER} USER *name* IDENTIFIED BY *password* statement from the one used in the source statement. Replicat will replace the placeholder that Extract writes to the trail with the specified password. When using DEFAULTUSERPASSWORD, use the NOREPLICATEPASSWORD option of DDLOPTIONS for Extract.
DEFAULTUSERPASSWORD *password* without options specifies a clear-text password. If the password is case-sensitive, type it that way.

> **Note:**
>
> Replication of CREATE | ALTER PROFILE will fail as the profile/password verification function must exist in the SYS schema. To replicate these DDLs successfully, password verification function must be created manually on both source/target(s) since DDL to SYS schema is excluded.

Use the following options if the password was encrypted with the ENCRYPT PASSWORD command:

algorithm
Specifies the encryption algorithm that was used to encrypt the password with the ENCRYPT PASSWORD command: AES128, or AES192, AES256.

ENCRYPTKEY *key_name*
Specifies the logical name of a user-created encryption key in the ENCKEYS lookup file. Use if ENCRYPT PASSWORD was used with the KEYNAME *key_name* option, and specify the same key name.

ENCRYPTKEY DEFAULT
Directs Oracle GoldenGate to use a random key. Use if ENCRYPT PASSWORD was used with the KEYNAME DEFAULT option.

CAPTUREGLOBALTEMPTABLE
Valid for Oracle

Allows Global Temporary Tables (GTT) DDLs to be visible to Extract so that they can be replicated. By default, GTT DDLs are not visible to Extract so using CAPTUREGLOBALTEMPTABLE you can set Extract to include GTT DDLs that then can be filtered by the DDL statement and if passed, written to the trail.
The GTT DDLs that are present in the trail will be filtered and executed by the Replicat if they pass the filtering criteria.
For trigger-version of Extract, this option is set to false regardless of whether the table is GTT or not.

DEFAULTUSERPASSWORDALIAS *alias* [DOMAIN *domain*]
Valid for Replicat. (Oracle only)
Can be used instead of the DEFAULTUSERPASSWORD option if an Oracle GoldenGate credential store is being used. Specifies the alias of a credential whose password replaces the one in the IDENTIFIED BY clause of a replicated CREATE USER or ALTER USER statement. The alias is resolved to the encrypted password in the Oracle GoldenGate credential store. Replicat replaces the placeholder that Extract writes to the trail with the resolved password before applying the DDL to the target.
When using DEFAULTUSERPASSWORDALIAS, use the NOREPLICATEPASSWORD option of DDLOPTIONS for Extract.

alias
Specifies the alias of the credential whose password will be used for the replacement password. This credential must exist in the Oracle GoldenGate credential store. If you are not sure what alias to use, you can inspect the content of the credential store by issuing the INFO CREDENTIALSTORE command.

DOMAIN *domain*
Specifies the domain that is assigned to the specified user in the credential store.

[EXCLUDETAG [*tag* | + | NULL]

> **Note:**
>
> Starting with Oracle GoldenGate 23ai, the syntax for using the EXCLUDETAG and INCLUDETAG option has been enhanced and would no longer use the GETREPLICATES, IGNOREREPLICATES, GETAPPLOPS, and IGNOREAPPLOPS options.

Use EXCLUDETAG *tag* to direct the Extract process to ignore the individual records that are tagged with the specified redo tag. Compare with older versions, new trail file contains tag tokens, which would not introduce problems for older trail readers.
Use EXCLUDETAG + to direct the Extract process to ignore the individual records that are tagged with any redo tag.
The EXCLUDETAG is used to exclude changes that were earlier tagged either by Replicat using the DBOPTIONS SET TAG option or within the Oracle database session using the dbms_xstream.set_tag procedure.

Example
The following are examples of how to use tag specifiers with EXCLUDETAG.

To exclude all tagged changes:

```
DDLOPTIONS EXCLUDETAG +
```

To exclude specific tagged changes:

```
DDLOPTIONS EXCLUDETAG 00
DDLOPTIONS EXCLUDETAG 0952
```

To have multiple exclude tags in a single DDLOPTIONS statement:

```
DDLOPTIONS EXCLUDETAG 00 EXCLUDETAG 97ab
```

INCLUDETAG [*tag* | +]
Valid for integrated Extract.
Use INCLUDETAG *tag* to include specific changes trail files. The tag value can be up to 2000 hexadecimal digits (0-9 A-F).
When specifying both EXCLUDETAG and INCLUDETAG parameters with the DDLOPTIONS command, the EXCLUDETAG should come first.
Example:
To include all tagged changes:

```
DDLOPTIONS INCLUDETAG +
```

To include specific tagged changes:

```
DDLOPTIONS INCLUDETAG 00
```

Considerations while using EXCLUDETAG and INCLUDETAG Parameters
While using EXCLUDETAG and INCLUDETAG parameters with TRANLOGOPTIONS and DDLOPTIONS commands, consider the following:

- If the TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG are specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are not specified, then the TRANLOGOPIIONS EXCLUDETAG/INCLUDETAG parameters apply to both DML and DDL operations.
- If the TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG options are specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are also specified, then the TRANLOGOPIIONS EXCLUDETAG/INCLUDETAG apply to DML operations, and the DDLOPTIONS EXCLUDETAG/INCLUDETAG apply to DDL operations.
- If the TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG are not specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are specified, then the DDLOPTIONS EXCLUDETAG/INCLUDETAG applies to DDL operations, and there is no tag filtering for DML operations.
- If TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG are not specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are also not specified, then the default option DDLOPTIONS EXCLUDETAG + is applicable, which excludes all tagged DDL operations.
- For DDLOPTIONS when specifying both EXCLUDETAG and INCLUDETAG, then EXCLUDETAG should come first.

IGNOREMAPPING
Valid for Replicat. Disables the evaluation of name mapping that determines whether DDL is of MAPPED or UNMAPPED scope. This option improves performance in like-to-like DDL replication configurations, where source and target schema names and object names match, and where mapping functions are therefore unnecessary. With IGNOREMAPPING enabled, MAPPED or

UNMAPPED scope cannot be determined, so all DDL statements are treated as OTHER scope. Do not use this parameter when source schemas and object names are mapped to different schema and object names on the target.

MAPDERIVED | NOMAPDERIVED
Valid for Replicat (Oracle). 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 How Oracle GoldenGate Handles Derived Object Names.

MAPSCHEMAS
Valid for Replicat (Oracle and Teradata). Use only when MAPSESSIONSCHEMA is used.

- MAPSESSIONSCHEMA establishes a source-target mapping for session schemas and is used for objects whose schemas are not qualified in the DDL.
- MAPSCHEMAS maps objects that do have qualified schemas in the source DDL, but which do not qualify for mapping with MAP, to the same session-schema mapping as in MAPSESSIONSCHEMA. Examples of such objects are the Oracle CREATE TABLE AS SELECT statement, which contains a derived object in the AS SELECT clause, or the Teradata CREATE REPLICATION RULESET statement.

This mapping takes place after the mapping that is specified in the MAP statement.
As an example, suppose the following DDL statement is issued on a source Oracle database:

```
create table a.t as select from b.t;
```

Suppose the MAP statement on the target is as follows:

```
MAP a.*, TARGET c.*;
DDLOPTIONS MAPSESSIONSCHEMA b, TARGET b1, MAPSCHEMAS
```

As a result of this mapping, Replicat issues the following DDL statement on the target:

```
create table c.t as select from b1.t;
```

- The base table gets mapped according to the TARGET clause (to schema c).
- The qualified derived object (table t in SELECT FROM) gets mapped according to MAPSESSIONSCHEMA (to schema b1) because MAPSCHEMAS is present.

Without MAPSCHEMAS, the derived object would get mapped to schema c (as specified in the TARGET clause), because MAPSESSIONSCHEMA alone only maps unqualified objects.

MAPSESSIONSCHEMA *source_schema* TARGET *target_schema*
Valid for Replicat (Oracle only). Enables a source session schema to be mapped to (transformed to) a different session schema on the target.

- *source_schema* is the session schema that is set with ALTER SESSION set CURRENT_SCHEMA on the source.

- *target_schema* is the session schema that is set with ALTER SESSION set CURRENT_SCHEMA on the target.

Wildcards are not supported. You can use multiple MAPSESSIONSCHEMA parameters to map different schemas.
MAPSESSIONSCHEMA overrides any mapping of schema names that is based on master or derived object names
See the example at the end of this topic for usage.
See also MAPSCHEMAS.

NLSLENGTHSEMANTICS CHAR | BYTE | DEFAULT]
Valid for Replicat (Oracle only).
Allows Replicat to override the NLS_LENGTH_SEMANTICS Oracle database session parameter setting that is part of the DDL replication trail file record. For example, if the DDL is run in a database session with NLS_LENGTH_SEMANTICS parameter value as BYTE, then by default, before Replicat applies the DDL, it will set the database session parameter NLS_LENGTH_SEMANTICS to BYTE. To override this, you can set NLSLENGTHSEMANTICS parameter to CHAR, and before the DDL is applied, the NLS_LENGTH_SEMANTICS session parameter is set to CHAR.

NOAPPLYGLOBALTEMPTABLE
Prevents Replicat from applying Global Temporary Tables (GTT). If this option is not specified, then Replicat will always apply DDLs to Global Temporary Tables (GTT) from the trail.

NOTAG
Valid for Replicat
Prevents the tagging of DDL that is applied by Replicat with a redo tag (either the default tag '00' or one set with the DBOPTIONS parameter with the SETTAG option).
The default tag value used by the Replicat is 00, and it can be manually changed by using the DBOPTIONS SETTAG parameter. See DBOPTIONS.

PASSWORD *algorithm* ENCRYPTKEY {*key_name* | DEFAULT}
Valid for Extract (Oracle only)
Directs Extract to encrypt all passwords in source DDL before writing the DDL to the trail.

algorithm
Specifies the encryption algorithm to be used to encrypt the password. Valid values are AES128, AES192, or AES256.

ENCRYPTKEY *key_name*
Specifies the logical name of a user-created encryption key in an ENCKEYS lookup file.

ENCRYPTKEY DEFAULT
Directs Oracle GoldenGate to use a random key.

REMOVECOMMENTS {BEFORE | AFTER}
(Optional) 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 "DDLSUBST" for more information.

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. This is the default behavior if `REMOVECOMMENTS` is not specified.

REPLICATEPASSWORD | NOREPLICATEPASSWORD
Valid for Extract (Oracle only). Applies to the password in a `{CREATE | ALTER} USER user IDENTIFIED BY password` command.

- By default (`REPLICATEPASSWORD`), Oracle GoldenGate uses the source password in the target `CREATE` or `ALTER` statement.
- To prevent the source password from being sent to the target, use `NOREPLICATEPASSWORD`.

When using `NOREPLICATEPASSWORD`, specify a password for the target DDL statement by using a `DDLOPTIONS` statement with the `DEFAULTUSERPASSWORD` or `DEFAULTUSERPASSWORDALIAS` option in the Replicat parameter file.

REPORT | NOREPORT
Valid for Extract and Replicat (Oracle and Teradata). Controls whether or not expanded DDL processing information is written to the report file. The default of `NOREPORT` reports basic DDL statistics. `REPORT` adds the parameters being used and a step-by-step history of the operations that were processed.

UPDATEMETADATA
Valid for Replicat (Oracle only). Use in an active-active bi-directional configuration. This parameter notifies Replicat on the system where DDL originated that this DDL was propagated to the other system, and that Replicat should now update its object metadata cache to match the new metadata. This keeps Replicat's metadata cache synchronized with the current metadata of the local database.

USEPASSWORDVERIFIERLEVEL {10|11}
Only valid in an Oracle to Oracle configuration. Checks if the password verifier being sent in a DDL `CREATE USER` statement requires modifying. The reason for this check is because Oracle has different password verifiers, depending on the database version:

- 10g: A weak verifier kept in `user$.password`.
- 11g: The SHA-1 verifier.
- 12c: The SHA-2 and HTTP digest verifiers.

The SHA-1, SHA-2 and HTTP verifiers are captured in `user$.spare4` in the format of: `'S:<SHA-1-verifier>;H:<http-verifier>;T:<SHA-2-verifier>'`. Integrated Extract returns the following DDL in 12c for create user DDL statements:

- In 12.0.1.0 it returns: `CREATE USER username IDENTIFIED BY VALUES 'S:SHA-1;H:http;weak'`.
- In 12.0.2.0 and later it returns: `CREATE USER username IDENTIFIED BY VALUES 'S:SHA-1;H:http;T:SHA-2;weak'`.

If Replicat runs against Oracle 12*c*, these forms of `CREATE USER` are handled at the RDBMS level, but if Replicat runs against Oracle 10*g* or 11, these forms are not handled by the RDBMS. Oracle 10*g* only accepts the weak verifier, whereas Oracle 11*g* only accepts the `S:SHA-1` and weak verifiers.
To allow the `CREATE USER` DDL generated for an Extract connected to Oracle 12*c* to work with a Replicat connected to Oracle 10*g* or 11*g*, this parameter can be used to filter out the unwanted verifiers, as follows:

- If USEPASSWORDVERIFIERLEVEL is set to 10, everything except the weak verifier is filtered out of the CREATE USER DDL verification string.
- If USEPASSWORDVERIFIERLEVEL is set to 11, everything except the S:*SHA-1* and weak verifiers is filtered out of the CREATE USER DDL verification string.

Examples

Example 1
The following shows how MAPSESSIONSCHEMA works to allow mapping of a source session schema to another schema on the target.
Assume the following DDL capture and mapping configurations in Extract and Replicat:
Extract:

```
DDL INCLUDE OBJNAME SRC.* INCLUDE OBJNAME SRC1.*
TABLE SRC.*;
TABLE SRC1.*;
DDL INCLUDE OBJNAME SRC.* INCLUDE OBJNAME SRC1.*
TABLE SRC.*;
TABLE SRC1.*;
```

Replicat:

```
DDLOPTIONS MAPSESSIONSCHEMA SRC TARGET DST
DDLOPTIONS MAPSESSIONSCHEMA SRC1 TARGET DST1
MAP SRC.*, TARGET DST.*;
MAP SRC1.*, TARGET DST1.*;
DDL INCLUDE OBJNAME DST.* INCLUDE OBJNAME DST1.*
```

Assume that the following DDL statements are issued by the logged-in user on the source:

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

Replicat will perform the DDL as follows (explanations precede each code segment):

```
-- Set session to DST, because SRC.* is mapped to DST.* in MAP statement.
ALTER SESION SET CURRENT_SCHEMA=DST;
-- Create the first TAB table in the DST schema, using the DST session schema.
CREATE TABLE DST.tab (X NUMBER);
-- Restore Replicat schema.
ALTER SESSION SET CURRENT_SCHEMA=REPUSER
-- Set session schema to DST, per MAPSESSIONSCHEMA, so that AS SELECT succeeds.
ALTER SESION SET CURRENT_SCHEMA=DST;
-- Create the DST1.TAB table AS SELECT * FROM the first table (DST.TAB).
CREATE TABLE DST1.tab (X NUMBER) AS SELECT * FROM tab;
-- Restore Replicat schema.
ALTER SESSION SET CURRENT_SCHEMA=REPUSER
```

Without MAPSESSIONSCHEMA, the SELECT * FROM TAB would attempt to select from a non-existent SRC.TAB table and fail. The default is to apply the source schema to unqualified objects in a target DDL statement. The DDL statement in that case would look as follows and would fail:

```
-- Set session to DST, because SRC.* is mapped to DST.* in MAP statement.
ALTER SESION SET CURRENT_SCHEMA=DST;
-- Create the first TAB table in the DST schema, using the DST session schema.
CREATE TABLE DST.tab (X NUMBER);
-- Restore Replicat schema.
```

```
ALTER SESSION SET CURRENT_SCHEMA=REPUSER
-- Set session schema to SRC, because TAB in the AS SELECT is unqualified-- and SRC is
the source session schema.
ALTER SESION SET CURRENT_SCHEMA=SRC;
-- Create DST1.TAB AS SELECT * from SRC.TAB (SRC=current session schema).
CREATE TABLE DST1.tab (X NUMBER) AS SELECT * FROM tab;
-- SRC.TAB does not exist.
-- Abend with an error unless the error is handled by a DDLERROR statement.
```

Example 2
The following shows how to use `DEFAULTUSERPASSWORDALIAS` to specify a different password for a replicated `{CREATE | ALTER} USER` *`name`* `IDENTIFIED BY` *`password`* statement from the one used in the source statement. In this example, the alias `ddlalias` is in the `target` domain in the credential store.

```
DDLOPTIONS DEFAULTUSERPASSWORDALIAS ddlalias DOMAIN target
```

DDLSUBST

Valid For

Extract and Replicat

Description

Use the `DDLSUBST` parameter to substitute strings in a DDL operation. For example, you could substitute one table name for another or substitute a string within comments. The search is not case-sensitive. To represent a quotation mark in a string, use a double quote mark.

Guidelines for Using DDLSUBST

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

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

  In this example, no substitution occurs because the objects in the `INCLUDE` and `DDLSUBST` statements are different. The `fin`-owned objects are included in the Oracle GoldenGate DDL configuration, but the `sales`-owned objects are not.
- You can use multiple `DDLSUBST` parameters. They execute in the order listed in the parameter file.
- For Oracle DDL that includes comments, do not use the `DDLOPTIONS` parameter with the `REMOVECOMMENTS BEFORE` option if you will be doing string substitution on those comments. `REMOVECOMMENTS BEFORE` removes comments before string substitution occurs. To remove comments, but allow string substitution, use the `REMOVECOMMENTS AFTER` option.

- There is no maximum string size for substitutions, other than the limit that is imposed by the database. If the string size exceeds the database limit, the Extract or Replicat process that is executing the operation abends.

Default

No substitution

Syntax

```
DDLSUBST 'search_string' WITH 'replace_string'
[INCLUDE inclusion_clause | EXCLUDE exclusion_clause]
```

'search_string'
The string in the source DDL statement that you want to replace. Enclose the string within single quote marks. To represent a quotation mark in a string, use a double quotation mark.

WITH
Required keyword.

'replace_string'
The string that you want to use as the replacement in the target DDL. Enclose the string within single quote marks. To represent a quotation mark in a string, use a double quotation mark.

INCLUDE *inclusion_clause* | EXCLUDE *exclusion_clause*
Specifies one or more `INCLUDE` and `EXCLUDE` statements to filter the DDL operations for which the string substitution rules are applied. See "DDL Filtering Options" for syntax and usage.

Examples

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 2
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 3
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

DB2 z/OS

Description

This is a GLOBALS parameter. Use the DDLTABLE parameter to specify the name of the DDL history table, if other than the default of GGS_DDL_HIST. The DDL history table stores a history of DDL operations processed by Oracle GoldenGate.
In DB2 z/OS, an acceptable value is a valid DB2 z/OS table name.

Default

GGS_DDL_HIST

Syntax

DDLTABLE *table_name*

table_name
The fully qualified name of the DDL history table. This can be a two-part name (*schema.table*) or a three-part name, if stored in a container database (*container.schema.table*).

Example

```
DDLTABLE GG_DDL_HISTORY
```

DECRYPTTRAIL

Valid For

Replicat

Description

Use the DECRYPTTRAIL parameter to decrypt data in a trail or Extract file. This parameter is required in the following cases:

- If the trail was encrypted with the encryption profile method, use DECRYPTTRAIL to decrypt trail on the Replicat side.
- If the trail was encrypted with the ENCKEYS method for the HP NonStop platform, use the DECRYPTTRAIL for Replicat to decrypt the data before applying it to the target.

Data encryption is controlled by the ENCRYPTTRAIL | NOENCRYPTTRAIL parameters.

For Oracle, if you are using wallet based encryption DECRYPTTRAIL does not require a cipher because it is recorded in the trail file header.

Default

None

Syntax

```
DECRYPTTRAIL ({AES128 | AES192 | AES256}, KEYNAME name)
```

DECRYPTTRAIL
Valid without any other options only if the trail or file was encrypted with ENCRYPTTRAIL without options to use 256-key byte substitution.

{AES128 | AES192 | AES256}
Valid for **Encryption Profile** and **ENCKEYS** methods.
When using the Encryption Profile method, then you need to create an encryption profile in advance and associate it with the Replicat.
When using the ENCKEYS method, an ENCKEYS file that has the master key must exist in the *deployment_home*/etc/conf/ogg directory.

KEYNAME *name*
The KEYNAME option is only required when the trail files from HP NonStop systems are encrypted using the ENCKEY option. See DECRYPTTRAIL in *Reference Guide for Oracle GoldenGate for HP NonStop (Guardian)*

Example

Example 1
The following is an example of the encryption profile method.

```
DECRYPTTRAIL AES192
```

Example 2
The following decrypts using the ENCKEYS method.

```
DECRYPTTRAIL AES192, KEYNAME mykey1
```

DEFERAPPLYINTERVAL

Valid For

Replicat

Description

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.

If you want to stop a process (like the Replicat) at a specific time, use the END parameter.

> **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
```

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 HOURS
```

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

Description

Use the DEFSFILE parameter to identify the name of the file to which DEFGEN will write data definitions. By default, the data definitions file is written in the character set of the local operating system. You can change the character set with the CHARSET option.

Default

None

Syntax

```
DEFSFILE file_name [APPEND | PURGE] [CHARSET character_set] [FORMAT RELEASE major.minor]
```

file_name
The relative or fully qualified file name. The file is created when you run DEFGEN.

APPEND
Directs DEFGEN to write new content (from the current run) at the end of any existing content, if the specified file already exists. If the definitions file already exists, but is of an older Oracle GoldenGate release version, you can set the FORMAT RELEASE option to the same version as the existing file to prevent errors. Otherwise, DEFGEN will try to add newer metadata features and abend. The following are the restrictions when using APPEND:

- If the existing data definitions file is in a format older than Oracle GoldenGate 11.2.1, DEFGEN appends the table definitions in the old format, where table and column names with multi-byte and special characters are not supported.
- If the existing data definitions file is in the newer format introduced in version 11.2.1, DEFGEN appends the table definitions in the existing character set of the file.
- If the existing file is from version 11.2 or earlier, it was written when DEFGEN did not support three-part object names and will cause an error if the new metadata contains three-part names. You can specify objects from an Oracle container database if you remove the container or catalog portion by using the NOCATALOG parameter in the DEFGEN parameter file.

PURGE
Directs DEFGEN to purge the specified file before writing new content from the current run. When using PURGE, you can overwrite an existing definitions file that was created by an older version of DEFGEN with newer metadata that supports newer features, such as three-part object names.

CHARSET *character_set*
Generates the definitions file in the specified character set. Without CHARSET, the default character set of the operating system is used. If APPEND mode is specified for a definitions file that is version 11.2.1 or later, CHARSET is ignored, and the character set of the existing definitions file is used.

FORMAT RELEASE *major.minor*
Specifies the metadata format of the definitions that are sent by DEFGEN to the definitions file. The metadata tells the reader process whether the file records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones. Use FORMAT when the definitions file will be used by a process that is of an older Oracle GoldenGate version than the current one.

- FORMAT is a required keyword.
- RELEASE specifies an Oracle GoldenGate release version. *major* is the major version number, and *minor* is the minor version number. The X.x must reflect a current or earlier, generally available (GA) release of Oracle GoldenGate. Valid values are 9.0 through the current Oracle GoldenGate X.x version number, for example 11.2 or 12.1. (If you use an Oracle GoldenGate version that is earlier than 9.0, specify either 9.0 or 9.5.)

  The release version is programmatically mapped back to an appropriate internal compatibility level. The default is the current version of the process that writes to this trail. Note that RELEASE versions earlier than 12.1 do not support three-part object names.

FORMAT RELEASE *major.minor*
Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones.

`FORMAT RELEASE` specifies an Oracle GoldenGate release version. *`major`* is the major version number, and *`minor`* is the minor version number. The X.x must reflect a current or earlier, generally available (GA) release of Oracle GoldenGate. Valid values are 11.1 through the current Oracle GoldenGate X.x version number, for example 11.2 or 12.1. The release version is programmatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail.

> **Note:**
>
> `RELEASE` versions earlier than 12.1 do not support three-part object names.

> **Note:**
>
> If using multiple trails in a single Extract, only `RELEASE` versions that are the same can coexist.

The following settings are supported for Oracle Database 12.2 and higher:

- For Oracle Database 12.2 non-CDB or higher with compatibility set to 12.1, `FORMAT RELEASE` 12.2 or above is supported.
- For Oracle Database 12.2 non-CDB or higher with compatibility set to 12.2, `FORMAT RELEASE` 12.2 or above is supported.
- For Oracle Database 12.2 CDB/PDB or higher with compatibility set to 12.2, only `FORMAT RELEASE` values 12.3 or higher are supported. This is due to the use of local undo for PDBs, which requires augmenting the transaction ID with the PDB number to ensure uniqueness of trx IDs.

Example

```
DEFSFILE ./dirdef/orcldef CHARSET ISO-8859-11 FORMAT RELEASE 11.2
```

DIAGLOGRECS

Valid For

Valid for DB2 z/OS.

Description

This parameter enables a diagnostic dump of all log records read by the Extract. This implies that the parameter, if set to `true`, will produce the log record trace, independent of the activity trace.

The log record trace is used to determine issues with log records that the Extract cannot process and other issues like sequence errors or changes in the log.

Syntax

```
DIAGLOGRECS true | false
```

The parameter provides the `true` or `false` option. The default value is `false`.

DICTIONARY_CACHE_SIZE

Valid For

Valid for Extract.

Description

The `dictionary_cache_size` (default 5000) parameter sets the LogMiner dictionary cache size.

The cache size can become a limiting factor during heavy DDL workloads causing LogMiner builder to become CPU-bound and messages in trace file indicating dictionary objects being reloaded.

This example explains such a situation:

```
krvrdgcidi_GetChunkIdInt: loaded obj 137584, scn 0x000000001f65a041, beg scn
0x000000001c16a034, end scn 0xffffffffffffffff, tsn 6, SOESMALLTS, chunk 0
```

DISCARDFILE | NODISCARDFILE

Valid For

Extract and Replicat

Description

Use the `DISCARDFILE` parameter to do the following:

- Customize the name, location, size, and write mode of the discard file. By default, a discard file is generated whenever a process is started with the `START` command. To retain the default properties, a `DISCARDFILE` parameter is not required.
- Specify the use of a discard file for processing methods where the process starts from the command line of the operating system and a discard file is not created by default.

Use the `NODISCARDFILE` parameter to disable the use of a discard file. If `NODISCARDFILE` is used with `DISCARDFILE`, the process abends.

When using `DISCARDFILE`, use either the `PURGE` or `APPEND` option. Otherwise, you must specify a different discard file name before starting each process run, because Oracle GoldenGate will not write to an existing discard file without one of these instructions and will terminate.

See "DISCARDROLLOVER" for how to control how often the discard file is rolled over to a new file.

For more information about the discard file, see Overview of Oracle GoldenGate Error Handling.

Default

If a process is started with the `START` command, it generates a discard file as follows:

- The file is named after the process that creates it, with a `.dsc` extension. If the process is a coordinated Replicat, it generates one file per thread. Each file name is appended with the thread ID of the corresponding thread.

- The file is created in the `dirrpt` sub-directory of the Oracle GoldenGate installation directory.
- The maximum file size is 50 MB.
- At startup, if a discard file exists, it is purged before new data is written.
- The maximum filename is 250 characters including the directory.

When you start a process from the command line of the operating system, you should not generate a discard file by default.

Syntax

```
DISCARDFILE { [file_name]
[, APPEND | PURGE]
[, MAXBYTES n | MEGABYTES n] } |
NODISCARDFILE
```

DISCARDFILE
Indicates that the name or other attribute of the discard file is being changed.

file_name
The relative or fully qualified name of the discard file, including the actual file name. For a coordinated Replicat, specify a file name of up to five characters, because each file name is appended with the thread ID of the thread that writes it. To store the file in the Oracle GoldenGate directory, a relative path name is sufficient, because Oracle GoldenGate qualifies the name with the Oracle GoldenGate installation directory.

APPEND
Adds new content to existing content if the file already exists. If neither `APPEND` nor `PURGE` is used, you must specify a different discard file name before starting each process run.

PURGE
Purges the file before writing new content. If neither `PURGE` nor `APPEND` is used, you must specify a different discard file name before starting each process run.

MAXBYTES *n*
Sets the maximum size of the file in bytes. The valid range is from 1 to 4096967295. The default is 3000000. If the specified size is exceeded, the process abends.

MEGABYTES *n*
Sets the maximum size of the file in megabytes. The valid range is from 1 to 4096. The default is 3. If the specified size is exceeded, the process abends.

NODISCARDFILE
Prevents the process from creating a discard file.

Example

Example 1
This example specifies a non-default file name and extension, non-default write mode, and non-default maximum file size. This example shows how you could change the default properties of a discard file for an online process or specify the use of a discard file for a process that starts from the command line of the operating system and has no discard file by default.

```
DISCARDFILE .dirrpt/discard.txt, APPEND, MEGABYTES 20
```

Example 2
This example changes only the write mode of the default discard file for an online process.

```
DISCARDFILE .dirrpt/finance.dsc, APPEND
```

Example 3
This example disables the use of a discard file for an online process.

```
NODISCARDFILE
```

DISCARDROLLOVER

Valid For

Extract and Replicat

Description

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 *GROUPn.extension*, where:

- *GROUP* 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.
- *extension* is the file extension, such as .dsc.

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.

Discard files roll over at the start of a process run. However, if APPEND or PURGE is specified in DISCARDFILE parameter, then discard files don't roll over at the start of a process.

If the NODISCARDFILE parameter is used with the DISCARDROLLOVER parameter, the process abends.

For more information about the discard file, see Overview of Oracle GoldenGate Error Handling.

Default

Disabled. By default, discard files are rolled over when a process starts.

Syntax

```
DISCARDROLLOVER
{AT hh:mi |
ON day |
AT hh:mm ON day}
```

AT *hh:mi*
The time of day to age the file.
Valid values:

- *hh* is an hour of the day from 00 through 23.
- *mm* is minutes 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.

Examples

Example 1

```
DISCARDROLLOVER AT 05:30
```

Example 2

```
DISCARDROLLOVER ON friday
```

Example 3

```
DISCARDROLLOVER AT 05:30 ON friday
```

DYNAMICRESOLUTION

Valid For

Extract and Replicat

Description

Use the `DYNAMICRESOLUTION` parameter to control how table names are resolved.

`DYNAMICRESOLUTION`, the default, enables fast process startup when there are numerous tables specified in `TABLE` or `MAP` statements. To get metadata for transaction records that it needs to process, Oracle GoldenGate queries the database and then builds a record of the tables that are involved. `DYNAMICRESOLUTION` causes the record to be built one table at a time, instead of all at once. The metadata of any given table is added when Extract first encounters the object ID in the transaction log, while record-building for other tables is deferred until their object IDs are encountered. `DYNAMICRESOLUTION` is the same as `WILDCARDRESOLVE DYNAMIC`.

See "WILDCARDRESOLVE" for more information.

Default

```
DYNAMICRESOLUTION
```

Syntax

```
DYNAMICRESOLUTION
```

EBCDICTOASCII

Valid For

Extract data pump and Replicat

Description

Use the `EBCDICTOASCII` parameter to convert character data in the input trail from EBCDIC to ASCII format when sending it to a DB2 target database on a z/OS system. This parameter can be specified to request conversion of all EBCDIC columns and user token data to ASCII. This parameter must precede the `SOURCEDB` parameter. This parameter is only needed if the input trail file was created by an Extract version prior to v10.0. It is ignored for all other cases, because the conversion is done automatically.

This parameter should be used in the `TRANLOG` Extract. It is not valid for Extract data pumps.

Default

None

Syntax

```
EBCDICTOASCII
```

ENABLEMONITORING

Valid For

GLOBALS

Description

Use the `ENABLEMONITORING` parameter to enable the monitoring of Oracle GoldenGate instances from Oracle GoldenGate Monitor and collect trend data for Performance Metrics Service.

Performance Metrics Server is used to monitor processes or services and collect statistics. Starting with Oracle GoldenGate 21c, Unix Domain Sockets (UDS) is used to communicate with the local Performance Metrics Service.
UDS is the default mode in Performance Metrics Service. This feature is applicable for Oracle GoldenGate Microservices Architecture for Oracle and non-Oracle databases.
For operating systems such as Windows and others, which do not support UDS, UDP would continue to be used. If you are working on operating systems that don't support UDS, you need to set the `ENABLEMONITORING UDP` parameter before starting the server in legacy, to bring up the `PMSRVR GLOBALS` parameter. For Oracle GoldenGate MA it is done by default. For details, see Protocols for Performance Monitoring for Different Operating Systems in *Oracle GoldenGate Microservices Architecture*.

> **Note:**
>
> When monitoring is enabled on a UNIX system for a high number of Oracle GoldenGate processes (approximately 400), the system-imposed limit on the maximum amount of allowed shared memory may be exceeded. The message returned by Manager is similar to this:
>
> ```
> WARNING OGG-01934  Datastore repair failed" reported during "start...
> ```
>
> If this occurs, increase the kernel parameter `kernel.shmall` by eight times the default for the operating system.

Default

Disabled

Syntax

```
ENABLEMONITORING
    [ UDP ]
    [ UDPPORT portnumber ]
    [ HTTPPORT portnumber ]
```

`UDPPORT` *`portnumber`*
Valid with `UDP` for monitoring with a Performance Metrics Server (`PMSRVR`) on Windows and other operating systems that don't support UDS.
The UDP listening port. It is optional. If provided, it overrides the existing `GLOBALS` parameter,`REPOUDPPORT`. If not provided, it uses the value of `REPOUDPPORT` as the port number. You can change the UDP port of a `PMSRVR` in a secure deployment by adding the `repoUDPPORT` parameter to the `GLOBALS` file. For more information on configuring the UDP and TCP ports for PMSRVR, see Add a Deployment.

`HTTPPORT` *`portnumber`*
Valid with `UDP` for monitoring with a Performance Metrics Server. Not valid for the BDB or LMDB monitoring modes.
The HTTP listening port for the service. It is optional. If not provided, 9004 is the default port number.

ENABLE_HEARTBEAT_TABLE | DISABLE_HEARTBEAT_TABLE

Valid For

Extract, Replicat, and GLOBALS

Description

The `ENABLE_HEARTBEAT_TABLE` and `DISABLE_HEARTBEAT_TABLE` commands specify whether the Oracle GoldenGate process will be handling records from `GG_HEARTBEAT` table or not. When specified as a `GLOBALS`, it is true for the entire installation unless overridden by a specific process.

Default

ENABLE_HEARTBEAT_TABLE

Syntax

ENABLE_HEARTBEAT_TABLE | DISABLE_HEARTBEAT_TABLE

ENABLE_HEARTBEAT_TABLE
Enables Oracle GoldenGate processes to handle records from a GG_HEARTBEAT table. This is the default.

DISABLE_HEARTBEAT_TABLE
Disables Oracle GoldenGate processes from handing records from a GG_HEARTBEAT table

ENCRYPTTRAIL | NOENCRYPTTRAIL

Valid For

Extract

Description

The encryption profile must be set up before using the ENCRYPTTRAIL parameter for encrypting trail files. The encryption profile contains information about the ***location of the master key*** and how Oracle GoldenGate will use it. See What is an Encryption Profile? in *Oracle GoldenGate Microservices Documentation* to know more.

Use the ENCRYPTTRAIL and NOENCRYPTTRAIL parameters to control whether Oracle GoldenGate encrypts or does not encrypt trail data that is written to a trail or Extract file.

Use the EXTTRAIL parameter in your Extract parameter file for encrypting trails.

> **Note:**
>
> When using the ENCRYPTRAIL parameter with the EXTTRAIL parameter, ensure that the ENCRYPTRAIL parameter is mentioned before EXTTRAIL, else the trail will not be encrypted.

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.

ENCRYPTTRAIL and NOENCRYPTTRAIL cannot be used when FORMATASCII is used to write data to a file in ASCII format. The trail file must be written in the default Oracle GoldenGate canonical format.

ENCRYPTTRAIL encrypts the trail data across all data links and within the files themselves. Only the data blocks are encrypted. User tokens are not encrypted.

Default

NOENCRYPTTRAIL

Syntax

```
ENCRYPTTRAIL (AES)
```

ENCRYPTTRAIL (*AES*)
ENCRYPTTRAIL without options specifies 256-key byte substitution AES256 as the default for all database types except the NonStop platform because Advanced Encryption Standard (AES) encryption is not supported on that platform.
It's mandatory to provide a value for the ENCRYPTTRAIL parameter, otherwise Extract will abend.
ENCRYPTTRAIL supports **AES 128, AES 192, AES 256 (Master key and wallet method)** encryption. Use the master key based on the encryption profile. AES includes encryption key length to use. This is a symmetric-key encryption standard that is used by governments and other organizations that require a high degree of data security.

- AES128 has a 128-bit block size with a key size of 128 bits.
- AES192 has a 192-bit block size with a key size of 192 bits.
- AES256 has a 256-bit block size with a key size of 256 bits.

To use AES encryption for any database other than Oracle on a 32-bit platform, the path of the /lib sub-directory of the Oracle GoldenGate installation directory must be specified as an environment variable before starting any processes. This is not required on 64-bit platforms. Set the path as follows:

- Linux: Specify the path as an entry to the LD_LIBRARY_PATHvariable. For example:

  ```
  setenv LD_LIBRARY_PATH ./lib:$LD_LIBRARY_PATH
  ```

- For Solaris: Specify the path as an entry to the SHLIB_PATH variable.
- For IBMi and AIX: Specify the path as an entry to the LIBPATH variable.
- For Windows: Add the path to the PATH variable.

You can use the SETENV parameter to set it as a session variable for the process.

NOENCRYPTTRAIL
Prevents the trail from being encrypted. This is the default.

Examples

Example 1
In the following example, the master key and wallet method is used. The Extract process writes to two trails. The data for the emp table is written to trail /home/ggsora/dirdat/em, which is encrypted with the AES-192 cipher. The data for the stores table is written to trail /home/ggsora/dirdat/st, which is not encrypted.

```
ENCRYPTTRAIL AES192
EXTTRAIL /home/ggsora/dirdat/em
TABLE hr.emp;
NOENCRYPTTRAIL
EXTTRAIL /home/ggsora/dirdat/st
TABLE ops.stores;
```

Example 2
As an alternative to the preceding example, you can omit `NOENCRYPTTRAIL` if you list all non-encrypted trails before the `ENCRYPTTRAIL` parameter.

```
EXTTRAIL /home/ggsora/dirdat/st
TABLE ops.stores;
ENCRYPTTRAIL AES192
EXTTRAIL /home/ggsora/dirdat/em
TABLE hr.emp;
```

END

Valid For

Replicat

Description

Use the `END` parameter to terminate Replicat when it encounters the first record in the data source whose timestamp is the specified point in time.

You can only use `END` with the `SPECIALRUN` parameter. This parameter allows you to post data as a point-in-time snapshot, rather than continuously updating the target tables.

Without `END`, the process runs continuously until:

- the end of the trail is reached, at which point it will stop gracefully.
- manually terminated from the command shell.

Default

Continuous processing

Syntax

```
END {date [time] | RUNTIME}
```

date [time]
Causes Replicat to terminate when it reaches a record in the data source whose timestamp exceeds the one that is specified with this parameter.
Valid values:

- *`date`* is a date in the format of *`yyyy-mm-dd`*.
- *`time`* is the time in the format of *`hh:mi[:ss[.cccccc]]`* based on a 24-hour clock.

`RUNTIME`
Causes Replicat to terminate when it reaches a record in the data source whose timestamp exceeds the current date and clock time. All unprocessed records with timestamps up to this point in time are processed. One advantage of using `RUNTIME` is that you do not have to alter the parameter file to change dates and times from run to run. Instead, you can control the process start time within your batch programming.

Examples

Example 1

```
SPECIALRUN
END 2010-12-31 17:00:00
```

Example 2

```
SPECIALRUN
END RUNTIME
```

EOFDELAY | EOFDELAYCSECS

Valid For

Extract and Replicat

Description

Use the `EOFDELAY` or `EOFDELAYCSECS` parameter to control how often Extract 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. This parameter cannot be set to zero (0).

Default

The minimum is 1 second (1 second or 100 centiseconds ; the maximum is 60 seconds (60 seconds or 6000 centiseconds). It can be set to 1 (which is 1 centisecond) but should never be set to below 3.

Syntax

```
EOFDELAY seconds | EOFDELAYCSECS centiseconds
```

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
```

EXCLUDEHIDDENCOLUMNS

Valid For

Oracle Integrated Extract Capture; It's not valid for data pump.

Description

The parameter disables all the Oracle hidden columns including the timestamp columns created using automatic CDR. The parameter requires Oracle GoldenGate 12c (12.2.01) format trail or higher and must not specify the `NO_OBJECTDEFES` parameter. The `userexit` callback structure has the hidden column attributes and callback structure version is 5. You can specify the parameter at any location of the parameter file, as long as it is after the `EXTRACT group` parameter.

Syntax

```
EXTRACT ext1...
EXCLUDEHIDDENCOLUMNS
EXTTRAIL ./dirdat/a1
TABLE src.tab1;
```

EXCLUDETAG

Valid For

(Oracle) Extract and Replicat

(All databases) Extract or Replicat

Description

Use `EXCLUDETAG` *`tag`* in your Extract or Replicat parameter file to specify changes to be excluded from trail files. The limitation for this parameter is that the tag value can be up to 2000 hexadecimal digits (0-9A-F) or the plus sign (`+`). You can have multiple `EXCLUDETAG` lines, but each `EXCLUDETAG` should have a single value. By default, Replicat the individual records every change it applies to the database by `00` in both classic mode or integrated mode. Compared with older versions, new trail file contains tag tokens, which would not introduce problems for older trail readers.

Use `EXCLUDETAG +` to ignore the individual records that are tagged with any redo tag.

Do not use `NULL` with `tag` or `+` because it operates in conflict resulting in errors.

To tag the individual records, use the `DBOPTIONS` parameter with the `SETTAG` option in the Replicat parameter file. Use these parameters to prevent cycling (loop-back) of Replicat the individual records in a bi-directional configuration or to filter other transactions from capture. The default `SETTAG` value is 00 and this is the tag that Replicat uses when applying transactions to the target Oracle database.

Valid value is any single Oracle Streams tag. A tag value can be up to 2000 hexadecimal digits (0-9 A-F) long.

> **Note:**
>
> These parameters should be used instead of `EXCLUDEUSER` or `TRACETABLE` when possible.

Default

None

Syntax

```
[EXCLUDETAG [tag | NULL] | [+]
```

Example 1

For Replicat:

```
excludetag tag
```

EXCLUDEWILDCARDOBJECTSONLY

Valid For

GLOBALS

Description

Use the `EXCLUDEWILDCARDOBJECTSONLY` parameter to force the inclusion of non-wildcarded source objects specified in `TABLE` or `MAP` parameters when an exclusion parameter contains a wildcard that otherwise would exclude that object. Exclusion parameters are `CATALOGEXCLUDE`, `SCHEMAEXCLUDE`, `MAPEXCLUDE`, and `TABLEEXCLUDE`.

The exclusion parameters get evaluated and satisfied before the `TABLE` or `MAP` statements. Without `EXCLUDEWILDCARDOBJECTSONLY`, it would be possible for an object in a `TABLE` or `MAP` statement to be wrongly excluded because it satisfies the wildcard in the exclude specification. For `EXCLUDEWILDCARDOBJECTSONLY` to work on an object, that object must be explicitly named without using wildcards in any of the name components.

Default

None

Syntax

```
EXCLUDEWILDCARDOBJECTSONLY
```

Example

In this example, `schema1.src_table1` is included in processing because the `TABLEEXCLUDE` parameter is wildcarded and the `TABLE` specification is not wildcarded. Without `EXCLUDEWILDCARDOBJECTSONLY`, `schema1.src_table1` would be excluded because of the wildcard specification in `TABLEEXCLUDE`.

```
TABLEEXCLUDE schema1.src_table*;
    TABLE schema1.src_table1;
```

EXTFILE

Valid For

Extract and Replicat

Description

Use the EXTFILE parameter to specify an extract file on the local system that will be created by an initial load Extract and read by an initial load Replicat when SPECIALRUN is used.

Use this parameter for initial load configurations. For online change synchronization, use the EXTTRAIL parameter.

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

Replicat only supports the *file_name* value and no options.

You can encrypt the data in this file by using the ENCRYPTTRAIL parameter. See "ENCRYPTTRAIL | NOENCRYPTTRAIL" for more information.

Default

None

Syntax

```
EXTFILE file_name
[ APPEND]
[, PURGE]
[, FORMAT RELEASE major.minor]
[, MEGABYTES megabytes]
[, OBJECTDEFS | NO_OBJECTDEFS]
[, TRAILBYTEORDER {BIGENDIAN | LITTLEENDIAN | NATIVEENDIAN}]
```

file_name
The relative or fully qualified name of the trail. Use only two characters for the trail name. As trail files are aged, a six-character sequence number will be added to this name, for example /ogg/dirdat/ef000001. If using FORMAT RELEASE 12.2 or earlier, the trail file created is a static file that does not increment, and the naming convention is not limited to two characters.

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.

FORMAT RELEASE *major.minor*
Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones.
FORMAT RELEASE specifies an Oracle GoldenGate release version. *major* is the major version number, and *minor* is the minor version number. The X.x must reflect a current or earlier,

generally available (GA) release of Oracle GoldenGate. Valid values are 12.2 through the current Oracle GoldenGate X.x version number, for example 19.1. The release version is programmatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail.

> **Note:**
>
> `RELEASE` versions earlier than 12.2 do not support three-part object names.

> **Note:**
>
> If using multiple trails in a single Extract, only `RELEASE` versions that are the same can coexist.

The following settings are supported for Oracle Database 12.2 and higher:

- For Oracle Database 12.2 non-CDB or higher with compatibility set to 12.1 or higher, `FORMAT RELEASE` 12.2 or above is supported, due to the larger SCN value.
- For Oracle Database 12.2 CDB/PDB or higher with compatibility set to 12.2 or higher, only `FORMAT RELEASE` values 12.3 or higher are supported. This is due to the use of local undo for PDBs, which requires augmenting the transaction ID with the PDB number to ensure uniqueness of trx IDs.

`MEGABYTES` *`megabytes`*
The maximum size, in megabytes, of a file in the trail. The default is 2000.

`OBJECTDEFS | NO_OBJECTDEFS`
Use the `OBJECTDEFS` and `NO_OBJECTDEFS` options to control whether or not to include the object definitions in the trail. These two options are applicable only when the output trail is formatted in Oracle GoldenGate canonical format and the trail format release is greater than 12.2. Otherwise, both options are ignored because no metadata record will be added to the trail.

`TRAILBYTEORDER {BIGENDIAN | LITTLEENDIAN | NATIVEENDIAN}`
Sets the byte format of the metadata in the file records. This parameter does not affect the column data. Valid only for trail files that have a `FORMAT RELEASE` version of at least 12.2. Valid values are `BIGENDIAN` (big endian), `LITTLEENDIAN` (little endian), and `NATIVEENDIAN` (default of the local system). The default is `BIGENDIAN`. See the `GLOBALS` version of TRAILBYTEORDER for additional usage instructions.

Examples

Example 1

```
EXTFILE dirdat/ef
```

Example 2

```
EXTFILE dirdat/ef, MEGABYTES 200
```

Example 3

```
EXTFILE /ggs/dirdat/extdat, FORMAT RELEASE 18.1
```

EXTRACT

Valid For

Extract

Description

Use the EXTRACT parameter to specify an Extract group for online (continuous) change synchronization. This parameter links the current run with previous runs, so that data continuity is maintained between source and target tables. Unless stopped by a user, Extract runs continuously and maintains 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.

Default

None

Syntax

```
EXTRACT group_name
[NLS_LENTH_SEMANTICS BYTE | CHAR ]
```

group_name
The group name as defined with the ADD EXTRACT command.

NLS_LENGTH_SEMANTICS [BYTE | CHAR]
Use this option to switch index values between bytes and characters. The default is bytes.

Example

The following specifies an Extract group named finance.

```
EXTRACT finance
```

EXTTRAIL

Valid For

Extract

Description

Use the EXTTRAIL parameter to specify a trail on the local system that was created with the ADD EXTTRAIL command. The trail is read by an Distribution Path, or by a Replicat on the local system.

EXTTRAIL must precede all associated TABLE statements. Multiple EXTTRAIL statements can be used to define different trails.

From Oracle GoldenGate 19c onwards, the primary Extract writes trail file in the same format as existing trail file format when you upgrade, unless you explicitly specify the trail file format version using the FORMAT RELEASE option. This prevents subsequent Replicats from abending if they are not upgraded.

You can encrypt the data in this trail by using the `ENCRYPTTRAIL` parameter. See "ENCRYPTTRAIL | NOENCRYPTTRAIL" for more information.

> **Note:**
>
> When using the `ENCRYPTRAIL` parameter with the `EXTTRAIL` parameter, ensure that the `ENCRYPTRAIL` parameter is mentioned before EXTTRAIL, else the trail will not be encrypted.

Default

None

Syntax

```
EXTTRAIL file_name
[, FORMAT RELEASE major.minor]
[, TRAILBYTEORDER {BIGENDIAN | LITTLEENDIAN | NATIVEENDIAN}]
```

trail_name
The relative or fully qualified path name of the trail. The trail name can contain only two characters.

> **Note:**
>
> In Microservices Architecture, the trail file name two-character prefix must start with an alphabet only.

Oracle GoldenGate appends this name with a nine-digit sequence number whenever a new file is created. For example, a trail named `/tr` would have files named `/tr000000001`, `/tr000000002`.

FORMAT RELEASE *major.minor*
Not valid for an Extract. Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones.
`FORMAT RELEASE` specifies an Oracle GoldenGate release version. *`major`* is the major version number, and *`minor`* is the minor version number. The X.x must reflect a current or earlier, generally available (GA) release of Oracle GoldenGate. Valid values are 12.2 through the current Oracle GoldenGate X.x version number, 19.1.
The release version is programmatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail.

> **Note:**
>
> The lowest supported version is 12.2.

> **Note:**
>
> RELEASE versions earlier than 12.1 do not support three-part object names.

> **Note:**
>
> If using multiple trails in a single Extract, only RELEASE versions that are the same can coexist.

TRAILBYTEORDER {BIGENDIAN | LITTLEENDIAN | NATIVEENDIAN}
Not valid for an Extract. Sets the byte format of the metadata in the trail records. This parameter does not affect the column data. Valid only for trails that have a FORMAT RELEASE version of at least 12.1. Valid values are BIGENDIAN (big endian), LITTLEENDIAN (little endian), and NATIVEENDIAN (default of the local system). The default is BIGENDIAN. See the GLOBALS version of TRAILBYTEORDER for additional usage instructions.

Examples

Example 1

```
EXTTRAIL dirdat/ny
```

Example 2

```
EXTTRAIL /ggs/dirdat/ex, FORMAT RELEASE 18.1
```

Example 3

Two trail formats within the same sets of tables being captured:

```
EXTTRAIL ./dirdat/ea
TABLE hr.tab1
TABLE hr.tab2
EXTTRAIL ./dirdat/eb
TABLE scott.tab3
TABLE scott.tab4
```

FETCHOPTIONS

Valid For

Extract

Description

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

- When the transaction record does not contain enough information for Extract to reconstruct an update operation.
- When Oracle GoldenGate must fetch a column value as the result of a MISSINGCOLS clause of a TABLE statement.

FETCHOPTIONS is table-specific. One FETCHOPTIONS statement applies for all subsequent TABLE statements until a different FETCHOPTIONS statement is encountered.

Default fetch properties are adequate for most installations.

Default

Ignore missing rows and continue processing

Syntax

```
FETCHOPTIONS
[, FETCHPKUPDATECOLS]
[, MISSINGCOLS]
[, INCONSISTENTROW action]
[, MAXFETCHSTATEMENTS number]
[, MISSINGROW action]
[, NOFETCH]
[, SUPPRESSDUPLICATES]
[, USEKEY | NOUSEKEY]
[, USELATESTVERSION | NOUSELATESTVERSION]
[, USESNAPSHOT | NOUSESNAPSHOT]
[, USEROWID | NOUSEROWID]
```

FETCHPKUPDATECOLS
Fetches all unavailable columns when a primary key is updated. This option is off by default. When off, column fetching is performed according to other `FETCHOPTIONS` options that are enabled.
When on, it only takes effect during an update to a primary key column. The results are the same as using `(*)` in the `TABLE` statement. `LOB` columns are included in the fetch.
Use this parameter when using `HANDLECOLLISIONS`. When Replicat detects a missing update, all of the columns will be available to turn the update into an insert.

MISSINGCOLS
Fetches any missing columns from update and delete operations, including LOB columns. This option is only valid for Oracle Database. It can negatively impact the database and the Extract performance due to additional queries to fetch the data. Especially if there are large LOB values that need to be fetched and written to the trail.
Setting this parameter is the same as setting the following parameters:

```
MISSINGCOLS(*) in the TABLE statement
NOCOMPRESSDELETES FETCHMISSINGCOLS
GETUPDATEBEFORES
NOCOMPRESSUPDATES
LOGALLSUPCOLS
```

However, setting `FETCHOPTIONS MISSINGCOLS` conflicts with the following parameters:

```
FETCHOPTIONS NOFETCH
FETCHOPTIONS FETCHPKUPDATECOLS
COMPRESSDELETES
COMPRESSUPDATES
GETUPDATEBEFORES
LOGALLSUPCOLS
```

`INCONSISTENTROW` *action*
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).

action can be one of the following:

ALLOW
Allow the condition and continue processing.

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.

DISCARD
Discard the data and do not process the partial row.

ABEND
Discard the data and quit processing.

MAXFETCHSTATEMENTS *number*
Controls the maximum allowable number of prepared queries that can be used by Extract to fetch row data from a source database. The fetched data is used when not enough information is available to construct a logical SQL statement from a transaction log record. Queries are prepared and cached as needed. When the value set with `MAXFETCHSTATEMENTS` is reached, the oldest query is replaced by the newest one. The value of this parameter controls the number of open cursors maintained by Extract for fetch queries only. Additional cursors may be used by Extract for other purposes, such as those required for stored procedures. This parameter is only valid for Oracle databases.The default is 100 statements. Make certain that the database can support the number of cursors specified, plus cursors used by other applications and processes.

MISSINGROW *action*
Provides a response when Oracle GoldenGate cannot locate a row to be fetched, causing only part of the row (the changed values) to be available for processing. Typically a row cannot be located because it was deleted between the time the change record was created and when the fetch was triggered, or because the row image required was older than the undo retention specification.
action can be one of the following:

ALLOW
Allow the condition and continue processing. This is the default.

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.

DISCARD
Discard the data and do not process the partial row.

ABEND
Discard the data and quit processing.

NOFETCH
Prevents Extract from fetching the column from the database, this option is off by default. Extract writes the record to the trail, but inserts a token indicating that the column is missing.

SUPPRESSDUPLICATES
Valid for Oracle. Avoids target tablespaces becoming overly large when updates are made on LOB columns. By default, SUPPRESSDUPLICATES is set to off. For example, after replication a source tablespace of 232MB becomes a target tablespace of 7.52GB.

USEKEY | NOUSEKEY
Determines whether or not Oracle GoldenGate uses the primary key to locate the row to be fetched.
If both USEKEY and USEROWID are specified, ROWID takes priority for faster access to the record. USEROWID is the default.

USELATESTVERSION | NOUSELATESTVERSION
Valid for Oracle. Use with USESNAPSHOT. The default, USELATESTVERSION, directs Extract to fetch data from the source table if it cannot fetch from the undo tablespace. NOUSELATESTVERSION directs Extract to ignore the condition if the snapshot fetch fails, and continue processing.
To provide an alternate action if a snapshot fetch does not succeed, use the MISSINGROW option.

USESNAPSHOT | NOUSESNAPSHOT
Valid for Oracle. The default, USESNAPSHOT, causes Extract to use the Oracle Flashback mechanism to fetch the correct snapshot of data that is needed to reconstruct certain operations that cannot be fully captured from the redo record. NOUSESNAPSHOT causes Extract to fetch the needed data from the source table instead of the flashback logs.

USEROWID | NOUSEROWID
Valid for Oracle. Determines whether or not Oracle GoldenGate uses the row ID to locate the row to be fetched.
If both USEKEY and USEROWID are specified, ROWID takes priority for faster access to the record. USEROWID is the default.

Examples

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

```
MAXFETCHSTATEMENTS 150
```

Example 3
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
```

FETCHUSERIDALIAS

Valid For

Extract on Oracle

Description

Use the `FETCHUSERIDALIAS` parameter to specify authentication for an Oracle GoldenGate process to use when logging into a database. The use of `FETCHUSERIDALIAS` requires the use of an Oracle GoldenGate credential store. Specify `FETCHUSERIDALIAS` before any `TABLE` or `MAP` entries in the parameter file.

Default

None

Syntax

```
FETCHUSERIDALIAS alias [DOMAIN domain] [SYSDBA]
```

alias
Specifies the alias of a database user credential that is stored in the Oracle GoldenGate credential store.

DOMAIN *domain*
Specifies the credential store domain for the specified alias. A valid domain entry must exist in the credential store for the specified alias.

SYSDBA
Specifies that the user logs in as `sysdba`.

Example

```
fetchuseridalias gg_user@adg_inst password pwd
```

FILTERDUPS | NOFILTERDUPS

Valid For

Replicat

Description

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.

`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
```

FILEGROUP

Valid For

For SQL Server only.

Description

Overrides the default database filegroup (PRIMARY) when adding TRANDATA to tables and creating the SQL Server CDC capture tables on the designated filegroup.

The filegroup must exist in the database and have a valid database file attached to it. The `GLOBALS FILEGROUP` parameter can be overwritten by the following statement, if required, but this is not normally necessary:

```
ADD TRANDATA schemaname.tablename FILEGROUP filegroupname
```

If the `FILEGROUP` parameter exists in GLOBALS, then the `ADD HEARTBEATTABLE` command also creates the SQL Server CDC capture tables for the heartbeat tables on the designated filegroup.

See Enabling Supplemental Logging (CDC Extract).

FLUSHSECS | FLUSHCSECS

Valid For

Extract

Description

Use the FLUSHSECS or FLUSHCSECS parameters to control when Oracle GoldenGate flushes the Extract memory buffer. When sending data to remote systems, Extract buffers data to optimize network performance. The buffer is flushed to the target system when it is full or after the amount of time specified with FLUSHSECS or FLUSHCSECS. Data changes are not available to the target users until the buffer is flushed and the data is posted.

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.

This parameter cannot be set to zero (0).

Default

The default is 1. The minimum is 0; the maximum is 5000.

Syntax

```
FLUSHSECS seconds | FLUSHCSECS centiseconds
```

seconds
The delay, in seconds, before flushing the buffer.

centiseconds
The delay, in centiseconds, before flushing the buffer.

Example

```
FLUSHSECS 80
```

FUNCTIONSTACKSIZE

Valid For

Extract and Replicat

Description

Use the FUNCTIONSTACKSIZE parameter to control the size of the memory stack that is used for processing Oracle GoldenGate column-conversion functions. The memory stack holds arguments supplied to and from an Oracle GoldenGate function. You should not need to use this parameter unless Oracle GoldenGate returns a message indicating that the size of the stack should be increased. The message is similar to:

```
Not enough stack space. Specify FUNCTIONSTACKSIZE greater than {0,number,0}
```

This could happen when you are using a very large number of functions or arguments.

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

When setting FUNCTIONSTACKSIZE for a coordinated Replicat, take into account that the specified value is applied to each thread in the configuration, not as an aggregate threshold for

Replicat as a whole. For example, if FUNCTIONSTACKSIZE 400 is specified, it is possible for each thread to have 399 arguments without any warning or error from Replicat.

FUNCTIONSTACKSIZE must appear in the parameter file before any parameters that include functions are listed. FUNCTIONSTACKSIZE is a global parameter. It affects all clauses in a parameter file.

Default

200 arguments

Syntax

```
FUNCTIONSTACKSIZE number
```

number
A value between 0 and 5000 that denotes the number of function arguments to allow in a parameter clause.

Example

```
FUNCTIONSTACKSIZE 300
```

GETDELETES | IGNOREDELETES

Valid For

Extract and Replicat

Description

Use the GETDELETES and IGNOREDELETES parameters to control whether or not Oracle GoldenGate processes DELETE operations. These parameters are table-specific. One parameter remains in effect for all subsequent TABLE or MAP statements, until the other parameter is encountered.

Because you can selectively enable or disable these parameters between MAP statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the GETDELETES threads in one set of MAP statements, and specify the IGNOREDELETES threads in a different set of MAP statements.

If this parameter is used, and a primary key or unique key is reused then that Replicat may get a duplicate primary key or unique key error when it attempts to apply the insert. You need to disable this constraint (and leave the index) on the target. If this is done, the source table gets supplemental logging on all columns. Use KEYCOLS (*) in the TABLE statement on the source, so that the Replicat has all the necessary columns to perform any update operations.

Default

```
GETDELETES
```

Syntax

```
GETDELETES | IGNOREDELETES
```

Example

This example shows how you can apply GETDELETES and IGNOREDELETES selectively to different MAP statements, each of which represents a different thread of a coordinated Replicat.

```
GETDELETES
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
IGNOREDELETES
MAP sales.loc, TARGET sales.loc, THREAD (3);
```

In this example, delete operation on `sales.loc` is skipped. As a best practice, you should re-enable `GETDELETES`.

GETINSERTS | IGNOREINSERTS

Valid For

Extract and Replicat

Description

Use the `GETINSERTS` and `IGNOREINSERTS` parameters to control whether or not `INSERT` operations are processed by Oracle GoldenGate. These parameters are table-specific. One parameter remains in effect for all subsequent `TABLE` or `MAP` statements, until the other parameter is encountered.

Because you can selectively enable or disable these parameters between `MAP` statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the `GETINSERTS` threads in one set of `MAP` statements, and specify the `IGNOREINSERTS` threads in a different set of `MAP` statements.

Default

```
GETINSERTS
```

Syntax

```
GETINSERTS | IGNOREINSERTS
```

Example

This example shows how you can apply `GETINSERTS` and `IGNOREINSERTS` selectively to different `MAP` statements, each of which represents a different thread of a coordinated Replicat.

```
IGNOREINSERTS
MAP sales.loc, TARGET sales.loc, THREAD (3);
GETINSERTS
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
```

GETTRUNCATES | IGNORETRUNCATES

Valid For

Extract and Replicat

Description

Use the `GETTRUNCATES` and `IGNORETRUNCATES` parameters to control whether or not Oracle GoldenGate processes table truncate operations. By default, truncate operations are not captured from the source or replicated to the target.

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

In a coordinated Replicat configuration, truncates are always processed by the thread that is responsible for barrier transactions.

Supported Databases

- GETTRUNCATES and IGNORETRUNCATES are supported by Extract for Oracle Database, MySQL, DB2 LUW, PostgreSQL, and DB2 for i.
- GETTRUNCATES and IGNORETRUNCATES are supported by Replicat for Oracle Database, SQL Server, DB2 for i, DB2 LUW, DB2 z/OS, MySQL, Teradata, PostgreSQL, and TimesTen.

> **Note:**
>
> GETTRUNCATES and IGNORETRUNCATES for DB2 z/OS is only valid for TRUNCATE IMMEDIATE operations. TRUNCATES without the IMMEDIATE qualifier will be processed without regard to GETTRUNCATES and IGNORETRUNCATES as it appears as a DROP FROM command in the DB2 logs.

DB2 LUW Limitations

- DB2 LUW does not support a TRUNCATE command, so Replicat replicates a truncate operation by performing an IMPORT REPLACE from a NULL (blank) file.

Oracle Limitations

- Oracle GoldenGate supports the Oracle TRUNCATE TABLE command, but not TRUNCATE PARTITION. You can replicate TRUNCATE PARTITION as part of the full Oracle GoldenGate DDL replication support.
- The database does not log truncates against an empty table, so those operations are not captured by Oracle GoldenGate. The DDL support of Oracle GoldenGate can be used for this purpose.
- The database does not log truncates for empty partitions, so Oracle GoldenGate cannot reliably process TRUNCATE TABLE when the table contains any empty partitions. Do not use GETTRUNCATES on any partitioned table. Oracle GoldenGate DDL support can be used to capture truncates on tables that might include empty partitions.

PostgreSQL Limitations

Oracle GoldenGate capture supports GETTRUNCATES from PostgreSQL version 11 and higher.

Default

```
IGNORETRUNCATES
```

Syntax

```
GETTRUNCATES | IGNORETRUNCATES
```

GETUPDATEAFTERS | IGNOREUPDATEAFTERS

Valid For

Extract and Replicat

Description

Use the GETUPDATEAFTERS and IGNOREUPDATEAFTERS parameters to control whether or not the after images of columns in UPDATE operations are included in the records processed by Oracle GoldenGate. After images contain the results of the UPDATE.

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

Because you can selectively enable or disable these parameters between MAP statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the GETUPDATEAFTERS threads in one set of MAP statements, and specify the IGNOREUPDATEAFTERS threads in a different set of MAP statements.

Default

GETUPDATEAFTERS

Syntax

```
GETUPDATEAFTERS | IGNOREUPDATEAFTERS
```

Example

This example shows how you can apply GETUPDATEAFTERS and IGNOREUPDATEAFTERS selectively to different MAP statements, each of which represents a different thread of a coordinated Replicat.

```
GETUPDATEAFTERS
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
IGNOREUPDATEAFTERS
MAP sales.loc, TARGET sales.loc, THREAD (3);
```

GETUPDATEBEFORES | IGNOREUPDATEBEFORES

Valid For

Extract and Replicat

Description

Use the GETUPDATEBEFORES and IGNOREUPDATEBEFORES parameters to control whether or not the before images of columns in UPDATE operations are included in the records that are processed by Oracle GoldenGate. Before images contain column details that existed before a row was updated.

(Oracle only) Oracle GoldenGate captures both the pre-change and post-change values for update operations in a single unified update record by default. For other databases, only the pre-change values are written to trail file. In previous releases the default was to only capture the post-change value. Beginning in this release, custom SQL statements (SQLEXEC) now only

execute once per update operation with the new default update format. Prior to this release, custom SQL statements would execute twice, once when encountering the pre-change value and once when encountering the post-change value. If you are using the Oracle GoldenGate with the unified update format, you can explicitly pass the pre or post-value to the custom SQL statement using the `@BEFORE`, `@AFTER`, and `@BEFOREAFTER` functions. Though Oracle GoldenGate can use this update format by default, the old format cam be preserved if there are conflicting parameters that would have previously generated two separate pre and post change records. In these cases, an informational message is logged in the report file.

Use the `GETUPDATEBEFORES` parameter as follows:

- in the Extract parameter file to extract before images from the data source.
- in the Replicat parameter file to include before images in a Replicat operation.

You can compare before images with after images to identify the net results of a transaction or perform other delta calculations. For example, if a `BALANCE` field is $100 before an update and $120 afterward, a comparison would show the difference of $20. You can use the column-conversion functions of Oracle GoldenGate to perform the comparisons and calculations.

To reference before images in the parameter file, use the `@BEFORE` conversion function. For example:

```
COLMAP (previous = @BEFORE (balance))
```

`GETUPDATEBEFORES` is required when using the Conflict Detection and Resolution (CDR) feature. See Manual Conflict Detection and Resolution for more information about CDR.

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.

Because you can selectively enable or disable these parameters between `MAP` statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the `GETUPDATEBEFORES` threads in one set of `MAP` statements, and specify the `IGNOREUPDATEBEFORES` threads in a different set of `MAP` statements.

Limitations for `GETUPDATESBEFORES`:

- For PostgreSQL, before images of LOB columns are not logged and will not be written to the trails.
- For SQL Server, columns of `IMAGE`, `NTEXT`, and `TEXT` data types are logged as a `NULL` value for before image update operations, and columns of `VARBINARY(MAX)`, `VARCHAR(MAX)`, and `NVARCHAR(MAX)` are logged as a NULL value for before image update operations unless the column was updated.

Default

```
IGNOREUPDATEBEFORES
```

Syntax

```
GETUPDATEBEFORES | IGNOREUPDATEBEFORES
```

Example

This example shows how you can apply `GETUPDATEBEFORES` and `IGNOREUPDATEBEFORES` selectively to different `MAP` statements, each of which represents a different thread of a coordinated Replicat.

```
GETUPDATEBEFORES
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
IGNOREUPDATEBEFORES
MAP sales.loc, TARGET sales.loc, THREAD (3);
```

GETUPDATES | IGNOREUPDATES

Valid For

Extract and Replicat

Description

Use the `GETUPDATES` and `IGNOREUPDATES` parameters to control whether or not Oracle GoldenGate processes `UPDATE` operations. These parameters are table-specific. One parameter remains in effect for all subsequent `TABLE` or `MAP` statements, until the other parameter is encountered.

Because you can selectively enable or disable these parameters between `MAP` statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the `GETUPDATES` threads in one set of `MAP` statements, and specify the `IGNOREUPDATES` threads in a different set of `MAP` statements.

Default

```
GETUPDATES
```

Syntax

```
GETUPDATES | IGNOREUPDATES
```

Example

This example shows how you can apply `GETUPDATES` and `IGNOREUPDATES` selectively to different `MAP` statements, each of which represents a different thread of a coordinated Replicat.

```
IGNOREUPDATES
MAP sales.loc, TARGET sales.loc, THREAD (3);
GETUPDATES
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
```

GGSCHEMA

Valid For

GLOBALS

Description

Use this parameter to specify the name of the schema that contains Oracle GoldenGate database objects, such as those that support Oracle DDL replication for trigger-based replication, those that are a part of a heartbeat table implementation, and those that are part of the SQL Server CDC Capture and Cleanup implementation.

The schema name mentioned under `GGSCHEMA` should be treated as a reserved schema and should not be used as part of a `TABLE` or `MAP` statement within an Extract, or Replicat. If you need to capture and replicate objects in GGSCHEMA, don't use wildcards and ensure that you explicitly map the respective table names. It's recommended that you take assistance from Oracle Support to make any changes including any `CREATE`, `ALTER`, or `DROP` of objects in this schema.

This parameter is valid for all databases.

Default

None

Syntax

```
GGSCHEMA [container.]schema_name
```

[container.]schema_name
The fully qualified name of the Oracle GoldenGate objects' schema. Use the full two-part name if the schema is within an Oracle container database.

Example

```
GGSCHEMA ogg
```

GROUPTRANSOPS

Valid For

Replicat

Description

Use the `GROUPTRANSOPS` parameter to control the number of SQL operations that are contained in a Replicat transaction when operating in its normal mode. For using `GROUPTRANSOPS` with `BATCHSQL`, see BATCHSQL parameter, which has additional options for changing how transactions are applied to the target database that may improve Replicat performance.

Increasing the number of operations in a Replicat transaction improves the performance of Oracle GoldenGate by:

- Reducing the number of transactions executed by Replicat.
- Reducing I/O activity to the checkpoint file and the checkpoint table, if used. Replicat issues a checkpoint whenever it applies a transaction to the target, in addition to its scheduled checkpoints.

Replicat accumulates operations from source transactions, in transaction order, and applies them as a group within one transaction on the target. `GROUPTRANSOPS` sets a minimum value rather than an absolute value, to avoid splitting apart source transactions. Replicat waits until it

receives all operations from the last source transaction in the group before applying the target transaction.

For example, if transaction 1 contains 200 operations, and transaction 2 contains 400 operations, and transaction 3 contains 500 operations, the Replicat transaction contains 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.

Table 2-10 Replicat GROUPTRANSOPS

Source Transactions (assumes same table and column list)	Replicat transaction in normal (GROUPTRANSOPS) mode
Transaction 1:	**Transaction:**
INSERT	INSERT
DELETE	DELETE
Transaction 2:	INSERT
INSERT	DELETE
DELETE	INSERT
Transaction 3:	DELETE
INSERT	
DELETE	

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.

(Oracle only) For an integrated Replicat, `GROUPTRANSOPS` is effective only when the integrated Replicat parameter `PARALLELISM` is set to 1.

Default

Nonintegrated Replicat: 1000 operations, Integrated Replicat: 50 operations

Syntax

```
GROUPTRANSOPS number
```

number
The minimum number of operations to be applied in a Replicat transaction. A value of 1 executes the operations within the same transaction boundaries as the source transaction. The value must be at least 1.

Example

```
GROUPTRANSOPS 2000
```

HANDLECOLLISIONS | NOHANDLECOLLISIONS

Valid For

Replicat

Description

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, called *collisions*, occur during an initial load, when data from source tables is being loaded to target tables while Oracle GoldenGate is replicating transactional changes that are being made to those tables. When Oracle GoldenGate applies the replicated changes after the load is finished, HANDLECOLLISIONS provides Replicat with error-handling logic for these collisions.

You can use HANDLECOLLISIONS and NOHANDLECOLLISIONS in the following ways:

- You can enable HANDLECOLLISIONS and NOHANDLECOLLISIONS in a global manner by specifying them at the root level of the parameter file. One parameter remains enabled for all subsequent MAP statements in the parameter file, until the opposing parameter is encountered.
- You can enable HANDLECOLLISIONS or NOHANDLECOLLISIONS within a specific MAP parameter to enable or disable error handling only for that source-target mapping.

The preceding methods can be combined. You can specify a global collisions-handling rule and then override that rule with different collisions-handling rules in the MAP statements. A MAP specification always overrides the global specification.

> **Note:**
>
> Error Handling of Integrated Replicat is not appropriate with HANDLECOLLISIONS. Oracle recommends that you use precise instantiation methods instead of using HANDLECOLLISIONS.

How HANDLECOLLISIONS Works

The following example explains how HANDLECOLLISIONS works:

- When Replicat encounters an update to a column that Oracle GoldenGate is using as a key, the handling is as follows:
  - If the row with the old key is not found in the target, the change record in the trail is converted to an insert.
  - If a row with the new key exists in the target, Replicat deletes the row that has the old key (it would not exist if the update had executed successfully), and then the row with the new key is updated as an overlay where the trail values replace the current values.

  This logic requires all of the columns in the table (not just the ones that changed) to be logged to the transaction log, either by default or by force, such as by using the COLS option of ADD TRANDATA for an Oracle database. See Possible Solutions to Avoid Missing Column Values.
- When Replicat encounters a duplicate-record error, the static record that was applied by the initial load is overwritten by the change record in the trail. Overlaying the change is safer from an operational standpoint than ignoring the duplicate-record error.
- Replicat with HANDLECOLLISIONS doesn't discard the change record in the trail even if update or delete operation doesn't affect a key column in the source and Replicat encounters a missing-record error in the target. These errors happen when a record is

changed on the source system and then the record is deleted before the table data is extracted by the initial-load process. For example:

1. The application updates record A in source table1.
2. Extract extracts the update.
3. The application deletes record A in source table1.
4. Extract extracts the delete.
5. Oracle GoldenGate extracts initial-load data from source table1, without record A.
6. Oracle GoldenGate applies the initial load, without record A.
7. Replicat attempts to apply the update of record A.
8. The database returns a "record missing" error.
9. Replicat attempts to apply the delete of record A.
10. The database returns a "record missing" error.

Disable `HANDLECOLLISIONS` after the transactional changes captured during the initial load are applied to the target tables, so that Replicat does not automatically handle subsequent errors. Errors generated after initial synchronization indicate an abnormal condition and should be evaluated by someone who can determine how to resolve them. For example, a missing-record error could indicate that a record which exists on the source system was inadvertently deleted from the target system.

You can turn off `HANDLECOLLISIONS` in the following ways:

- Stop Replicat and remove `HANDLECOLLISIONS` from the Replicat parameter file (can cause target latency). Alternatively, you can edit the parameter file to add `NOHANDLECOLLISIONS` before the `MAP` statements for which you want to disable the error handling.
- While Replicat is running, run GGSCI and then use the `SEND REPLICAT` command with the `NOHANDLECOLLISIONS` option for the tables that you want to affect.

> **Note:**
>
> If using `SEND REPLICAT`, make certain to remove `HANDLECOLLISIONS` from the parameter file or add a `NOHANDLECOLLISIONS` parameter before starting another Replicat run, so that `HANDLECOLLISIONS` does not activate again.

Possible Solutions to Avoid Missing Column Values

When a database does not log all of the column values of a source table by default, there can be errors if the target table has `NOT NULL` constraints when Replicat attempts to convert a primary-key update to an insert. You can work around this scenario in the following ways:

- `HANDLECOLLISIONS` requires that the table have a NOT NULL primary key or NOT NULL unique constraint on the target table.
- Use the `NOCOMPRESSUPDATES` parameter in the Extract parameter file to send all of the columns of the table to the trail, and configure the database to log all column values. By default, Extract only writes the primary key and the columns that changed to the trail. This is the safest method, because it writes the current values at the time when the operation is performed and eliminates the need for fetching.

- Use the FETCHOPTIONS parameter with the FETCHPKUPDATECOLS option in the Extract parameter file. This configuration causes Extract to fetch unavailable columns when a key column is updated on the source. A fetch is the *current* value, not necessarily the value at the time of a particular update, so there can be data integrity issues. See "FETCHOPTIONS" for more information and additional fetch options to handle unsuccessful fetches.

If the database includes all columns by default, then you must use NOCOMPRESSUPDATES and NOCOMPRESSDELETES for HANDLECOLLISIONS to work properly. If the database does not support NOCOMPRESSDELETES, you must use FETCHOPTIONS MISSINGCOLS.

See About Instantiating with Initial Load Extract for more information about Oracle GoldenGate initial load methods.

Default

NOHANDLECOLLISIONS

Syntax

```
HANDLECOLLISIONS | NOHANDLECOLLISIONS [_ALLOWPKMISSINGROWCOLLISIONS]
[THREADS (threadID[, threadID][, ...][, thread_range[, thread_range][, ...])]
```

HANDLECOLLISIONS
Enables collision handling.

_ALLOWPKMISSINGROWCOLLISIONS
Use HANDLECOLLISIONS with _ALLOWPKMISSINGROWCOLLISIONS to skip primary-key UPDATE operations if the corresponding target row does not exist.

> **Note:**
>
> Skipping operations can cause data corruption. See the Description in this topic.

NOHANDLECOLLISIONS
Turns off collision handling.

THREADS (*threadID*[, *threadID*][, ...][, *thread_range*[, *thread_range*][, ...])
Enables HANDLECOLLISIONS for the specified threads. When used in a global HANDLECOLLISIONS statement at the root level of the parameter file, HANDLECOLLISIONS is enabled for the specified threads wherever they are in all MAP statements where . When used in a HANDLECOLLISIONS clause of a MAP statement, HANDLECOLLISIONS is enabled only for that MAP statement.

threadID[, threadID][, ...]
Specifies a thread ID or a comma-delimited list of threads in the format of threadID, threadID, threadID.

thread_range[, thread_range][, ...]
Specifies a range of threads in the form of threadIDlow-threadIDhigh or a comma-delimted list of ranges in the format of threadIDlow-threadIDhigh, threadIDlow-threadIDhigh.

A combination of these formats is permitted, such as threadID, threadID, threadIDlow-threadIDhigh.

Examples

Example 1
This example 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
This example 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
This example shows the basic use of HANDLECOLLISIONS within a MAP statement.

```
MAP dbo.tcust, TARGET dbo.tcust, HANDLECOLLISIONS;
```

Example 4
This example 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;
```

Example 5
In the following example, HANDLECOLLISIONS is enabled globally for all MAP statements, except for default thread 0 in the first MAP statement and for thread 3 in the second MAP statement.

```
HANDLECOLLISIONS
MAP fin.*, TARGET fin.*;
MAP sales.*, TARGET sales.*;
MAP orders.*, TARGET orders.*;
MAP scott.cust, TARGET scott.cust, NOHANDLECOLLISIONS;
MAP amy.cust, TARGET amy.cust, THREAD(3), NOHANDLECOLLISIONS;
```

Example 6
In this example, HANDLECOLLISIONS is enabled globally, but turned off for thread 3. The remaining threads 1, 2, and 4 will handle collisions.

```
HANDLECOLLISIONS
NOHANDLECOLLISIONS THREAD(3)
MAP scott.emplyees, TARGET scott.employees, THREADRANGE(1,4, OID);
MAP scott.inventory, TARGET scott.inventory, THREADRANGE(1,4, OID);
MAP scott.cust, TARGET scott.cust, THREADRANGE(1,4, OID);
```

Example 7
In this example, HANDLECOLLISIONS is enabled globally, then disabled globally for threads 5 through 7. In the first map statement, all threads will handle collisions, since the HANDLECOLLISIONS parameter does not specify a thread or a range. In the second map statement, only threads 4, 8, and 9 will handle collisions, because the global NOHANDLECOLLISIONS applies to threads 5-7.

```
HANDLECOLLISIONS
NOHANDLECOLLISIONS THREADRANGE(5-7)
MAP scott.cust, TARGET scott.cust, THREADRANGE(4,9,OID), HANDLECOLLISIONS;
MAP scott.offices, TARGET scott.offices, THREADRANGE(4,9,OID);
MAP scott.emp, TARGET scott.emp, THREADRANGE(4,9,OID);
MAP scott.ord, TARGET scott.ord, THREADRANGE(4,9,OID);
MAP acct.*, TARGET acct.*;
MAP admin.*, TARGET admin.*;
```

HAVEUDTWITHNCHAR

Valid For

Replicat (Oracle only)

Description

Use the HAVEUDTWITHNCHAR parameter when the source data contains user-defined types that have an NCHAR, NVARCHAR2, or NCLOB attribute. When this data is encountered in the trail, HAVEUDTWITHNCHAR causes Replicat to connect to the Oracle target in AL32UTF8, which is required when a user-defined data type contains one of those attributes.

HAVEUDTWITHNCHAR is not required if the character set of the target is AL32UTF8. However, it is required if only NLS_LANG is set to AL32UTF8 on the target. By default Replicat ignores NLS_LANG and connects to an Oracle database in the native character set of the database. Replicat uses the OCIString object of the Oracle Call Interface, which does not support NCHAR, NVARCHAR2, or NCLOB attributes, so Replicat must bind them as CHAR. Connecting to the target in AL32UTF8 prevents data loss in this situation.

HAVEUDTWITHNCHAR must be specified before the USERIDALIAS parameter in the parameter file.

Default

None

Syntax

```
HAVEUDTWITHNCHAR
```

HEARTBEATTABLE

Valid For

GLOBALS

Description

Use HEARTBEATTABLE to specify a non-default name of the heartbeat table. The table name GG_HEARTBEAT is the default. This name used to denote the heartbeat table is used to create a seed and history table, GG_HEARTBEAT_SEED and GG_HEARTBEAT_HISTORY respectively.

Specifying one name reserves all names used by the heartbeat infrastructure. If the schema name is not specified, the value in GGSCHEMA is used for schema name.

Default

None

Syntax

```
HEARTBEATTABLE schema_name.heartbeat_table_name
```

schema_name
The name of the schema you want to use with the heartbeat table. This is not needed if you have specified the schema using the GGSCHEMA parameter in your GLOBALS file.

heartbeat_table_name
The heartbeat table name you want to use. The default table name is GG_HEARTBEAT.

INCLUDE

Valid For

Extract and Replicat

Description

Use the INCLUDE parameter to include a macro library in a parameter file. See Using Macros for more information about using macros.

Default

None

Syntax

```
INCLUDE library
```

library
The relative or full path to library file.

Example

The following example includes macro library mdatelib.mac.

```
INCLUDE /ggs/dirprm/mdatelib.mac
```

INCLUDETAG

Valid For

(Oracle) Extract and Replicat

(All databases) Extract or Replicat

Description

Use INCLUDETAG *tag* in your Replicat parameter file to include specific changes trail files. The tag value can be up to 2000 hexadecimal digits (0-9 A-F).

> **Note:**
>
> FFFF and + (plus symbol) are either reserved or not supported for tag usage.

This tag implicitly implies an `EXCLUDETAG+`, so any tagged operations that don't match the tag listed in `INCLUDETAG` are filtered out. For example, `includetag AA` includes untagged transactions in addition to transactions that are tagged AA, while filtering out all other tagged operations.

Default

None

Syntax

```
INCLUDETAG tag
```

Example

For Replicat:

```
includetag 00
```

INITIALLOADOPTIONS

Valid For

Valid for Initial Load Extract for PostgreSQL and SQL Server.

Description

This parameter is used to enable precision instantiation for an initial load Extract, which allows initial data instantiation without incurring application downtime. The Precise instantiation is achieved by creating a consistent point of the database in relation to the transaction log, or identifiable by a transaction log position like LSN. The initial-load Extract reads the data sets that are already committed up to the consistent point. Changes after that consistent point are not captured by the initial load Extract. A CDC Extract can be used in conjunction to capture the transactions after the precise instantiation LSN.

Queries in snapshot isolation level returns data that are committed by the time the transaction is started. Uncommitted changes after the transaction is started are ignored.

By default, precise instantiation is disabled. To enable precise instantiation, use the `INITIALLOADOPTIONS` parameter with the `USESNAPSHOT` option when configuring an online load Extract.

Syntax

```
INITIALLOADOPTIONS USESNAPSHOT
```

Examples

The example shows the use of the parameter with the `USESNAPSHOT` option in the initial load Extract parameter file. The name of the initial load Extract is `extinit`.

```
EXTRACT extinit
INITIALLOADOPTIONS USESNAPSHOT
SOURCEDB psql_src USERIDALIAS ggma PASSWORD We1come23
EXTFILE ei, MEGABYTES 500, PURGE
TABLE public.*;
```

For details on implementation steps, see Add Initial Load Extract for PostgreSQL.

INSERTALLRECORDS

Valid For

Replicat

This parameter does not work with `UPDATERECORDFORMAT COMPACT`.

Description

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 that is 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.

Some cases for using `INSERTALLRECORDS` are the following:

- To work within an exceptions `MAP` statement. In an exceptions `MAP` statement, `INSERTALLRECORDS` causes the values of operations that generated errors to be inserted as new records in an exceptions table as part of an error-handling strategy.
- To maintain a transaction history. By inserting every change to a specific row as a new record in the database, you can maintain a history of all changes made to that row, instead of maintaining just the current version. Each insert is a point-in-time snapshot that can be queried as needed for auditing purposes. Combining historical data with special transaction information provides a way to create a more useful target reporting database.

`INSERTALLRECORDS` can be used at the root level of the parameter file to affect all subsequent `MAP` statements, and it can be used within a `MAP` statement to affect a specific table or multiple tables specified with a wildcard.

Getting More Information about INSERTALLRECORDS

See "TABLE | MAP" for `MAP` syntax.

Default

None

Syntax

```
INSERTALLRECORDS
```

Examples

Example 1
This example shows INSERTALLRECORDS at the root level of the parameter file as part of an exceptions handling configuration.

```
COLMAP (USEDEFAULTS,
         TRAN_TIME = @GETENV("GGHEADER","COMMITTIMESTAMP"),
         OP_TYPE = @GETENV("GGHEADER", "OPTYPE"),
         BEFORE_AFTER_IND = @GETENV("GGHEADER", "BEFOREAFTERINDICATOR"),
         SEQUENCE_ID =  @COMPUTE(@COMPUTE(@NUMSTR(@GETENV ("RECORD",
"FILESEQNO"))*100000000000)+@NUMSTR(@GETENV ("RECORD", "FILERBA")))
);
```

Example 2
This example shows INSERTALLRECORDS in a MAP statement.

```
REPLICAT deliv
USERIDALIAS tiger1
SOURCEDEFS /ggs/dirdef/defs
REPERROR DEFAULT, ABEND
MAP fin.accTAB, TARGET fin.custTAB, INSERTALLRECORDS;
```

INSERTAPPEND | NOINSERTAPPEND

Valid For

Replicat (Oracle Nonintegrated mode)

Description

Use the INSERTAPPEND and NOINSERTAPPEND parameters to control whether or not a Replicat operating in nonintegrated mode uses an APPEND hint when it applies INSERT operations (used for array binding) to Oracle target tables. These parameters are valid only for Oracle databases and are only compatible with BATCHSQL mode.

INSERTAPPEND causes Replicat to use the APPEND_VALUES hint when it applies INSERT operations to Oracle target tables. It is appropriate for use as a performance improvement when the replicated transactions are large and contain multiple inserts into the same table. If the transactions are small, using INSERTAPPEND can cause a performance decrease. For more information about when APPEND hints should be used, consult the Oracle documentation.

The BATCHSQL parameter must be used when using INSERTAPPEND. Replicat will abend if BATCHSQL is not used.

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.

If the table is compressed with row compression or hybrid columnar compression, DML applied by the Replicat is not compressed even when using this parameter.

See "TABLE | MAP" for more information about the MAP parameter.

Default

NOINSERTAPPEND

Syntax

INSERTAPPEND | NOINSERTAPPEND

Examples

Example 1
The following is part of a Replicat parameter file that shows how INSERTAPPEND is used for all of the tables in the fin schema, except for the inventory table.

```
BATCHSQL
INSERTAPPEND
MAP fin.*, TARGET fin2.*;
MAPEXCLUDE fin.inventory;
NOINSERTAPPEND
MAP fin.inventory, TARGET fin2.inventory;
```

Example 2
The following is part of a Replicat parameter file that shows how INSERTAPPEND is used for all of the tables in the MAP statements, except for the inventory table.

```
BATCHSQL
MAP fin.orders, TARGET fin.orders;
MAP fin.customers, TARGET fin.customers;
MAP fin.inventory, TARGET fin.inventory, NOINSERTAPPEND;
```

INSERTDELETES | NOINSERTDELETES

Valid For

Replicat

Description

Use the INSERTDELETES and NOINSERTDELETES parameters to control whether or not Oracle GoldenGate converts source delete operations to insert operations on the target database. The parameters are table-specific. One parameter remains in effect for all subsequent MAP statements, until the other parameter is encountered.

When using INSERTDELETES, use the NOCOMPRESSDELETES parameter so that Extract does not compress deletes.

Because you can selectively enable or disable these parameters between MAP statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the INSERTDELETES threads in one set of MAP statements, and specify the NOINSERTDELETES threads in a different set of MAP statements.

Default

NOINSERTDELETES

Syntax

INSERTDELETES | NOINSERTDELETES

Example

This example shows how you can apply INSERTDELETES and NOINSERTDELETES selectively to different MAP statements, each of which represents a different thread of a coordinated Replicat.

```
NOINSERTDELETES
MAP sales.loc, TARGET sales.loc, THREAD (3);
INSERTDELETES
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
```

INSERTMISSINGUPDATES | NOINSERTMISSINGUPDATES

Valid For

Replicat

Description

Use the INSERTMISSINGUPDATES and NOINSERTMISSINGUPDATES parameters to control whether or not Oracle GoldenGate inserts a record based on the source record when the target record does not exist.

INSERTMISSINGUPDATES inserts the missing update but should only be used when the source database logs all column values, whether or not they changed). It can work with a database that uses a compressed form of updates (where only the changed values are logged) if the target database allows NULL to be used for the missing column values.

If the database includes all columns by default, then you must use NOCOMPRESSUPDATES and NOCOMPRESSDELETES for INSERTMISSINGUPDATES to work properly. If the database does not support NOCOMPRESSDELETES, then you must use FETCHOPTIONS MISSINGCOLS.

When the default of NOINSERTMISSINGUPDATES is in effect, a missing record causes an error, and the transaction may abend depending on REPERROR settings.

The INSERTMISSINGUPDATES and NOINSERTMISSINGUPDATES parameters are table-specific. One parameter remains in effect for all subsequent MAP statements, until the other parameter is encountered.

Default

```
NOINSERTMISSINGUPDATES
```

Syntax

```
INSERTMISSINGUPDATES | NOINSERTMISSINGUPDATES
```

INSERTUPDATES | NOINSERTUPDATES

Valid For

Replicat

Description

Use the INSERTUPDATES and NOINSERTUPDATES parameters to control whether or not Oracle GoldenGate converts update operations to insert operations. For updates to be converted to

inserts, the database must log all column values either by default or by means of supplemental logging.

The parameters are table-specific. One parameter remains in effect for all subsequent `MAP` statements, until the other parameter is encountered.

To ensure that updates are not compressed by Extract when using `INSERTUPDATES`, use the `NOCOMPRESSUPDATES` parameter. If the database includes all columns by default, then you must use `NOCOMPRESSUPDATES` and `NOCOMPRESSDELETES` for `INSERTUPDATES` to work properly. If the database does not support `NOCOMPRESSDELETES`, you must use `FETCHOPTIONS MISSINGCOLS`.

Because you can selectively enable or disable these parameters between `MAP` statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the `INSERTUPDATES` threads in one set of `MAP` statements, and specify the `NOINSERTUPDATES` threads in a different set of `MAP` statements.

Default

```
NOINSERTUPDATES
```

Syntax

```
INSERTUPDATES | NOINSERTUPDATES
```

Example

This example shows how you can apply `INSERTUPDATES` and `NOINSERTUPDATES` selectively to different `MAP` statements, each of which represents a different thread of a coordinated Replicat.

```
INSERTUPDATES
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
NOINSERTUPDATES
MAP sales.loc, TARGET sales.loc, THREAD (3);
```

INSERTUPSERTS | NOINSERTUPSERTS

Valid For

Replicat.

Default is `INSERTUPSERTS`.

Trail file format 19.1 supports `UPSERT` operation type. Older trail file format must be used with `INSERTUPSERTS` to convert `UPSERT` record to `INSERT` record.

By default, specifying `INSERTUPSERTS`, enables Replicat to apply UPSERT record as INSERT. If the row exists, Replicat overwrites the row by the new record.

If the output trail format is 18.1 or older, the `INSERTUPSERTS` option is required, otherwise the primary Extract fails. Primary Extract always writes UPSERT record as INSERT record for 18.1 or older trail, and you need to specify `OVERRIDEDUPS` option to apply the INSERT record that was originally UPSERT.

If the user exit module version is 5 or older, `INSERTUPSERTS` is required. UPSERT record is converted to INSERT record for the user exit version 5 or older, as well as user exit stats record count.

If the output trail format 18.1 or older is specified with NOINSERTUPSERTS, primary Extract fails. User exit module version 6 (enable from 19.1 release) supports UPSERT record type and stats count if NOINSERTUPSERTS is specified. If user exit module is version 5 or older and NOINSERTUPSERTS is specified, primary Extract, pump or Replicat fail.

If UPSERT operation is applied as INSERT by specifying INSERTUPSERTS, stats still count as UPSERT operation.

UPSERT operation type is only output if NOINSERTUPSERTS is specified, otherwise output as INSERT.

Parallel Replicat and Oracle Integrated Replicat does not support both UPSERT and INSERT converted from UPSERT, and fallback to non-integrated classic Replicat mode.

LIST | NOLIST

Valid For

Extract and Replicat

Description

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
```

LOGALLSUPCOLS

Valid For

Extract

Description

Use the LOGALLSUPCOLS parameter to control the writing of supplementally logged columns specified with ADD TRANDATA or ADD SCHEMATRANDATA to the trail.

LOGALLSUPCOLS supports integrated Replicat (for Oracle database) and the Oracle GoldenGateConflict Detection and Resolution feature (CDR). The supplementally logged

columns are a union of the scheduling columns that are required to ensure data integrity across parallel Replicat threads and the conflict detection and resolution (CDR) columns. Scheduling columns are primary key, unique index, and foreign key columns. Including all of these supplementally logged columns satisfies the requirements of both CDR and dependency computation in parallel Replicat processing.

`LOGALLSUPCOLS` causes Extract to do the following with these supplementally logged columns:

- Automatically includes in the trail record the before image for `UPDATE` operations.
- Automatically includes in the trail record the before image of all supplementally logged columns for both `UPDATE` and `DELETE` operations.

> **Note:**
>
> Certain columns cannot be part of supplemental logging in Oracle due to their data type. If you want those columns to be present in the trail file, even if they did not change, you must use `FETCHCOLS` in the Extract parameter file.

For Extract versions older than 12c, you can use `GETUPDATEBEFORES` and `NOCOMPRESSDELETES` parameters to satisfy the same requirement. See GETUPDATEBEFORES | IGNOREUPDATEBEFORES and COMPRESSUPDATES | NOCOMPRESSUPDATES for more information.

`LOGALLSUPCOLS | NOLOGALLSUPCOLS` takes precedence over the following parameters, if used:

- `GETUPDATEBEFORES | IGNOREUPDATEBEFORES`
- `COMPRESSDELETES | NOCOMPRESSDELETES`
- `COMPRESSUPDATES | NOCOMPRESSUPDATES` for before images, but `COMPRESSUPDATES | NOCOMPRESSUPDATES` takes precedence over `LOGALLSUPCOLS` on after images.

Default

`LOGALLSUPCOLS`

Syntax

`LOGALLSUPCOLS`

LOOK_AHEAD_TRANSACTIONS

Valid For

Parallel Replicat

Description

It controls how far ahead the Scheduler looks when batching transactions. The benefit and value to set this parameter at is highly dependent on the amount of data typically that has is in the trail files that Replicat is reading, and depends on the transaction mix. So, different values may work better or worse in a specific environment. Generally increment or decrement the parameter in 10,000 interval until the best throughput is achieved.

Default

The default value is 10000.

Syntax

```
LOOK_AHEAD_TRANSACTIONS
```

LOGOUT_RECV_TIMEOUT

Valid For

GLOBALS, Extract, Replicat, and Manager

Description

A new parameter `LOGOUT_RECV_TIMEOUT` is available from the Oracle GoldenGate 21c (21.3.0) release. Specify the amount of time OCI client waits for a response from database server when releasing the connection.

The time can be specified in any time units such milliseconds (default), seconds, minutes, and hours in the parameter file. Default value is 10000.

See *Oracle Database Net Services Reference* to learn about `SQLNET.RECV_TIMEOUT`.

Default

This parameter takes a value with default units as milliseconds and a default value of 10000.

Syntax

```
LOGOUT_RECV_TIMEOUT value
```

Example

```
LOGOUT_RECV_TIMEOUT=10s
```

LRSNTIMEDELTA

Valid For

Extract for DB2 z/OS

Description

Oracle GoldenGate extended LRSN and RBA support for DB2 z/OS. Because the data stored in DB2 increased exponentially, the LRSN and RBA value had to be increased in size from six bytes to ten bytes. The RBA value is adjusted automatically to match each log record's location in the log file. At this time DB2 z/OS has no way to adjust the LRSN timestamp value to the proper value automatically. DB2 z/OS uses a Store Clock to LRSN delta value internallly to adjust the LRSN timestamp. The Boot Strap Data Set (BSDS) utility report lists the LRSN

delta. The following three lines are an excerpt from the first page of the BSDS report showing the STCK TO LRSN DELTA value.

```
MAX RBA FOR TORBA          00000000000000000000
MIN RBA FOR TORBA          00000000000000000000
STCK TO LRSN DELTA         00000053402130000000
```

If the STCK TO LRSN DELTA value is 00000000000000000000, then no change to the Extract parameter file is needed. If there is a value other than zero, then the Extract uses a parameter with this DELTA value to automatically adjust the log record timestamps. This is done using the LRSNTIMEDELTA parameter and it is supplied in the Extract parameter file in a format similar to the following:

```
TRANLOGOPTIONS LRSNTIMEDELTA 00000053402130000000
```

For a normal run you would specify cut and paste the 20 byte delta value from your BSDS report into the parameter file Data Sharing group. If you are not using a Data Sharing group, this value will not exist in the BSDS report.

You will see this message in the Extract report with your delta value.

```
2019-01-23 12:24:10  INFO    OGG-25226  The Extract is using LRSN delta
value: 00000030214053000000
This delta is used to adjust operation timestamp values.
```

MACRO

Valid For

Extract and Replicat

Description

Use the MACRO parameter to create an Oracle GoldenGate macro. See Using Macros for more information about using macros, including how to invoke them properly.

Default

None

Syntax

The following must be used in the order shown:

```
MACRO #macro_name
PARAMS (#param_name [, ...])
BEGIN
macro_body
END;
```

MACRO
Starts the macro specification.

#
The macro character. Macro and parameter names must begin with a macro character. Anything in the parameter file that begins with the macro character is assumed to be either a macro or a macro parameter.
The default macro character is the pound (#) character, as in the following examples:

```
MACRO #macro1
PARAMS (#param1, #param2)
```

You can change the macro character with the `MACROCHAR` parameter.

macro_name
The name of the macro. Macro names must be one word with alphanumeric characters (underscores are allowed) and are not case-sensitive. Each macro name in a parameter file must be unique. Do not use quotes, or else the macro name will be treated as text and ignored.

PARAMS
Starts a parameter clause. A parameters clause is optional. The maximum size and number of parameters is unlimited, assuming sufficient memory is available.

param_name
Describes a parameter 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.
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.

BEGIN
Begins the macro body. Must be specified before the macro body.

macro_body
The body of the macro. The size of the macro body is unlimited, assuming sufficient available memory. 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. The semicolon is required to complete the definition.

Examples

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

Description

Use the `MACROCHAR` parameter to change the macro character of a macro definition to something other than the # character. You might need to change the macro character when, for example, table names include the # character.

The `MACROCHAR` parameter can only be used once in the parameter file. Place the `MACROCHAR` parameter before the first `MACRO` parameter in the parameter file. Anything in the parameter file that begins with the specified macro character is assumed to be either a macro or a macro parameter. All macro definitions in the parameter file must use the specified character.

`MACROCHAR` cannot be used with query parameters.

See also "MACRO".

See the Using Macros for more information about using macros.

Default

`#` (pound symbol)

Syntax

```
MACROCHAR character
```

character
The character to be used as the macro character. Valid user-defined macro characters are letters, numbers, and special characters such as the ampersand (`&`) or the underscore (_).

Example

In the following example, `$` is defined as the macro character.

```
MACROCHAR $
MACRO $mymac
PARAMS ($p1)
BEGIN
col = $p1
END;
```

MAP for Extract

Valid For

Extract

Description

You can also use `MAP` in an Extract parameter file to change the name of the transactions that Oracle GoldenGate stores for the table.

For example, consider that you capture the table `scott.emp`. For the first use case, you capture it to apply all the changes to anther table called `scott.emp` with the same structure, but you also want to capture `scott.emp` with a different set of columns and replicate that to a table called `scott.emp_old`. To make this work, you'll need an Extract parameter file similar to this:

```
TABLE scott.emp;
MAP scott.emp,
cols(emp_no, employee_name),
target scott.emp_old;
```

In the Replicat, you can do the following:

```
MAP *.*, target *.*;
```

And the changes from `scott.emp` would go into `scott.emp`, the trail data for `scott.emp_old` would go into `scott.emp_old`.

Use the `MAP` parameter for Extract when Extract is operating in classic capture mode and you need to use the `ALTID` component of this parameter to map an object ID to an object name. `ALTID` specifies the correct object ID if Extract is capturing from Oracle transaction logs that were generated by a database other than the one to which Extract is connected. This configuration is required when Extract is not permitted to connect directly to the production (source) database to capture production transactions.

When Extract cannot connect directly to a source database, it connects to a live standby or other facsimile database, but it reads transaction logs that are sent from the source database. By querying the catalog of the alternate database, Extract can get the metadata that it needs to expand the transaction data into valid SQL statements, but it cannot use the object ID from this query. The local object ID for a table is different from the object ID of that table in the source database (and, thus, in the transaction log). You must manually map each table name to the source object ID by using a `MAP` statement with `ALTID`.

To Use MAP with ALTID

- Create one `MAP` statement with `ALTID` for each table that you want to capture. Wildcarded table names are not allowed for a `MAP` parameter that contains `ALTID`.

- To specify other processing for the same table (or tables), such as data filtering or manipulation, you must also create a `TABLE` statement for each of those tables. Wildcarding can be used to specify multiple tables with one `TABLE` statement, if appropriate.
- Use a regular Replicat `MAP` statement in the Replicat parameter file, as usual. `MAP` for Extract does not substitute for `MAP` for Replicat, which is required to map source tables to target tables.
- DDL capture and replication is not supported when using `ALTID`.

Default

None

Syntax

```
MAP [container.]schema.table, ALTID object_ID [, object_ID]
```

[container.]schema.table
The fully qualified name of the source table.

object_ID
The object ID of the table as it exists in the production (source) database.
If a table is partitioned, you can list the object IDs of the partitions that you want to replicate, separating each with a comma.

Examples

Example 1
This example maps a non-partitioned table or just one partition of a partitioned table.

```
MAP QASOURCE.T2, ALTID 75740;
```

Example 2
This example maps partitions of a partitioned table.

```
MAP QASOURCE.T_P1, ALTID 75257,75258;
```

MAP

See "TABLE | MAP".

MAPALLCOLUMNS| NOMAPALLCOLUMNS

Valid For

Valid as a standalone Replicat parameter or as an option to `MAP`.

Description

An Extract or Replicat checks if all source columns are mapped directly to the target, without using the column mapping function when `MAPALLCOLUMNS` parameter is specified. If any source column is not mapped, then the Extract or Replicat abends.

> **Note:**
>
> This parameter is mainly used when performing heavy transformation using the Replicat, and you must ensure that all the data is available for transformation.

MAPALLCOLUMNS and NOMAPALLCOLUMNS can be used in two different ways. When specified at a global level, one parameter remains in effect for all subsequent MAP statements, until the other parameter is specified. When used within a MAP statement, they override the global specifications.

Default

```
NOMAPALLCOLUMNS
```

Syntax

```
MAPALLCOLUMNS | NOMAPALLCOLUMNS
```

Examples

Example 1
This example enables MAPALLCOLUMNS for some MAP statements while disabling it for others.

```
MAPALLCOLUMNS
MAP hr.emp, TARGET hr.emp2;
NOMAPALLCOLUMNS
MAP hr.dep, TARGET hr.dep2;
```

Example 2
This example shows a combination of global and MAP-level use of MAPALLCOLUMNS. The MAP specification overrides the global specification for the specified table.

```
NOMAPALLCOLUMNS
MAP hr.dep, TARGET hr.dep2;
MAP hr.emp, TARGET hr.emp2, MAPALLCOLUMNS ;
```

MAP_PARALLELISM

Valid for

Parallel Replicat

Description

Configures number of mappers. It controls the number of threads used to read the trail file. The minimum value is 1, maximum value is 100 and the default value is 2. Increasing this parameter directly impacts the trail files. On a physical disk drive, you will start seeing IO contention if you use more than 4 or 5 mappers. If it's in solid state, you can increase the value a bit higher before seeing IO contention. However, this depends on how much IO contention you need.

Syntax

```
MAP_PARALLELISM value
```

Examples

```
MAP_PARALLELISM 3
```

MAPEXCLUDE

Valid For

Replicat

Description

Use the `MAPEXCLUDE` parameter with the `MAP` parameter to explicitly exclude source tables and sequences from a wildcard specification. You can use multiple `MAPEXCLUDE` statements for specific `MAP` statements.

`MAPEXCLUDE` is evaluated before evaluating the associated `MAP` parameters. Thus, the order in which they appear does not make a difference.

When using wildcards, be careful not to place them such that all objects are excluded, leaving nothing to process. For example, the following example captures nothing from `cat1`:

```
MAP cat1.schema*.tab*, TARGET schema*.tab*;
MAPEXCLUDE cat1.*.*
```

See also the EXCLUDEWILDCARDOBJECTSONLY parameter.

The default for resolving wildcards is `WILDCARDRESOLVE DYNAMIC`. Therefore, if a table that is excluded with `MAPEXCLUDE` is renamed to a name that satisfies a wildcard, the data will be captured. The `DYNAMIC` setting enables new table names that satisfy a wildcard to be resolved as soon as they are encountered and included in the Oracle GoldenGate configuration immediately. For more information, see WILDCARDRESOLVE.

Default

None

Syntax

```
MAPEXCLUDE [container.]owner.{table | sequence}
```

container.
If the source database requires three-part names, specifies the name or wildcard specification of the Oracle container that contains the object to exclude.

owner
Specifies the name or wildcard specification of the owner, such as the schema, of the object to exclude.

table | sequence
The name or wildcard specification of the source object to exclude.

Example 1

In this example, the source tables from catalog `pdb1` with schema `test` beginning with `tab` and the source table `pdb2.fin.acct` are excluded from the trail files:

```
MAPEXCLUDE pdb1.test.tab*
MAP pdb1.*.*, TARGET *.*;
MAPEXCLUDE pdb2.fin.acct
MAP pdb2.*.*, TARGET *.*;
```

Example 2

The following example excludes all source tables from catalog beginning with `pdb`, that is, it excludes all tables from `pdb1`, `pdb2`, `pdb3` and so on:

```
MAP pdb1.*.*, TARGET *.*;
MAP pdb2.*.*, TARGET *.*;
MAPEXCLUDE pdb1.test.tab*
MAPEXCLUDE pdb*.*.*
MAPEXCLUDE pdb2.fin.acct
```

MAPINVISIBLECOLUMNS | NOMAPINVISIBLECOLUMNS

Valid For

Replicat on Oracle. Valid as a standalone parameter or as an option to `MAP`.

Description

Use `MAPINVISIBLECOLUMNS` and `NOMAPINVISIBLECOLUMNS` to control whether or not Replicat includes invisible columns in Oracle target tables for default column mapping. For invisible columns in Oracle target tables that use explicit column mapping, they are always mapped so do not require this option. `MAPINVISIBLECOLUMNS` is required to Automatic Conflict Detection and Resolution.

`MAPINVISIBLECOLUMNS` and `NOMAPINVISIBLECOLUMNS` can be used in two different ways. When specified at a global level, one parameter remains in effect for all subsequent `MAP` statements, until the other parameter is specified. When used within a `MAP` statement, they override the global specifications.

Default

For integrated Replicat or parallel integrated Replicat the default value is `MAPINVISIBLECOLUMNS` unless you explicitly specify `NOMAPINVISIBLECOLUMNS` in the Replicat parameter file.

For all other types of Replicat, the default is `NOMAPINVISIBLECOLUMNS`.

Syntax

```
MAPINVISIBLECOLUMNS | NOMAPINVISIBLECOLUMNS
[, THREAD (threadID[, threadID][, ...][, thread_range[, thread_range][, ...])]
```

`THREADS (`*`threadID`*`[, `*`threadID`*`][, ...][, `*`thread_range`*`[, `*`thread_range`*`][, ...])`
Specifies `MAPINVISIBLECOLUMNS | NOMAPINVISIBLECOLUMNS` only for the specified thread or threads of a coordinated Replicat.

***`threadID`*`[, `*`threadID`*`][, ...]`**
Specifies a thread ID or a comma-delimited list of threads in the format of `threadID, threadID, threadID`.

[, *thread_range*[, *thread_range*][, ...]
Specifies a range of threads in the form of `threadIDlow-threadIDhigh` or a comma-delimited list of ranges in the format of `threadIDlow-threadIDhigh`, `threadIDlow-threadIDhigh`.

A combination of these formats is permitted, such as `threadID`, `threadID`, `threadIDlow-threadIDhigh`.

Examples

Example 1
This example enables `MAPINVISIBLECOLUMNS` for some `MAP` statements while disabling it for others.

```
MAPINVISIBLECOLUMNS
MAP hr.emp, TARGET hr.emp2;
NOMAPINVISIBLECOLUMNS
MAP hr.dep, TARGET hr.dep2;
```

Example 2
This example shows a combination of global and `MAP`-level use of `MAPINVISIBLECOLUMNS`. The `MAP` specification overrides the global specification for the specified table.

```
NOMAPINVISIBLECOLUMNS
MAP hr.dep, TARGET hr.dep2;
MAP hr.emp, TARGET hr.emp2, MAPINVISIBLECOLUMNS;
```

Example 3
In this example, `MAPINVISIBLECOLUMNS` is enabled globally, but turned off for thread 3. The remaining threads 1, 2, and 4 will include invisible target columns in default column mapping.

```
MAPINVISIBLECOLUMNS
NOMAPINVISIBLECOLUMNS THREAD(3)
MAP hr.dep, TARGET hr.dep2, THREADRANGE(1, 4);
MAP hr.emp, TARGET hr.emp2, THREADRANGE(1, 4);
```

MASTERKEYNAME

Valid For

GLOBALS

Description

`MASTERKEYNAME` controls the name of the masterkey that Oracle GoldenGate processes in a deployment will use to retrieve the key from the wallet. If no masterkey is provided, the default value is `OGG_DEFAULT_MASTERKEY`. The non-mandatory option `VERSION` takes one number between 1 and 65535 (0xffff). When present, it forces the Oracle GoldenGate processes in the deployment to use that particular version of the masterkey to encrypt or decrypt trails. This is not needed during normal operation, but might be useful when debugging old trail files if the key has been rolled over since the date the old trail was created.

Default

`OGG_DEFAULT_MASTERKEY`

Syntax

```
MASTERKEYNAME [VERSION]
```

MAXDISCARDRECS

Valid For

Extract and Replicat

Description

Use the MAXDISCARDRECS parameter to limit the number of errors that are 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.

More than one instance of MAXDISCARDRECS can be used in a parameter file to specify different maximums for different sets of MAP statements. An instance of MAXDISCARDRECS applies to all subsequent MAP statements until the next instance of MAXDISCARDRECS is encountered. The minimum is 0.

Default

None

Syntax

```
MAXDISCARDRECS number
```

number
The maximum number of errors to report.

Example

```
MAXDISCARDRECS 1000
```

MAXGROUPS

Valid For

GLOBALS

Description

Use the MAXGROUPS parameter to specify the maximum number of process groups that can run in an instance of Oracle GoldenGate. Oracle GoldenGate process checks this parameter to control the maximum number of groups that it allows to be created.

Each Replicat thread in a coordinated Replicat group is considered to be a *group* in the context of MAXGROUPS. Therefore, the value of the MAXTHREADS option of COORDINATED in the ADD REPLICAT command (default is 25), plus the number of other Replicat and Extract groups in the Oracle GoldenGate instance, cannot exceed the MAXGROUPS value, or ADD REPLICAT returns an error.

The actual number of processes that can run on a given system depends on the system resources that are available. If those resources are exceeded, Oracle GoldenGate returns errors regardless of the setting of `MAXGROUPS`.

Default

1000 groups

Syntax

```
MAXGROUPS number
```

number
The number of groups allowed in one instance of Oracle GoldenGate. Valid values are from 1000 to 5000.

Example

```
MAXGROUPS 1500
```

MAXSQLSTATEMENTS

Valid For

Replicat

Description

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. 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 that other applications and processes use. Before changing `MAXSQLSTATEMENTS`, contact Oracle Support.

When setting `MAXSQLSTATEMENTS` for a coordinated Replicat, take into account that the specified maximum number of cursors is applied to each thread in the configuration, not as an aggregate threshold for Replicat as a whole. For example, if `MAXSQLSTATEMENTS 100` is specified, it is possible for each thread to have 99 open cursors without any warning or error from Replicat.

See "BATCHSQL" for more information about `BATCHSQL` mode.

Default

250 cursors

Syntax

```
MAXSQLSTATEMENTS number
```

number
The maximum number of cursors that Replicat (or each thread in a coordinated Replicat) can use. Valid values are from 1 to 250.

Example

```
MAXSQLSTATEMENTS 200
```

MAXTRANSOPS

Valid For

Replicat (Not supported in integrated and parallel Replicat mode)

Description

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.

To use `MAXTRANSOPS` is to alter the transactional boundaries that are imposed by the source application, even though Replicat applies the operations in the correct order. This can cause errors if Extract fails during that transaction. Extract rewrites the transaction to the end of the trail, instead of overwriting the old one. Because the trail is sequential, Replicat starts processing the old transaction and must roll it back when it receives the recovery marker and the new transaction, and then start applying the new transaction. If `MAXTRANSOPS` caused Replicat to split the original transaction into multiple smaller transactions, Replicat may only be able to roll back the portion that was not committed to the target. When Replicat processes the committed operations again, they will result in duplicate-row errors or missing-row errors, depending on the SQL operation type. The minimum is 1.

> **Note:**
>
> When troubleshooting Replicat abend errors, Oracle Support may request `GROUPTRANSOPS` to be set to 1 and `MAXTRANSOPS` to be set to 1. This is only a temporary configuration for troubleshooting purposes and should not be used permanently in production, or it will cause data integrity errors.

Default

10,000,000

Syntax

```
MAXTRANSOPS number
```

number
The number of operations to portion into a single transaction group.

Example

```
MAXTRANSOPS 10000
```

MGRSERVNAME

Valid For

GLOBALS

Description

Use the MGRSERVNAME parameter in a GLOBALS parameter file to specify the name of the Manager process when it is installed as a Windows service. This parameter is only required when installing multiple instances of Manager as a service on the same system, for example when installing multiple Oracle GoldenGate instances or when also installing the Oracle GoldenGate Veridata Agent, which uses a Manager process.

There must be a GLOBALS file containing MGRSERVNAME for each Manager service that is installed with the INSTALL utility. The files must be created before the services are installed, because the installation program refers to MGRSERVNAME when registering the service name on the system.

Default

None

Syntax

```
MGRSERVNAME name
```

name
A one-word name for the Manager service.

Example

```
MGRSERVNAME Goldengate
```

NAMECCSID

Valid for

GLOBALS, Extract, Replicat, DEFGEN for DB2 on IBM i

Description

Use the NAMECCSID parameter to specify the CCSID (coded character set identifier) of the database object names stored in the SQL catalog tables. The SQL catalog tables are created with the CCSID of the system, but the actual database object names could be represented in the catalog with characters from a different CCSID. The catalog does not indicate this difference when queried, and therefore Oracle GoldenGate could retrieve the name incorrectly unless NAMECCSID is present to supply the correct CCSID value.

To set the CCSID session, use the SET NAMECCSID command.

To view the current CCSID, use the SHOW command. If the CCSID is not set through the Admin Client session or through the parameter NAMECCSID, the SHOW value will be DEFAULT.

Default

```
DEFAULT
```

Syntax

```
NAMECCSID {CCSID | DEFAULT}
```

CCSID
A valid DB2 for i coded character set identifier that is to be used for object names in catalog queries.

DEFAULT
Indicates that the system CCSID is to be used for object names in catalog queries.

Example

```
NAMECCSID 1141
```

NAMEMATCH parameters

Valid For

GLOBALS

Description

Use the `NAMEMATCH` parameters to control the behavior of fallback name mapping. Fallback name mapping is enabled by default when the source database is case-sensitive and the target database support both case-sensitive and case-insensitive object names, such as Oracle and DB2 LUW.

By default, `NAMEMATCHIGNORECASE` fallback name matching works as follows: When a source table name is case-sensitive, Oracle GoldenGate applies case-sensitive wildcard mapping on the target database to find an exact match. If the target database does not contain the exact target table name, including case, fallback name mapping performs a case-insensitive target table mapping to find a name match.

Default

```
NAMEMATCHIGNORECASE
```

Syntax

```
NAMEMATCHIGNORECASE | NAMEMATCHNOWARNING | NAMEMATCHEXACT
```

NAMEMATCHIGNORECASE
Performs a case-insensitive target table mapping to find a name match when the target database does not contain the exact target table name, including case.

NAMEMATCHNOWARNING
Outputs a warning message to the report file when fallback name matching is used.

NAMEMATCHEXACT
Disables fallback name mapping. If an exact, case-sensitive match is not found, Oracle GoldenGate returns an error and abends.

NLS_LENGTH_SEMANTICS

Valid For

Extract and Replicat

Description

Use NLS_LENGTH_SEMANTICS parameter for Extract or Replicat to switch index values between byte position and character position. For example, the function @STRNCMP(col1, col2, 3) compares the first three bytes if BYTE semantics is specified, but compares the first three characters if CHAR semantics is specified.

Functions affected:

- @STRFIND
- @STRNCMP

Using NLS_LENGTH_SEMANTICS causes the following column mapping functions to returns the number of bytes or number characters.

- @STRLEN

The following functions always work in CHAR semantics to prevent truncation in the middle of multibyte characters.

- @STRNCAT
- @STREXT

Default

Byte

Syntax

```
NLS_LENGTH_SEMANTICS [BYTE | CHAR]
```

Example

The following forces semantics to CHAR:

```
NLS_LENGTH_SEMANTICS CHAR
```

The following forces semantics to BYTE, which is the default:

```
NLS_LENGTH_SEMANTICS BYTE
```

NOCATALOG

Valid For

DEFGEN

Description

Use NOCATALOG in the DEFGEN parameter file to remove the Oracle Database container name from table names before their definitions are written to the definitions file. This parameter is valid if the database supports container names or catalog names and the DEFSFILE parameter includes the FORMAT RELEASE option set to 12.1. Use this parameter if the definitions file is to be used for mapping to a database that only supports two-part names (*owner.object*).

DEFGEN abends with an error if duplicate *schema.table* names are encountered once the container or catalog names are removed. This prevents the possibility of processing errors

caused by different sets of metadata having the same *schema.table* name when there is no catalog name to differentiate them.

Default

None

Syntax

```
NOCATALOG
```

NODUPMSGSUPPRESSION

Valid For

GLOBALS

Description

Use NODUPMSGSUPPRESSION to prevent the automatic suppression of duplicate informational and warning messages in the report file, the error log, and the system log files. A message is issued to indicate how many times a message was repeated.

Default

Automatically suppress duplicate messages.

Syntax

```
NODUPMSGSUPPRESSION
```

NUMFILES

Valid For

Extract and Replicat

Description

Use the NUMFILES parameter to control the initial number of memory structures that are allocated to contain information about tables specified in TABLE or MAP statements. NUMFILES must occur before any TABLE or MAP entries, and before the SOURCEDEFS or TARGETDEFS parameter, to have any effect.

When setting NUMFILES for a coordinated Replicat, take into account that the specified value is applied to each thread in the configuration, not as an aggregate threshold for Replicat as a whole. For example, if NUMFILES 500 is specified, it is possible for each thread to have 499 initial memory structures without any warning or error from Replicat.

To control the number of additional memory structures that are allocated dynamically once the NUMFILES value is reached, use the ALLOCFILES parameter. See "ALLOCFILES" for more information. The default values should be sufficient for both NUMFILES and ALLOCFILES, because memory is allocated by the process as needed, system resources permitting. The minimum is 1 and the maximum is 20000. While the maximum values is 20000, Oracle GoldenGate can support up to 2 million tables in a single Replicat or Extract.

Default

1000

Syntax

```
NUMFILES number
```

number
The initial number of memory structures to be allocated. Do not set `NUMFILES` to an arbitrarily high number, or memory will be consumed unnecessarily.

Example

```
NUMFILES 50
```

OBEY

Valid For

Extract and Replicat

Description

Use the `OBEY` parameter to retrieve parameter settings from a file other than the current parameter file.

To use `OBEY`, create and save a parameter file that contains the parameters that you want to retrieve. This is known as an `OBEY` file. You can create a library of `OBEY` files that contain different, frequently used parameter settings. Then, use the `OBEY` parameter in the active parameter file to invoke the parameters in the `OBEY` file.

Upon encountering an `OBEY` parameter in the active parameter file, Oracle GoldenGate processes the parameters from the `OBEY` file and then returns to the active parameter file to process any remaining parameters.

`OBEY` statements cannot be nested within other `OBEY` statements.

Instead of using `OBEY`, or in addition to it, you can use Oracle GoldenGate macros to retrieve frequently used parameters. Refer to Using OBEY in *Oracle GoldenGate Microservices Documentation*.

Default

None

Syntax

```
OBEY file
```

file
The relative or fully qualified name of the file from which to retrieve parameters or commands.

Example

```
OBEY /home/ogg/myparams
```

OUTPUTFILEUMASK

Valid For

GLOBALS

Description

Use the `OUTPUTFILEUMASK` parameter to specify an octal umask for Oracle GoldenGate processes to use when creating all files. `OUTPUTFILEUMASK` is not valid for Windows systems.

Default

Umask of 027 (all privileges)

Syntax

```
OUTPUTFILEUMASK umask
```

umask
The umask value. Must be between 0 and 077; otherwise there will be an error: `Missing or invalid option for OUTPUTFILEUMASK.`

Example

```
OUTPUTFILEUMASK 066
```

OUTPUTFORMAT

Valid For

Extract

Description

Use the `OUTPUTFORMAT` parameter to output data in text, SQL, and XML formats.

Default

None

Syntax

```
OUTPUTFORMAT format_type  [, option]  [, ...]
```

OUTPUTFORMAT TEXT

Use the `TEXT format_type` to output data in external text format instead of the default Oracle GoldenGate canonical format. 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.

This type of 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, `OUTPUTFORMAT TEXT` will not take effect.

`option` can be *one* of the following:

INCLUDE (HEARTBEAT)
Includes the heartbeat table records. By default, the heartbeat table records are ignored.

BCP
Formats the output for compatibility with SQL Server's Bulk Copy Program and other bulk load utilities.
The following options are ignored when the `BCP` option is specified:

- `NAMES | NONAMES` — Specifies whether or not to include column names. `NAMES` is the default.
- `NULLISSPACE` — Output NULL columns as empty columns. Without `NULLISSPACE`, NULL columns are output as NULL.
- `PLACEHOLDERS` — Outputs placeholder for missing columns.
- `NOHDRFIELDS` — Does not include any metadata, such as the before and after indicator, and transaction information. Outputs column data only.
- `DELIMITER 'delimiter'` — Specifies the field delimiter character. To specify tabulation, use `TAB`. The default is a comma ‘,’.
- `OP | _NOOP` — Specifies whether or not to include operation type indicator (I, D, U, V). `OP` is the default.
- `IND | _NOIND` — Specifies whether or not to include the before and after image indicator (B or A). `IND` is the default.
- `_TRANSTMTS | _NOTRANSTMTS` — Specifies whether or not to include transaction information. `_TRANSTMTS` is the default.
- `_WHOLEFILE` — Includes the fully-qualified object name including the schema name.
- `_FILE` — Includes the object name only.

SQLLOADER
Produces a fixed-length text formatted file that is compatible with the Oracle SQL*Loader utility or the IBM load utility.

DATE | TIME | TS
Specifies the record timestamp precision to output. By default, this parameter does not output record timestamp. You can use 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 microseconds).

SQLLOADER
Produces a fixed-length, ASCII-formatted file that is compatible with the Oracle SQL*Loader utility or the IBM Load Utility program.

OUTPUTFORMAT SQL

Use the `OUTPUTFORMAT SQL` parameter to output data in external SQL format, instead of the default Oracle GoldenGate canonical format. `OUTPUTFORMAT SQL` generates SQL statements (`INSERT`, `UPDATE`, `DELETE`) that can be applied to SQL tables by utilities other than Oracle GoldenGate Replicat.

INCLUDE (HEARTBEAT)
Includes the heartbeat table records. By default, heartbeat table records are ignored.

ENCODING ***encoding***
Outputs the SQL format file in the specified encoding. Oracle GoldenGate character set names are supported. By default, is current operating system character set. No character set conversion on column data is performed with the default character set.
The following options specify the specific output format. The options are exclusive so cannot be specified together.

ORACLE
Formats records for compatibility with Oracle Databases by converting date and time columns to a format accepted by SQL*Plus.

SQLPLUS
Formats records for compatibility with Oracle Databases by converting date and time columns to a format accepted by SQL*Plus.

SQLLOADER
Produces a fixed-length text formatted file that is compatible with the Oracle SQL*Loader utility or the IBM load utility program.
This is exactly the same as `OUTPUTFORMAT TEXT SQLLOADER`, which Oracle recommends that you use..

_TRANSTMTS | _NOTRANSTMTS
Includes SQL transaction information as comment. `_NOTRANSTMTS` is the default.
For example:

```
—B, 2016-07-09:09:9:21.000000,1357991461,627
```

WHOLEFILE
Includes the fully-qualified object name including the schema name.

FILE
Includes the object name only.

NOPKUPDATES
Formats `PKUPDATE` and `UNIFIED UPDATE` operations as a pair of `DELETE` and `INSERT` operations. `PKUPDATE` and `UNIFIED UPDATE` operations are formatted as an `UPDATE` operation if the option is not specified. This option is ignored if the `SQLLOADER` option is used.

OUTPUTFORMAT XML

Use the `OUTPUTFORMAT XML` parameter to output data in XML format, instead of the default Oracle GoldenGate canonical format. An `OUTPUTFORMAT XML` statement affects all Extract files or trails that are defined after it. By default, the XML is output in the character set of the local operating system.

XML stored as CLOB or BLOB is output up to 4000 bytes. To include larger XML stored as BLOB or CLOB, use the `ENCODING` option.

XML stored as CLOB is always output in a `CDATA` section regardless of its size. This is to avoid the overhead of converting reserved characters such as `<`, `>` and `&` to the appropriate XML representation.

Binary data including BLOB are encoded as Base64, which represents binary data in an ASCII string format and allows output to XML.

The XML, the database object names, such as table and column names, and `CHAR` and `VARCHAR` data are written in the default character set of the operating system unless the `ENCODING` option is used to output in UTF-8.

INCLUDE (HEARTBEAT | LOB | USERTOKEN)
Includes the heartbeat table records. LOB more than 4000 bytes and Oracle GoldenGate user tokens.
By default, heartbeat table records are ignored and doesn't include LOB more than 4000 bytes and user tokens.
BLOB more than 4000 bytes is encoded in Base64, and CLOB more than 4000 bytes is formatted in a `CDATA` section.

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.

CLOSETRANS | NOCLOSETRANS
Forces the closure of opened transaction boundaries and commits the timestamp upon rollover. It adds same transaction boundaries and commit timestamp tags to the next XML file after rollover.
The option is ignored if the `TRANS` option is not specified.

ENCODING xml_encoding
Outputs an XML file in the specified encoding. The default is UTF-8. The following MIME encoding names are supported.

UTF-8	ISO-10646 UTF-8, surrogate pairs are 4 bytes per character
UTF-16	ISO-10646 UTF-16
windows-1250	Windows Central Europe
windows-1251	Windows Cyrillic
windows-1252	Windows Latin-1
windows-1253	Windows Greek
windows-1254	Windows Turkish
windows-1255	Windows Hebrew
windows-1256	Windows Arabic
windows-1257	Windows Baltic
windows-1258	Windows Vietnam
windows-874	Windows Thai
IBM437	DOS Latin-1
IBM775	DOS 775, Baltic
IBM850	DOS multilingual
cp851	DOS Greek-1
IBM852	DOS Latin-2
IBM855	DOS Cyrillic
IBM857	DOS Turkish
IBM00858	DOS Multilingual with Euro
IBM860	DOS Portuguese
IBM861	DOS Icelandic
IBM862	DOS Hebrew
IBM863	DOS French
IBM864	DOS Arabic
IBM865	DOS Nordic
IBM866	DOS Cyrillic / GOST 19768-87
IBM868	DOS Urdu
IBM869	DOS Greek-2
ISO-8859-1	ISO-8859-1 Latin-1/Western Europe
SO-8859-2	ISO-8859-2 Latin-2/Eastern Europe

ISO-8859-3	ISO-8859-3 Latin-3/South Europe
ISO-8859-4	ISO-8859-4 Latin-4/North Europe
ISO-8859-5	ISO-8859-5 Latin/Cyrillic
ISO-8859-6	ISO-8859-6 Latin/Arabic
ISO-8859-7	ISO-8859-7 Latin/Greek
ISO-8859-8	ISO-8859-8 Latin/Hebrew
ISO-8859-9	ISO-8859-9 Latin-5/Turkish
ISO-8859-10	ISO-8859-10 Latin-6/Nordic
ISO-8859-13	ISO-8859-13 Latin-7/Baltic Rim
ISO-8859-14	ISO-8859-14 Latin-8/Celtic
ISO-8859-15	ISO-8859-15 Latin-9/Western Europe
ISO-8859-16	ISO-8859-16, Latin-10, South Eastern Europe
KOI8-R	KOI8-R, Russian
KOI8U	KOI8-U, Ukranian
TIS-620	Thai Industrial Standard 620-2533
DEC-MCS	DEC Multilingual
hp-roman8	HP Latin-1 Roman8
Shift_JIS	Shift_JIS, Windows-932
GBK	GBK, Windows-936
KSC_5601	KSC-5601, Windows-949
Big5	Big-5 Traditional Chinese, Windows-950
EUC-JP	EUC Japanese
GB2312	GB-2312-1980
EUC-KR	EUC Korean
GB18030	GB-18030
HZ-GB-2312	HZ GB-2312
Big5-HKSCS	Big-5, HongKong extension

Example

```
OUTPUTFORMATXML NOINLINEPROPERTIES, NOTRANS
```

OVERRIDEDUPS | NOOVERRIDEDUPS

Valid For

Replicat

Description

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 where you do not want to truncate target tables prior to the load, or for the resynchronization of a target table with a trusted source. Use the `SQLDUPERR` parameter with `OVERRIDEUPS` to specify the numeric error code that is returned by the database for duplicate `INSERT` operations. See "SQLDUPERR" for more information.
- `NOOVERRIDEDUPS`, the default, generates a duplicate-record error instead of overwriting the existing record. You can use an exceptions `MAP` statement with a `SQLEXEC` clause to initiate a response to the error. Otherwise, the transaction may abend.

  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 *`error`* is the database error number for primary key constraint errors.

```
REPERROR (error, IGNORE)
```

For example, the statement for an Oracle database would be:

```
REPERROR (1, IGNORE)
```

Replicat writes ignored duplicate records to the discard file.

Place OVERRIDEDUPS or NOOVERRIDEDUPS before the TABLE or MAP statements that you want it to affect. You can create different rules for different groups of TABLE or MAP statements. The parameters act as toggles: one remains in effect for subsequent TABLE or MAP statements until the other is encountered.

OVERRIDEDUPS is enabled automatically when HANDLECOLLISIONS is used. See "HANDLECOLLISIONS | NOHANDLECOLLISIONS" for more information.

> **WARNING:**
>
> When OVERRIDEDUPS is in effect, records might not be processed in chronological order across multiple Replicat processes.

Default

NOOVERRIDEDUPS

Syntax

```
OVERRIDEDUPS | NOOVERRIDEDUPS
```

PARTITION | PARTITIONEXCLUDE

Valid For

Extract, Distribution Service, and Replicat. Oracle only.

Description

These parameters work in conjunction with the TABLE and TABLEEXCLUDE parameters. Only when a table is included, the partition rules are evaluated.

For consistency, these parameters behave much like their TABLE and TABLEEXCLUDE counterparts.

- Wildcarding will be allowed in all name portions.
- GLOBALS parameter EXCLUDEWILDCARDOBJECTSONLY is supported
- Container portion is only valid in CDB environment in the three-part name.
- SOURCECATALOG parameter will take affect when catalog portion is not specified.
- Container portion must be specified when SOURCECATALOG is not specified.
- If container portion is specified, then it takes precedence over SOURCECATALOG.

If the [container.]schema.table portion of any PARTITION or PARTITIONEXCLUDE rule matches the table, only then additional partition filtering will be performed.

Partition filtering rules are evaluated in the following order:

- If the partition name does not match any `PARTITION` parameter, it is excluded.
- If included by the `PARTITION` parameter, then exclusion rules are evaluated unless it was included by a non-wildcard inclusion rule and `EXCLUDEWILDCARDOBJECTSONLY` was specified.

If using `PARTITION` or `EXCLUDEPARTITION` in Replicat, then the `PARTITION` parameter must be used for the Extract `TABLE` parameter to write the partition metadata into the trail file so that Replicat can process it.

> **Note:**
>
> An error occurs if a `PARTITION` or `PARTITIONEXCLUDE` parameter has an invalid number of parts.

Syntax

```
PARTITION [container.]schema.table.partition;
PARTITIONEXCLUDE [container.]schema.table.partition;
```

For non-CDB, 3 parts must be specified (`schema.table.partition`).

For CDB, either 4 parts must be specified (`pdb.schema.table.partition`) or 3 parts with a preceding `SOURCECATALOG` parameter.

Examples

In the following example with DML operations on partition `P_Q4` of table `SH.SALES`, the partition is included because both table and partition rules include it.

```
TABLE
  sh.sales;
  PARTITION sh.sales*.p_q4;
```

In the following example with DML operations on partition `P_Q4` of table `SH.SALES`, all partitions are included on the Extract side using the `TABLE`/`PARTITION` parameters. The partition `P_Q4` is excluded at the Replicat side using the `MAP`/`PARTITIONEXCLUDE` parameters. All other changes on partitions are applied by Replicat.

```
TABLE sh.sales;
  PARTITION sh.sales.p_q*;


MAP sh.sales, TARGET sh.sales;
  PARTITION sh.sales.p_q4;
```

In the following example with DML operations on partition `P_Q4` of table `SH.SALES`, the partition `P_Q4` is excluded because it is only valid for the partition `P_Q3` of `SH.SALES`:

```
TABLE
   sh.sales;PARTITION
   sh.sales*.p_q3;
```

The following example with DML operations on partition `P_Q3` of `SH.SALES` shows how multiple partition rules can be specified. Partition will be included because it is matched by one of the partition inclusion rules.

```
TABLE
          sh.sales;PARTITION
          sh.sales.p_q3;PARTITION
          sh.sales.p_q4;
```

In the following example with DML operations on partition `P_Q4` of table `SH.SALES_HISTORY`, the partition `P_Q4` is excluded because of explicit partition exclude rule.

```
TABLE
  sh.sales_history
  PARTITION sh.sales*.p_q4;
  PARTITIONEXCLUDE sh.sales_history.p_q*;
```

In the following example with DML operations on partition `P_S1` of table `SH.PRODUCTS`, neither the partition `PART_S1` nor the complete table `SH.PRODUCTS` is captured because the table `SH.PRODUCTS` is not referenced.

```
TABLE
  sh.sales;PARTITION
  sh.products.p_s1;
```

In the following example with DML operations on partition `P_Q4` of table `SH.SALES`, the partition `PART_S1` is included because no partition rule has a table portion matching `SH.SALES`. Therefore, no partition rule is evaluated.

```
TABLE
  sh.sales;
  PARTITION sh.products.p_s1;
```

PTKMONITORFREQUENCY

Valid For

Extract, Replicat, and Manager

Description

Use `PTKMONITORFREQUENCY` to set the monitoring collection frequency interval.

Default

One second.

Syntax

```
PTKMONITORFREQUENCY seconds
```

seconds
Specifies the time interval, in seconds, for monitoring collection to occur. The minimum is 1 seconds and the maximum is 60 seconds.

Examples

```
PTKMONITORFREQUENCY 10
```

PRESERVETARGETTIMEZONE

Valid For

Replicat

Description

Use the `PRESERVETARGETTIMEZONE` parameter to override the default Replicat session time zone. By default, Replicat sets its session to the time zone of the source database, as written to the trail by Extract. `PRESERVETARGETTIMEZONE` causes Replicat to set its session to the time zone of the target database.

Default

None

Syntax

```
PRESERVETARGETTIMEZONE
```

PROCEDURE

This is an option that can be specified as a stand-alone statement in extract and replicat parameter file. It indicates which feature group of procedural calls will be replicated.

Valid For

Extract and Replicat in Oracle only. It also requires the database to Oracle 12.2 or higher, and requires Oracle GoldenGate and the database to be configured appropriately. See Configure Procedural Replication.

Syntax

```
PROCEDURE [ INCLUDE | EXCLUDE ]  FEATURE [ALL_SUPPORTED | feature_list]
```

Examples

Example 1
Include all system supplied packages:

```
PROCEDURE INCLUDE FEATURE ALL_SUPPORTED
```

Example 2
Include specific packages

```
PROCEDURE INCLUDE FEATURE AQ, FGA, DBFS
```

Example 3
Exclude a specific packages

```
PROCEDURE INCLUDE FEATURE ALL_SUPPORTED
PROCEDURE EXCLUDE FEATURE REDFINITION
```

REPERROR

Valid For

Replicat

Description

Use the `REPERROR` parameter to control how Replicat responds to errors. The default response of Replicat to any error is to abend.

You can use one `REPERROR` statement to handle most errors in a default manner, while using one or more other `REPERROR` statements to handle specific errors differently. For example, you can ignore duplicate-record errors but abend processing in all other cases.

You can use `REPERROR` globally (at the root of the parameter file) to affect all `MAP` statements that follow it, or you can use it within a `MAP` statement to affect the tables specified in that statement. Using `REPERROR` within a `MAP` statement gives you the ability to handle errors in a particular way for each thread of a coordinated Replicat.

Using Record-level Error Handling

All `REPERROR` options except `TRANSDISCARD` and `TRANSEXCEPTION` apply an error-handling action in response to an individual SQL operation on an individual record. Other, error-free records in the same transaction are processed as configured in the `MAP` statements and other parameters in the parameter file, as applicable.

Using Transaction-level Error Handling

The `TRANSDISCARD`, `TRANSEXCEPTION`, and `ABEND` options apply an error-handling action to an entire transaction. The triggering error can occur on an individual record in the transaction or on the commit operation. (Commit errors do not have a particular record associated with them.) These options can be used to:

- prevent an entire source transaction from being replicated to the target when any error is associated with it.
- respond to a commit error when deferred constraint checking is enabled on the target.

`TRANSDISCARD` and `TRANSEXCEPTION` are mutually exclusive.

Effect of Other Parameters on Transaction-level Options

`TRANSDISCARD` and `TRANSEXCEPTION` honor the boundaries of the source transaction; however, the presence of `BATCHSQL`, `GROUPTRANSOPS`, or `MAXTRANSOPS` in the parameter file may affect the error-handling logic or outcome, because they alter transaction boundaries.

Effect of BATCHSQL and GROUPTRANSOPS

BATCHSQL or GROUPTRANSOPS (the default) both group SQL operations from different transactions into larger transactions to improve performance, while maintaining transactional order. When these parameters are in effect and any error occurs, Replicat first tries to resolve it by entering an alternate processing mode (see the documentation for those parameters). If the error persists, TRANSDISCARD or TRANSEXCEPTION comes into effect, and Replicat reverts to source-processing mode as follows:

1. It rolls back the grouped or arrayed transaction.
2. It replays the offending transaction one SQL operation at a time, using the same transaction boundaries as the source transaction.
3. It performs the discard logic (TRANSDISCARD) or exceptions-mapping (TRANSEXCEPTION). (See those option descriptions for more detail.)
4. It resumes BATCHSQL or GROUPTRANSOPS mode after the TRANSDISCARD error handling is completed.

Effect of MAXTRANSOPS

The integrity of TRANSDISCARD and TRANSEXCEPTION transaction-level error handling can be adversely affected by the setting of the MAXTRANSOPS parameter. MAXTRANSOPS causes Replicat to split very large replicated source transactions into smaller transactions when it applies them on the target.

The TRANSDISCARD and TRANSEXCEPTION logic cause Replicat to roll back to the first record after the last successful commit. This may or may not be the actual beginning of the offending transaction. It depends on whether that transaction was split up and parts of it are in the previously committed transactions. If that is the case, Replicat cannot apply the TRANSDISCARD or TRANSEXCEPTION action to the whole transaction as it was issued on the source, but only to the part that was rolled back from the target.

If you use MAXTRANSOPS, make certain that it is set to a value that is larger than the largest transaction that you expect to be handled by TRANSDISCARD and TRANSEXCEPTION. This will ensure that transactions are not be split apart into smaller ones on the target.

Effect of Transaction-level Options on Statistics

The output of informational commands, such as STATS REPLICAT, will show the total number of records in the transaction that was processed by TRANSDISCARD or TRANSEXCEPTION logic. This number may reflect the following:

- Replicat writes all records of the transaction to the discard file, including any records that were excluded from Oracle GoldenGate processing by means of a FILTER or WHERE clause in a MAP statement.
- If a source table in the transaction has multiple targets, the discarded transaction will contain multiple copies of each record, one for each target.
- Replicat ignores any exceptions mapping statements (as specified with EXCEPTIONSONLY or MAPEXCEPTION in a MAP statement) when discarding the transaction.

Replicat abends on errors that are caused by the discard processing (TRANSDISCARD) or exceptions mapping (TRANSEXCEPTION).

Getting More Information about Error Handling

See Error Management in *Oracle GoldenGate Microservices Documentation* for more information about configuring error handling.

See "TABLE | MAP" for more information about the `MAP` parameter.

Default

`TRANSABORT` for deadlocks; `ABEND` for all others

Syntax

```
REPERROR {
(
{DEFAULT | DEFAULT2 | SQL_error | user_defined_error},
{ABEND |
DISCARD |
EXCEPTION |
IGNORE |
RETRYOP [MAXRETRIES n] |
TRANSABORT [, MAXRETRIES] [, DELAYSECS n | DELAYCSECS n] |
TRANSDISCARD |
TRANSEXCEPTION
}
{PROCEDURE,[ABEND|IGNORE|DISCARD]}) |
RESET }
```

Error Specification Options

`DEFAULT`
Sets a global response to all errors except those for which explicit `REPERROR` statements are specified.

`DEFAULT2`
Provides a backup default action when the response for `DEFAULT` is set to `EXCEPTION`. Use `DEFAULT2` when an exceptions `MAP` statement is not specified for a `MAP` statement for which errors are anticipated.

`SQL_error`
A SQL error number. This can be a record-level error or a commit-level error if using `TRANSDISCARD` and `TRANSEXCEPTION`.

`user_defined_error`
A user-defined error that is specified with the `RAISEERROR` option of a `FILTER` clause within a `MAP` statement.

Error Response Options

`ABEND`
Rolls back the transaction and terminates processing abnormally. `ABEND` is the default.

`DISCARD`
Logs the offending operation to the discard file but continue processing the transaction and subsequent transactions.

EXCEPTION
Handles the operation that causes an error as an exception, but processes error-free operations in the transaction normally. Use this option in conjunction with an exceptions `MAP` statement or to work with the `MAPEXCEPTION` option of `MAP`. For example, you can map columns from failed update statements into a "missing updates" table. In the parameter file, specify the exceptions `MAP` statement after the `MAP` statement for which the error is anticipated. `EXCEPTION` applies exception handling only to an individual SQL operation on an individual record. To apply exception handling to the entire transaction, use the `TRANSEXCEPTION` option.

> **Note:**
>
> When the Conflict Detection and Resolution (CDR) feature is active, CDR automatically treats all operations that cause errors as exceptions if an exceptions `MAP` statement exists for the affected table. In this case, `REPERROR` with `EXCEPTION` is not necessary, but you should use `REPERROR` with other options to handle conflicts that CDR cannot resolve, or for conflicts that you do not want CDR to handle.

IGNORE
Ignores the error.

RETRYOP [MAXRETRIES *n*]
Retries the offending operation. Use the `MAXRETRIES` option to control the number of retries. For example, if a table is out of extents, `RETRYOP` with `MAXRETRIES` gives you time to add extents so the transaction does not fail. Replicat abends after the specified number of `MAXRETRIES`.

TRANSABORT [, MAXRETRIES *n*] [, DELAYSECS *n* | DELAYCSECS *n*]
Aborts the transaction and repositions to the beginning of the transaction. This sequence continues either until the record(s) are processed successfully or `MAXRETRIES` expires. If `MAXRETRIES` is not set, the `TRANSABORT` action will loop continuously.
Use one of the `DELAY` options to delay the retry. `DELAYSECS` *n* sets the delay in seconds and the default is 60 seconds. `DELAYCSECS` *n* sets the delay in centiseconds.
The `TRANSABORT` option is useful for handling timeouts and deadlocks on databases that support those conditions.

TRANSDISCARD
Discards the entire source transaction if any operation within that transaction, including the commit operation, causes a Replicat error that is listed in the `REPERROR` error specification. Replicat aborts the transaction and, if the error occurred on a record, writes that record to the discard file. Replicat then replays the transaction and writes all of the records to the discard file, including the commit record. Replicat abends on errors that are caused by the discard processing.
If the discarded record has already been data-mapped to a target record, Replicat writes it to the discard file in the target format; otherwise, it will be written in source format. The replayed transaction itself is always written in source format.
`TRANSDISCARD` supports record-level errors as well as commit errors.
Additional information is at the beginning of this topic.

TRANSEXCEPTION
If an error specified with `REPERROR` occurs on any record in a transaction, performs exceptions mapping for every record in the transaction according to its corresponding exceptions-mapping specification, as defined by a `MAPEXCEPTION` or `EXCEPTIONSONLY` clause in an exceptions `MAP` statement. If any record does not have a corresponding exceptions mapping

specification, or if there is an error writing to the exceptions table, Replicat abends with an error message.
When an error is encountered and TRANSEXCEPTION is being used, Replicat aborts the transaction and, if the error occurred on a record, writes that record to the discard file. Replicat replays the transaction and examines the source records to find the exceptions-mapping specifications, and then executes them.
TRANSEXCEPTION supports record-level errors as well as commit errors. To handle errors at the record level (for individual SQL operations), without affecting error-free operations in the same transaction, use the EXCEPTION option in a MAP statement.

PROCEDURE,[ABEND|IGNORE|DISCARD]
Use PROCEDURE to configure behavior of Replicat when a procedural replication error occurs. By default, Replicat will ABEND when a procedural replication error occurs.
The IGNORE option ignores the call that failed. The DISCARD option stages the discarded errors in the apply error queue in the target database. These errors can be re-executed or deleted at a later time.

RESET
Use a REPERROR RESET statement to remove error-handling rules specified in previous REPERROR parameters and apply default error handling to all MAP statements that follow.

Examples of Using REPERROR Globally

These examples show REPERROR as used at the root of the parameter file to set global error-handling rules. You can override any or all of these rules for any given table or tables by using REPERROR in a MAP statement. See "Examples of Using REPERROR Globally and in a MAP Statement".

Example 1
The following example demonstrates how to stop processing for most errors, but ignore duplicate-record errors.

```
REPERROR (DEFAULT, ABEND)
REPERROR (-1, IGNORE)
```

Example 2
The following example invokes an exceptions MAP statement created to handle errors on the account table. Errors on the product table cause Replicat to end abnormally because an exceptions MAP statement was not defined.

```
REPERROR (DEFAULT, EXCEPTION)
REPERROR (DEFAULT2, ABEND)
MAP sales.product, TARGET sales.product;
MAP sales.account, TARGET sales.account;
INSERTALLRECORDS
MAP sales.account, TARGET sales.account_exception,
EXCEPTIONSONLY,
COLMAP (account_no = account_no,
optype = @GETENV ('lasterr', 'optype'),
dberr = @GETENV ('lasterr', 'dberrnum'),
dberrmsg = @GETENV ('lasterr', 'dberrmsg'));
```

Example 3
The following applies error rules for the first MAP statement and then restores the default of ABEND to the second one.

```
REPERROR (-1, IGNORE)
MAP sales.product, TARGET sales.product;
REPERROR RESET
MAP sales.account, TARGET sales.account;
```

Example 4
The following discards the offending record and then replays the entire transaction if any operation on a record within it generates an error 1403. Other error types cause Replicat to abend.

```
REPERROR DEFAULT ABEND
REPERROR 1403 TRANSDISCARD
```

Example 5
The following discards the offending record and then replays the entire transaction to search for an exceptions-mapping specification that writes to the exceptions table that is named `tgtexception`. Other errors cause Replicat to discard the offending record (if applicable) and then abend.

```
REPERROR DEFAULT ABEND
REPERROR 1403 TRANSEXCEPTION
MAP src, TARGET tgt, &
MAPEXCEPTION (TARGET tgtexception, INSERTALLRECORDS, COLMAP (…) );
```

Examples of Using REPERROR Globally and in a MAP Statement

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_name
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_name
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_name
REPERROR (error1 , response1)
MAP src1, TARGET tgt1, REPERROR (error1, response2);
MAP src2, TARGET tgt2, REPERROR (error2, response3);
REPERROR (error1 , response4)
MAP src2, TARGET tgt2, REPERROR (error3, response3);
```

In the preceding example, if `error1` occurs for the first `MAP` statement, the action should be `response2`. For the second one it would be `response1` (the global response), and for the third one it would be `response4` (because of the second `REPERROR` statement). A global `REPERROR` statement applies to all `MAP` statements that follow it in the parameter file until another `REPERROR` statement starts new rules.

Example 4

```
REPERROR DEFAULT ABEND
REPERROR 1403 TRANSDISCARD.
MAP src, TARGET tgt, REPERROR(600 TRANSDISCARD);
```

In the preceding example, if error 600 is encountered while applying source table `src` to target table `tgt`, the whole transaction is written to discard file. Encountering error 1403 also results in the same action based on the global `REPERROR` specification. On the other errors, the process simply discards only the offending record and then abends.

REPFETCHEDCOLOPTIONS

Valid For

Replicat

Description

Use the `REPFETCHEDCOLOPTIONS` parameter to determine how Replicat responds to operations for which a fetch from the source database was required. The Extract process fetches column data when the transaction record does not contain enough information to construct a SQL statement, when a `FETCHCOLS` clause is used, or when `FETCHOPTIONS MISSINGCOLS` is used.

See "`{FETCHCOLS | FETCHCOLSEXCEPT} (column_list)`" and MISSINGCOLS for more information. This parameter is used when testing Oracle GoldenGate transformation in the Replicat to help instruct the Replicat what to do when a value is missing from the trail, or to report information about how that value was obtained (if it was not directly obtained from the transaction log).

Default

None

Syntax

```
REPFETCHEDCOLOPTIONS
[, INCONSISTENTROW ALLOW|IGNORE|REPORT|DISCARD|ABEND]
[, LATESTROWVERSION ALLOW|IGNORE|REPORT|DISCARD|ABEND]
[, MISSINGROW ALLOW|IGNORE|REPORT|DISCARD|ABEND]
[, NOFETCH ALLOW|IGNORE|REPORT|DISCARD|ABEND]
[, REDUNDANTROW ALLOW|IGNORE|REPORT|DISCARD|ABEND]
[, SNAPSHOTROW ALLOW|IGNORE|REPORT|DISCARD|ABEND]
[, SETIFMISSING string]
```

`INCONSISTENTROW`
Determines the action to perform when 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). Valid values are

ALLOW
Process the operation unless the record length is zero (0).

IGNORE
Ignore the condition and continue processing.

REPORT
Write the record to the discard file and process the operation.

DISCARD
Discard the data and do not process the row.

ABEND
Discard the data and quit processing.

LATESTROWVERSION ***action***
Provides a response when column data was fetched from the current row in the table. Valid values are:

ALLOW
Process the operation unless the record length is zero (0).

IGNORE
Ignore the condition and continue processing.

REPORT
Write the record to the discard file and process the operation.

DISCARD
Discard the data and do not process the row.

ABEND
Discard the data and quit processing.

NOFETCH ***action***
Prevents fetching. One use for this option is when the database is a standby and Oracle GoldenGate does not have a database connection. In this case, an attempt to fetch from the database would result an error. Other scenarios may warrant the use of this parameter as well.
When Oracle GoldenGate cannot fetch data it normally would fetch, it probably will cause data integrity issues on the target.
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 zero (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:

ALLOW
Process the operation unless the record length is zero (0).

IGNORE
Ignore the condition and continue processing.

REPORT
Write the record to the discard file and process the operation.

DISCARD
Discard the data and do not process the partial row.

ABEND
Discard the data and quit processing.

REDUNDANTROW
Indicates that column data was not fetched because column data was previously fetched for this record.

SETIFMISSING [*string*]
Provides a value when a fetch was unsuccessful (and the value is missing from the trail record) but the target column has a not-null constraint. It takes an optional ASCII string as a value for `CHAR` and `BINARY` data types or defaults to the following.
`CHAR, VARCHAR`: Single space
`BINARY, VARBINARY`: A `NULL` byte
`TIMESTAMP`: Current date/time
`FLOAT, INTEGER`: Zero
Besides `SETIFMISSING`, you can use the `COLMAP` clause of the `MAP` statement to map a value for the target column. See "`COLMAP (column_mapping)`" for more information.

SNAPSHOTROW
Indicates that column data was fetched from a snapshot. Generally, this option would only be used for reporting or discarding operations. Valid values are:

ALLOW
Process the operation unless the record length is zero (0).

IGNORE
Ignore the condition and continue processing.

REPORT
Write the record to the discard file and process the operation.

DISCARD
Discard the data and do not process the row.

ABEND
Discard the data and quit processing.

REPLACEBADCHAR

Valid For

Extract and Replicat

Description

Use the `REPLACEBADCHAR` parameter to control the response of the process when a valid code point does not exist for either the source or target character set when mapping character-type columns. By default, the check for invalid code points is only performed when the source and target databases have different character sets, and the default response is to abend. You can use the `FORCECHECK` option to force the process to check for invalid code points when the source and target databases have the same character set. `REPLACEBADCHAR` applies globally.

Default

```
ABORT
```

Syntax

```
REPLACEBADCHAR {ABORT | SKIP | ESCAPE | SUBSTITUTE string | NULL | SPACE} [FORCECHECK]
[NOWARNING]
```

ABORT
The process abends on an invalid code point. This is the default.

SKIP
The process skips the record that has the invalid code point. Use this option with caution, because skipping a record can cause data discrepancies on the target.

ESCAPE
The process replaces the data value with an escaped version of the data value. Depending on the character set of the source database, the value is output as one of the following:

- If the source data is not `UTF-16` (`NCHAR`/`NVARCHAR`), the output is hexadecimal (`\xXX`).
- If the source data is `UTF-16`, the output is Unicode (`\uXXXX`).

SUBSTITUTE *string*
The process replaces the data with a specified string, either Unicode notation or up to four characters. By default the default substitution character of the target character set is used for replacement.

NULL
The process replaces an invalid character with the value of `NULL` if the target column is nullable or, otherwise, assigns a white space (U+0020).

SPACE
The process replaces an invalid character with a white space (U+0020).

FORCECHECK
The process checks for invalid code points when the source and target databases have identical character sets. This overrides the default, where the validation is skipped when the source and target character sets are identical.

NOWARNING
The process suppresses warning messages related to conversion and validation errors.

Examples

Example 1
The following example replaces invalid code points with the value of `NULL`.

```
REPLACEBADCHAR NULL
```

Example 2
Because `ESCAPE` is specified, Oracle GoldenGate will replace the Euro symbol in a source `NCHAR` column with the escaped version of `u20AC`, because the target is ISO-8859-1, which does not support the Euro code point.

```
REPLACEBADCHAR ESCAPE
```

Example 3
The following substitutes a control character for invalid characters.

```
REPLACEBADCHAR SUBSTITUTE \u001A
```

REPLACEBADNUM

Valid For

Replicat

Description

Use the `REPLACEBADNUM` parameter to specify a substitution value for invalid numeric data encountered when mapping number columns. `REPLACEBADNUM` applies globally.

Default

Replace invalid numbers with `NULL`.

Syntax

```
REPLACEBADNUM {number | NULL | UNPRINTABLE}
```

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.

Examples

Example 1

```
REPLACEBADNUM 1
```

Example 2

```
REPLACEBADNUM NULL
```

REPLICAT

Valid For

Replicat

Description

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. See "SPECIALRUN" for more information.

Default

None

Syntax

```
REPLICAT group_name
```

group_name
The group name as defined with the `ADD REPLICAT` command. To view the names of existing Replicat groups, use the `INFO REPLICAT *` command.

Example

```
REPLICAT finance
```

REPORT

Valid For

Extract and Replicat

Description

Use the `REPORT` parameter to specify the interval at which 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 "STATOPTIONS" for more information.

For more information about the process reports, see Using the Process Report in *Oracle GoldenGate Microservices Documentation*.

Default

Generate runtime statistics at the end of each run.

Syntax

```
REPORT
{AT hh:mi |
ON day |
AT hh:mi ON day}
```

`AT hh:mi`
Generates the report at a specific time of the day. Using `AT` without `ON` generates a report at the specified time every day.

`ON day`
Generates the report on a specific day of the week. Valid values are:

```
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
```

The values are not case-sensitive.

Examples

Example 1

```
REPORT AT 17:00
```

Example 2

```
REPORT ON SUNDAY AT 1:00
```

REPORTCOUNT

Valid For

Extract and Replicat

Description

Use the `REPORTCOUNT` parameter to report a count of transaction records that Extract or Replicat processed since startup. Each transaction record represents a logical database operation that was performed within a transaction that was captured by Oracle GoldenGate. The record count is printed to the report file and to the screen.

> **Note:**
>
> This count might differ from the number of records that are contained in the Oracle GoldenGate trail. If an operation affects data that is larger than 4K, it must be stored in more than one trail record. Hence, a report count might show 1,000 records (the database operations) but a trail count might show many more records than that. To obtain a count of the records in a trail, use the Logdump utility.

You can schedule record counts at regular intervals or after a specific number of records. Record counts are carried over from one report to the other.

`REPORTCOUNT` can be used only once in a parameter file. If there are multiple instances of `REPORTCOUNT`, Oracle GoldenGate uses the last one.

Default

None

Syntax

```
REPORTCOUNT [EVERY] count
{RECORD | RECORDS | SECOND | SECONDS | MINUTE | MINUTES | HOUR |HOURS} [, RATE]
```

count
The interval after which to output a count.

RECORD | RECORDS | SECOND | SECONDS | MINUTE | MINUTES | HOUR |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. 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.

Examples

Example 1
This example generates a record count every 5,000 records.

```
REPORTCOUNT EVERY 5000 RECORDS
```

Example 2
This example generates a record count every ten minutes and also reports processing statistics.

```
REPORTCOUNT EVERY 10 MINUTES, RATE
```

The processing statistics are similar to this:

```
12000 records processed as of 2011-01-01 12:27:40 (rate 203,delta 308)
```

REPORTROLLOVER

Valid For

Extract and Replicat

Description

Use the REPORTROLLOVER parameter to force report files to age on a regular schedule, in addition of when a process starts. For long or continuous runs, setting an aging schedule aids in controlling 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}
```

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
```

The values are not case-sensitive.

Examples

Example 1

```
REPORTROLLOVER AT 05:30
```

Example 2

```
REPORTROLLOVER ON friday
```

Example 3

```
REPORTROLLOVER AT 05:30 ON friday
```

RESTARTCOLLISIONS | NORESTARTCOLLISIONS

Valid For

Replicat

Description

Use the `RESTARTCOLLISIONS` and `NORESTARTCOLLISIONS` parameters to control whether or not Replicat applies `HANDLECOLLISIONS` logic after Oracle GoldenGate has stopped because of a conflict. By default, `NORESTARTCOLLISIONS` applies. However, there might be circumstances when you would want Oracle GoldenGate to apply `HANDLECOLLISIONS` logic for the first transaction after startup. For example, if the server is forcibly shut down, the database might have committed the last Replicat transaction, but Oracle GoldenGate might not have received the acknowledgement. Consequently, Replicat will retry the transaction upon startup. `HANDLECOLLISIONS` automatically handles the resultant errors that occur.

`RESTARTCOLLISIONS` enables `HANDLECOLLISIONS` functionality until the first Replicat checkpoint (transaction) is complete. You need not specify the `HANDLECOLLISIONS` parameter in the parameter file. After the first checkpoint, `HANDLECOLLISIONS` is automatically turned off.

See "HANDLECOLLISIONS | NOHANDLECOLLISIONS" for more information about handling collisions.

Default

```
NORESTARTCOLLISIONS
```

Syntax

```
RESTARTCOLLISIONS | NORESTARTCOLLISIONS
```

RMTFILE

Valid For

Extract

Description

Use the RMTFILE parameter to define the name of an extract file on a remote system that will be created by an initial-load Extract and read by an initial-load Replicat when SPECIALRUN is used. Use this parameter for initial load configurations. For online change synchronization, use the RMTTRAIL parameter.

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

RMTFILE is deprecated and ignored for Extract Pump.

You can encrypt the data in this file by using the ENCRYPTTRAIL parameter. See "ENCRYPTTRAIL | NOENCRYPTTRAIL" for more information.

Default

None

Syntax

```
RMTFILE file_name
[, APPEND]
[, PURGE]
[, MEGABYTES megabytes]
[, FORMAT RELEASE major.minor]
[, OBJECTDEFS | NO_OBJECTDEFS]
[, TRAILBYTEORDER {BIGENDIAN | LITTLEENDIAN | NATIVEENDIAN}]
```

file_name
The relative or fully qualified name of the remote trail. Use only two characters for the trail name. As trail files are aged, a six-character sequence number is added to this name, for example /ogg/dirdat/rf000001. If using FORMAT RELEASE 11.2 or earlier, the trail file created is a static file that does not increment, and the naming convention is not limited to two characters..

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.

MEGABYTES *megabytes*
Specifies the maximum size, in megabytes, of a file in the trail. The default size is 2000 MB.

FORMAT RELEASE *major.minor*
Specifies the metadata format of the data that is sent by Extract to a trail, a file, or (if a remote task) to another process. The metadata tells the reader process whether the data records are of a version that it supports. The metadata format depends on the version of the Oracle GoldenGate process. Older Oracle GoldenGate versions contain different metadata than newer ones.
FORMAT RELEASE specifies an Oracle GoldenGate release version. *major* is the major version number, and *minor* is the minor version number. The X.x must reflect a current or earlier, generally available (GA) release of Oracle GoldenGate. Valid values are 11.1 through the current Oracle GoldenGate X.x version number, for example 11.2 or 12.1. The release version is programmatically mapped back to the appropriate trail format compatibility level. The default is the current version of the process that writes to this trail.

> **Note:**
>
> RELEASE versions earlier than 12.1 do not support three-part object names.

> **Note:**
>
> If using multiple trails in a single Extract, only RELEASE versions that are the same can coexist.

The following settings are supported for Oracle Database 12.2 and higher:

- For Oracle Database 12.2 non-CDB or higher with compatibility set to 12.1, FORMAT RELEASE 12.2 or above is supported.
- For Oracle Database 12.2 non-CDB or higher with compatibility set to 12.2, FORMAT RELEASE 12.2 or above is supported.
- For Oracle Database 12.2 CDB/PDB or higher with compatibility set to 12.2, only FORMAT RELEASE values 12.3 or higher are supported. This is due to the use of local undo for PDBs, which requires augmenting the transaction ID with the PDB number to ensure uniqueness of trx IDs.

OBJECTDEFS | NO_OBJECTDEFS
Use the OBJECTDEFS and NO_OBJECTDEFS options to control whether or not to include the object definitions in the trail. These two options are applicable only when the output trail is formatted in Oracle GoldenGate canonical format and the trail format release is greater than 12.1. Otherwise, both options are ignored because no metadata record will be added to the trail.

TRAILBYTEORDER {BIGENDIAN | LITTLEENDIAN | NATIVEENDIAN}
Sets the byte format of the metadata in the file records. This parameter does not affect the column data. Valid only for files that have a FORMAT RELEASE version of at least 12.1. Valid values are BIGENDIAN (big endian), LITTLEENDIAN (little endian), and NATIVEENDIAN (default of the local system). The default is BIGENDIAN. See the GLOBALS version of TRAILBYTEORDER for additional usage instructions.

Examples

Example 1

```
RMTFILE /ggs/dirdat/rf, MEGABYTES 200
```

Example 2

```
RMTFILE /ggs/dirdat/salesny, MEGABYTES 500, FORMAT RELEASE 12.3
```

ROLLOVER

Valid For

Extract

Description

Use the ROLLOVER parameter to specify the interval at which 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 that represent distinct periods of time (for example, each day). It facilitates continuous processing while providing a means for organizing the output. It also provides a means for organizing batch runs by deactivating one file and starting another for the next run.

Files roll over between transactions, not in the middle of one, ensuring data integrity. Checkpoints are recorded when files roll over to ensure that previous files are no longer required for processing.

Rollover occurs only if the rollover conditions are satisfied during the run. For example, if ROLLOVER ON TUESDAY is specified, and data extraction starts on Tuesday, the rollover does not occur until the next Tuesday (unless more precise ROLLOVER rules are specified). You can specify up to 30 rollover rules.

Rollover is required when the DB_UNIQUE_NAME parameter is changed at the source. The reason is that changing that parameter requires the instance itself to be stopped and restarted. After restarting, Extract starts the new trail file in the sequence according to the existing rollover logic. Other processes such as Distribution Path, and Replicat don't abend or require a restart after DB_UNIQUE_NAME parameter is changed.

If the value of the DbUniqueName token on the source changes, then Data Pump and Distribution Service cause a rollover to the next trail file in order to maintain a correct value for the unique name in the trail header.

Either the AT or ON option is required. Both options can be used together, and in any order. Using AT without ON creates a new trail file at the specified time every day.

A trail sequence number can be incremented from 000001 through 999999, and then the sequence numbering starts over at 000000.

Default

Roll over when the default file size is reached or the size specified with the MEGABYTES option of the ADD RMTTRAIL or ADD EXTTRAIL command is reached.

Syntax

```
ROLLOVER {AT hh:mi | ON day | AT hh:mi ON day} [REPORT]
```

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
```

The values 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.

Examples

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
```

SCHEMAEXCLUDE

Valid For

Extract, Replicat, DEFGEN

Description

Use the SCHEMAEXCLUDE parameter to exclude source objects that are owned by the specified source owner (such as a schema) from the Oracle GoldenGate configuration when wildcards are being used to specify the owners in TABLE or MAP statements. This parameter is valid for two- and three-part names.

Wildcards can be used for the optional *catalog* or *container* specification, as well as the *schema* specification. Make certain not to use wildcards such that all objects are excluded.

The positioning of SCHEMAEXCLUDE in relation to parameters that specify files or trails determines its effect. Parameters that specify trails or files are: EXTFILE, RMTFILE, EXTTRAIL, RMTTRAIL. The parameter works as follows:

- When a SCHEMAEXCLUDE specification is placed before any TABLE or SEQUENCE parameters, and also before the parameters that specify trails or files, it applies globally to all trails or files, and to all TABLE and SEQUENCE parameters.
- When a SCHEMAEXCLUDE specification is placed after a parameter that specifies a trail or file, it is effective only for that trail or file and only for the TABLE or SEQUENCE parameters that are associated with it. Multiple trail or file specifications can be made in a parameter file, each followed by a set of TABLE, SEQUENCE, and TABLEEXCLUDE specifications.

SCHEMAEXCLUDE is evaluated before evaluating the associated TABLE or SEQUENCE parameters. Thus, the order in which they appear does not make a difference.

See also the EXCLUDEWILDCARDOBJECTSONLY parameter.

Default

None

Syntax

```
SCHEMAEXCLUDE [container.]schema
```

***container*.**
If the database requires three-part names, specifies the source Oracle container that contains the source owner that is to be excluded. Use if a qualifier is required to identify the correct owner to exclude.

schema
Specifies the name of the source owner that is to be excluded. For databases that require three-part names, you can use *schema* without *container* if the SCHEMAEXCLUDE specification precedes a set of TABLE or MAP parameters for which the default container is specified with the SOURCECATALOG parameter.

Examples

Example 1
This Oracle example requires both *container* and *schema* specifications and demonstrates how wildcards can be used as part of the specification.

```
EXTRACT capt
USERIDALIAS alias1
RMTHOST sysb, MGRPORT 7809
RMTTRAIL /ggs/dirdat/aa
SCHEMAEXCLUDE pdbtest.test*
TABLE pdb*.*.*;
```

Example 2
This example shows how to use SCHEMAEXCLUDE when the database requires only a two-part name.

```
TABLE abc*.*;
SCHEMAEXCLUDE abctest*
```

SEQUENCE

Valid For

Extract

Description

Use the SEQUENCE parameter to capture sequence values from the transaction log. Currently, Oracle GoldenGate supports sequences for the Oracle database.

> **Note:**
>
> Using sequences requires additional configuration steps within the Oracle Database. See Support for Oracle Sequences and Using Sequences topics in the *Oracle GoldenGate Microservices Documentation*.

Default

None

Syntax

```
SEQUENCE [container.]schema.sequence;
```

`[container.]schema.sequence`
Specifies the fully qualified name of the source sequence. Include the name of the pluggable database if the source is an Oracle container database.

`;`
Terminates the `SEQUENCE` parameter statement.

Example

```
SEQUENCE hr.employees_seq;
```

SESSIONCHARSET

Valid For

GLOBALS, valid for MySQL

Description

Use the `SESSIONCHARSET` parameter to set the database session character set for all database connections that are initiated by Oracle GoldenGate processes in the local Oracle GoldenGate instance. Processes that log into the database include, DEFGEN, Extract, and Replicat.

This parameter supports MySQL. The database character set for other databases is obtained programmatically.

The `SESSIONCHARSET` option of the `DBLOGIN` command can be used to override this setting for any commands issued in the same session. The `SESSIONCHARSET` option of the `SOURCEDB` and `TARGETDB` parameters can be used to override this setting for individual process logins.

Default

Character set of the operating system

Syntax

```
SESSIONCHARSET character_set
```

`character_set`
The database session character set.

Example

```
SESSIONCHARSET ISO-8859-11
```

SETENV

Valid For

Extract and Replicat

Any SETENV values should be set before any USERID, USERIDALIAS, FETCHUSERID, FETCHUSERIDALIAS, TRANLOGOPTIONS MININGUSER, MININGUSERALIAS, SOURCEDB or, TARGETDB entries in the parameter files.

Description

Use the SETENV parameter to set a value for an environment variable. When Extract or Replicat starts, it uses the specified value instead of the one that is set in the operating system environment. A variable set in the SETENV statement overrides any existing variables set at the operating system level. Use one SETENV statement per variable to be set.

For integrated extracts, you can set new environment variables if they are available from the lcr server. The new environment variables are:

- USERNAME: Database login user name
- OSUSERNAME: Operating System user name
- MACHINENAME: Name of the host, machine, or server where the database is running
- PROGRAMNAME: Name of program or application that started the transaction or session
- CLIENTIDENTIFIER: Value set using DBMS_SESSION.set_identifier()

SETENV cannot be used with query parameters.

Default

None

Syntax

```
SETENV (
{environment_variable |
  GGS_CacheRetryCount |
  GGS_CacheRetryDelay}
= 'value'
)
```

environment_variable
The name of the environment variable to be set.

'value'
A value for the specified variable. Enclose the value within single quotes.

GGS_CacheRetryCount
(SQL Server) Oracle GoldenGate environment parameter that controls the number of times that Extract tries to read the source transaction log files when they are blocked because of

excessive system activity. The default is 10 retries. After trying the specified number of times, Extract abends with an error that begins as follows:

```
GGS ERROR 600 [CFileInfo::Read] Timeout expired after 10 retries with 1000 ms delay
waiting to read transaction log or backup files.
```

If you continue to see timeout messages in the report file or error log, increase this parameter to allow more retries.

GGS_CacheRetryDelay
(SQL Server) Oracle GoldenGate environment parameter that controls the number of milliseconds that Extract waits before trying again to read the transaction logs when the previous attempt has failed. The default is 1000 milliseconds delay.

Examples

Example 1
Using separate SETENV statements allows a single instance of Oracle GoldenGate to connect to multiple Oracle database instances without having to change environment settings. The following parameter statements set a value for ORACLE_HOME and ORACLE_SID.

```
SETENV (ORACLE_HOME = '/home/oracle/ora9/product')
SETENV (ORACLE_SID = 'ora9')
```

Example 2
The following parameter statements set values for Oracle GoldenGate in a SQL Server environment where Extract tries to read the transaction log for a maximum of 20 times before abending, with a delay of 3000 milliseconds between tries.

```
SETENV (GGS_CacheRetryCount = 20)
SETENV (GGS_CacheRetryDelay = 3000)
```

SOURCECATALOG

Valid For

Extract and Replicat

Description

Use the SOURCECATALOG parameter to specify one of the following for subsequent TABLE or MAP statements that contain two-part names, where three-part object names are required to fully identify a default source Oracle pluggable database (PDB)

This parameter provides an efficient alternative to specifying the full three-part object name (container.schema.object or catalog.schema.object) when specifying source objects from an Oracle consolidated database. Only the two-part name (schema.object) need be specified in subsequent TABLE or MAP statements when SOURCECATALOG is used. You can use multiple instances of SOURCECATALOG to specify different default containers or catalogs for different sets of TABLE statements (or SEQUENCE statements, if Oracle).

Three-part name specifications encountered after SOURCECATALOG override the SOURCECATALOG specification in a TABLE statement, MAP statement, or other parameter that takes object names as input.

Default

None

Syntax

```
SOURCECATALOG {container}
```

container
The name of an Oracle pluggable database that contains the specified objects in the `TABLE` of `MAP` statement.

Example

In the following example, `SOURCECATALOG` is used to specify three different source Oracle PDBs in an Extract parameter file.

```
SOURCECATALOG FINANCE
TABLE SAP.*;
TABLE REPORTS.*;
SOURCECATALOG HR
TABLE SIEBEL.*;
TABLE REPORTS.*;
SOURCECATALOG MFG
TABLE CUSTOMER.ORDERS;
TABLE REPORTS.*;
TABLE HQ.LOCATIONS.*;
```

In this example, Extract captures the following:

- All tables in the `SAP` and `REPORTS` schemas in the `FINANCE` PDB.
- All tables in the `SIEBEL` and `REPORTS` schemas in the `HR` PDB.
- All tables in the `CUSTOMER` and `REPORTS` schemas in the `MFG` PDB.
- For the last `TABLE` statement, Extract captures all tables in the `LOCATIONS` schema in the `HQ` PDB. This statement is a fully qualified three-part name and overrides the previous `SOURCECATALOG` specification.

SOURCECHARSET

Valid For

Replicat

Description

Use the `SOURCECHARSET` parameter to control the conversion of data from the source character set to the target character set by Replicat. Replicat converts character sets by default for versions 11.2.1 and later, but you may need to intervene in the following cases:

- To enable accurate conversion of data written by an Extract version earlier than 11.2.1. Extract versions prior to version 11.2.1 do not write information about the source character set to the trail, so the information must be supplied to Replicat directly. Extract versions 11.2.1 and later write information about the source character set to the trail for use by Replicat, and any `SOURCECHARSET` specification is ignored.
- To override the source database character set in the trail file. Use `SOURCECHARSET` with the `OVERRIDE` option to specify the character set you want to use. An example use case is migrating a database to UNICODE or particular character set database from garbage in, garbage out type of non-character set aware database by ignoring the source database character set.

Replicat issues a warning message when it uses the SOURCECHARSET character set.

Use the REPLACEBADCHAR parameter to handle validation errors where there are invalid characters in the source data or the target character set does not support a source character. It provides options to abend on these errors, skip the record that caused the error, or specify a substitute value for the character.

Default

None

Syntax

```
SOURCECHARSET {source_charset | PASSTHRU | OVERRIDE} [DB2ZOS]
```

source_charset
Specifies the source character set for data that is written by an Extract version that is earlier than 11.2.1. Replicat uses the specified character set when converting character-type columns to the target character set.
For *source_charset*, specify the appropriate character-set identifier that represents the source database.
For Oracle, if SOURCECHARSET is not specified but there is an NLS_LANG environment variable on the target, Replicat uses the NLS_LANG value as the source database character set. If neither SOURCECHARSET nor NLS_LANG is present, Replicat abends to prevent possible data corruption.

PASSTHRU
PASSTHRU
Forces Replicat to apply the data without converting the character set. Character set differences are ignored as follows:

- If the database is Oracle, the data is applied the way it is stored in the trail.
- If the database is other than Oracle, the data is applied as binary data if the database supports a bind as binary data. Otherwise, the data is applied as-is.

PASSTHRU is not compatible with the BULKLOAD parameter (direct-bulk load).
If PASSTHRU is specified and a mapping between CHAR/VARCHAR/CLOB and NCHAR/NVARCHAR/NCLOB exists in the MAP statement, Replicat abends.
If any Oracle GoldenGate column-mapping functions are used for character-based columns when PASSTHRU mode is specified, Replicat issues a warning message and converts the results of those functions to the target database character set before mapping them to the target column.
PASSTHRU should only be used if you are certain the source and target character sets are compatible. If you are not sure whether PASSTHRU is appropriate in your environment, contact Oracle Support before using it.

OVERRIDE
Forces Replicat to use the specified character set thus overriding the source database character set in the trail file. This option overrides character type column character set except in the following cases:

- The character set is overridden by the CHARSET and COLCHARSET parameters.
- Use of NCHAR, NVARCHAR and NCLOB data types.
- The database overrides the column character set explicitly to a set other than the database character set.

DB2 for z/OS
Valid for DB2 for z/OS.
Required if the version of a trail that contains DB2 data from the z/OS platform is Oracle GoldenGate 12.1 or lower. This parameter ensures that Replicat recognizes that the data is from DB2 for z/OS, which permits a mix of ASCII and EBCDIC character formats.

Examples

Example 1

```
SOURCECHARSET ISO-8859-9
```

Example 2

```
SOURCECHARSET PASSTHRU
```

Example 3

```
SOURCECAHRSET JA16EUC
```

Example 4

```
SOURCECHARSET OVERRIDE WE8ISO8859P15
```

SOURCEDEFS

Valid For

Replicat

Description

Use the `SOURCEDEFS` parameter to specify the name of a file that contains definitions of source tables or files. Source definitions are not required, by default, when trail files with format Oracle GoldenGate release 12.2.*x* are used because the trail files contains metadata records with the object definitions. However, source definitions are required when replicating data between heterogenous source and targets using trail files with format Oracle GoldenGate release 12.1.*x* and lower or when trail files with created with the `no_objectdefs` option.

Use `SOURCEDEFS` for a Replicat process on the target system, as per your Oracle GoldenGate configuration:

To generate the source-definitions file, use the `DEFGEN` utility. Transfer the file to the intermediary or target system before starting a 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.

Do not use `SOURCEDEFS` and `ASSUMETARGETDEFS` in the same parameter file.

Default

None

Syntax

`SOURCEDEFS` *`file_name`* `[OVERRIDE]`

file_name
The relative or fully qualified name of the file containing the source data definitions.

OVERRIDE
By default, the table definitions from the metadata records override the definitions from any SOURCEDEFS file.
Specify OVERRIDE to request Replicat to use the definitions from the definitions file instead of the metadata.

Examples

Example 1

```
SOURCEDEFS dirdef\tcust.def
```

Example 2

```
SOURCEDEFS /ggs/dirdef/source_defs
```

SOURCEISTABLE

Valid For

Extract

Description

Use the SOURCEISTABLE parameter to extract complete records directly from source tables in preparation for loading them into another table or file. SOURCEISTABLE extracts all column data specified within a TABLE statement.

This parameter applies to the following initial load methods:

- Loading data from file to Replicat.
- Loading data from file to database utility.

Do not use this parameter for the following initial load methods:

- An Oracle GoldenGate direct load, where Extract sends load data directly to the Replicat process without use of a file.
- An Oracle GoldenGate direct bulk load to SQL*Loader.

For those processes, SOURCEISTABLE is specified as an ADD EXTRACT argument instead of being used in the parameter file.

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. See "DDL" for more information.

Default

None

Syntax

```
SOURCEISTABLE
```

SOURCETIMEZONE

Valid For

Replicat

Description

Use the SOURCETIMEZONE parameter to specify the time zone of the source database. Use this parameter for one of the following purposes:

- To override the source time zone that is stored in the trail. By default, Replicat sets its session to the specified time zone, in both region ID and offset value. This option applies to Oracle GoldenGate versions 12.1.2 or later, where the source time zone is written to the trail by Extract. Replicat will set its session to the specified time zone.
- To supply the time zone of the source database when the trail is written by an Extract version that is older than 12.1.2. In these versions, Extract does not write the source time zone to the trail, so it must be supplied by this parameter. Replicat will set its session to the specified time zone.

To disable the default use of the source time zone by Replicat, use the PRESERVETARGETTIMEZONE parameter in the Replicat parameter file. See PRESERVETARGETTIMEZONE for more information.

Default

None

Syntax

```
SOURCETIMEZONE time_zone
```

time_zone
The time zone of the source database as output by the database for DATE, TIME and TIMESTAMP data types. It can be specified in the following ways.

- As a region ID that is valid in the IANA Time Zone Database (tz database). (A region ID is also known as an Olson time zone ID). An adjustment for Daylight Saving Time can be performed by the target database, if supported.
- As an offset from UTC.

Examples

The following examples show different ways to specify SOURCETIMEZONE.

- These examples specify a region ID.

```
SOURCETIMEZONE America/New_York

SOURCETIMEZONE US/Pacific

SOURCETIMEZONE Japan

SOURCETIMEZONE UTC

SOURCETIMEZONE Pacific/Guam
```

- These examples specify an offset from UTC.

```
SOURCETIMEZONE +09:00

SOURCETIMEZONE -04:30
```

SPACESTONULL | NOSPACESTONULL

Valid For

Replicat on Oracle Database only

Description

Use the SPACESTONULL and NOSPACESTONULL parameters to control whether or not a source column that contains only spaces is converted to NULL in the target column. SPACESTONULL converts spaces to NULL if the target column accepts NULL values. NOSPACESTONULL converts spaces to a single space character in the target column.

This parameter is applicable to the follow two scenarios:

- a source column that contains only spaces
- a source column is empty, such as empty CHAR/VARCHAR column data from DB2

Oracle does not distinguish empty and NULL column though other databases do so you should consult your database documentation to determine how these types of columns.

The parameters are table specific. One parameter applies to all subsequent MAP statements, until the other parameter is encountered.

Because you can selectively enable or disable these parameters between MAP statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the SPACESTONULL threads in one set of MAP statements, and specify the NOSPACESTONULL threads in a different set of MAP statements.

Default

```
NOSPACESTONULL
```

Syntax

```
SPACESTONULL | NOSPACESTONULL
```

Example

This example shows how you can apply SPACESTONULL and NOSPACESTONULL selectively to different MAP statements, each of which represents a different thread of a coordinated Replicat.

```
SPACESTONULL
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
NOSPACESTONULL
MAP sales.loc, TARGET sales.loc, THREAD (3);
```

SPECIALRUN

Valid For

Replicat

Description

Use the SPECIALRUN parameter in a Replicat parameter file for a one-time processing run to direct Replicat not to create checkpoints. A one-time run has a beginning and an end, so checkpoints are not needed. Use SPECIALRUN for certain initial data load methods.

When Replicat is in SPECIALRUN mode, do not start it with the START REPLICAT command. It is started automatically during the initial load.

SPECIALRUN requires the use of the END parameter. Either REPLICAT or SPECIALRUN is required in the Replicat parameter file. See "REPLICAT" for more information.

Default

None

Syntax

```
SPECIALRUN
```

SPLIT_TRANS_RECS

Valid For

Parallel Replicat

Description

Specifies that large transactions should be broken into pieces of specified size and applied in parallel. Dependencies between pieces are still honored. Disabled by default.

Syntax

```
SPLIT_TRANS_RECS value
```

Example

```
SPLIT_TRANS_RECS 100000
```

SQLDUPERR

Valid For

Replicat

Description

Use the SQLDUPERR parameter to specify the numeric error code returned by the target database when a duplicate row is encountered. A duplicate-record error indicates that an INSERT operation was attempted with a primary key that matches the key of an existing record in the database.

You must use SQLDUPERR when you specify the special handling of duplicate records with the OVERRIDEDUPS parameter. See "OVERRIDEDUPS | NOOVERRIDEDUPS" for more information.

Default

None

Syntax

```
SQLDUPERR error_number
```

error_number
The numeric error code to return for duplicate records.

Example

```
SQLDUPERR -2601
```

SQLEXEC

Valid For

Extract and Replicat

Description

Use the SQLEXEC parameter to execute a stored procedure, query, or database command within the context of Oracle GoldenGate processing. SQLEXEC enables Oracle GoldenGate to communicate directly with the database to perform any work that is supported by the database. This work can be part of the synchronization process, such as retrieving values for column conversion, or it can be independent of extracting or replicating data, such as executing a stored procedure that executes an action within the database.

> **Note:**
>
> SQLEXEC provides minimal globalization support. To use SQLEXEC in the capture parameter file of the source capture, make sure that the client character set in the source .prm file is either the same or a superset of the source database character set.

SQLEXEC works as follows:

- As a standalone statement at the root level of a parameter file to execute a SQL stored procedure or query or to execute a database command. As a standalone statement, SQLEXEC executes independently of a TABLE or MAP statement during Oracle GoldenGate processing. When used in a standalone SQLEXEC parameter, a query or procedure cannot include parameters. See "Standalone SQLEXEC".
- As part of a TABLE or MAP parameter to execute a stored procedure or query with or without parameters. When used with parameters, the procedure or query that is executed can accept input parameters from source or target rows and pass output parameters. See "SQLEXEC in a TABLE or MAP Parameter".

> **Caution:**
>
> 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.

> **Note:**
>
> The SQLEXECONBEFOREIMAGE parameter supports SQLEXEC execution on Before Image records.

> **Note:**
>
> When SQLEXECONBEFOREIMAGE is specified with the MAP option, which is default, SQLEXEC is executed twice on PK update and unified update record. One for BEFORE image and another one for AFTER image. To prevent executing SQLEXEC twice, the SQLEXECONBEFOREIMAGE parameter is ignored on PK update or unified update record when EXEC type is TRANSACTION, SOURCEROW or ONCE.

Standalone SQLEXEC

A standalone SQLEXEC parameter is one that is used at the root level of a parameter file and acts independently of a TABLE or MAP parameter. The following are 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 credential for the Oracle GoldenGate user must precede the SQLEXEC clause. For Extract, use the SOURCEDB and USERID or USERIDALIAS parameters as appropriate for the database. For Replicat, use the TARGETDB and USERID or USERIDALIAS parameters, as appropriate.
- The database credential that the Oracle GoldenGate process uses is the one that executes the SQL. This credential 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. See "SQLEXEC in a TABLE or MAP Parameter".
- All objects affected by a standalone SQLEXEC statement must exist before the Oracle GoldenGate processes start. Because of this, DDL support must be disabled for those

objects; otherwise, DDL operations could change the structure of, or delete an object, before the SQLEXEC procedure or query executes on it.

- Object names must be fully qualified in their two-part or three-part name format.
- For DB2 on z/OS, Oracle GoldenGate uses the ODBC SQLExecDirect function to execute a SQL statement dynamically. ODBC prepares the SQL statement every time that it is executed, at a specified interval. To support this function, the connected database server must be configured to prepare SQL dynamically. See the DB2 for z/OS documentation for more information.

Getting More Information about Using Standalone SQLEXEC

See Use SQLEXEC to Execute Commands, Stored Procedures, and Queries for more information about how to use SQLEXEC.

Syntax for Standalone SQLEXEC

```
SQLEXEC
{'call procedure_name()' | 'SQL_query' | 'database_command'}
[EVERY n {SECONDS | MINUTES | HOURS | DAYS}]
[ONEXIT]
[, THREADS (threadID[, threadID][, ...][, thread_range[, thread_range][, ...])]
```

'call *procedure_name ()*'
Specifies the name of a stored procedure to execute. Enclose the statement within single quotes. The call keyword is required. The following is an example of how to execute a procedure with standalone SQLEXEC:

```
SQLEXEC 'call prc_job_count ()'
```

'SQL_query'
Specifies the name of a query to execute. Enclose the query within single quotes. Specify case-sensitive object names in the same format required by the database. The following is an example of how to execute a query with standalone SQLEXEC:

```
SQLEXEC ' select x from dual '
```

For a multi-line query, use the single 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).

'database_command'
Executes a database command. The following is an example of how to execute a database command with standalone SQLEXEC:

```
SQLEXEC 'SET TRIGGERS OFF'
```

EVERY *n* {SECONDS | MINUTES | HOURS | DAYS}
Causes a standalone stored procedure or query to execute at a defined interval, for example:

```
SQLEXEC 'call prc_job_count ()' EVERY 30 SECONDS
```

The interval must be a whole, positive integer.

ONEXIT
Executes the SQL when the Extract or Replicat process stops gracefully, for example:

```
SQLEXEC 'call prc_job_count ()' ONEXIT
```

THREADS (*threadID*[, *threadID*][, ...][, *thread_range*[, *thread_range*][, ...])
Executes SQLEXEC only for the specified thread or threads of a coordinated Replicat.

***threadID*[, *threadID*][, ...]**
Specifies a thread ID or a comma-delimited list of threads in the format of threadID, threadID, threadID.

[, *thread_range*[, *thread_range*][, ...]
Specifies a range of threads in the form of threadIDlow-threadIDhigh or a comma-delimted list of ranges in the format of threadIDlow-threadIDhigh, threadIDlow-threadIDhigh.

A combination of these formats is permitted, such as threadID, threadID, threadIDlow-threadIDhigh.

If no THREADS clause is used, the SQL is executed by all of the threads that were configured for this Replicat group by the ADD REPLICAT command. However, if the SQL satisfies the criteria for a barrier transaction, the entire SQLEXEC statement is processed by thread 0 regardless of the actual thread mapping.

SQLEXEC in a TABLE or MAP Parameter

A SQLEXEC parameter in a TABLE or MAP parameter can be used to execute a stored procedure or query that does or does not accept parameters. The following are SQLEXEC dependencies and restrictions when used in a MAP or TABLE statement:

- The SQL is executed by the database user under which the Oracle GoldenGate process is running. This user must have the privilege to execute stored procedures and call database-supplied procedures.
- A query or procedure must be structured correctly when executing a SQLEXEC statement. If Replicat encounters a problem with the query or procedure, the process abends immediately, despite any error-handling rules that are in place. Refer to the SQL reference guide provided by the database vendor for permissible SQL syntax.
- The COMMIT operation of a Replicat transaction to the target database also commits any DML changes that are made in a SQLEXEC statement within the boundary of the original source transaction. This is not true for Extract, because Extract does not perform SQL transactions. When using SQLEXEC for Extract, you can either enable implicit commits or execute an explicit commit within the SQLEXEC procedure.
- Specify literals in single quotes. Specify case-sensitive object names the same way they are specified in the database.
- Do not use SQLEXEC to change the value of 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 of the TABLE and MAP parameters.
- For DB2 on z/OS, Oracle GoldenGate uses the ODBC SQLExecDirect function to execute a SQL statement dynamically. ODBC prepares the SQL statement every time that it is executed, at a specified interval. To support this function, the connected database server

must be configured to prepare SQL dynamically. See the DB2 for z/OS documentation for more information.

- When using Oracle GoldenGate to replicate DDL, all objects that are affected by a stored procedure or query must exist with the correct structures prior to the execution of the SQL. Consequently, DDL on these objects that affects structure (such as `CREATE` or `ALTER`) must execute before the `SQLEXEC` executes.
- All object names in a `SQLEXEC` statement must be fully qualified with their two-part or three-part names, as appropriate for the database.
- Do not use `SQLEXEC` for tables being processed in pass-through mode by a data-pump Extract group.
- The following data types are supported by `SQLEXEC` for input and output parameters.
  - Numeric data types
  - Date data types
  - Character data types
- When executed by a coordinated Replicat, `SQLEXEC` is executed by the thread or threads that are specified with the `THREAD` or `THREADRANGE` option of the `MAP` statement. However, if the `SQLEXEC` is specified in a `MAP` parameter that contains the `COORDINATED` keyword, it is executed as a barrier transaction automatically by the thread with the lowest ID number, regardless of the actual thread mapping.

Getting More Information About Using SQLEXEC in TABLE and MAP

For more information about `TABLE` and `MAP`, see "TABLE | MAP".

Syntax for SQLEXEC in TABLE or MAP

```
SQLEXEC (
{SPNAME procedure_name[, ID logical_name] |
   ID logical_name, QUERY ' SQL_query '}
{, PARAMS [OPTIONAL | REQUIRED] parameter_name = {source_column |
OGG_function} |
   NOPARAMS}
[, AFTERFILTER | BEFOREFILTER]
[, ALLPARAMS {OPTIONAL | REQUIRED}]
[, ERROR {IGNORE | REPORT | RAISE | FINAL | FATAL}]
[, EXEC {MAP | ONCE | TRANSACTION | SOURCEROW}][, MAXVARCHARLEN bytes]
[, PARAMBUFSIZE bytes]
[, TRACE]
[, ...]
[, BEFORE_col1 = @BEFORE(col1),
)
```

SPNAME *procedure_name*[, ID *logical_name*]
Executes a stored procedure.

SPNAME *procedure_name*
Specifies the name of the procedure to execute.
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.

```
SQLEXEC (SPNAME lookup), PARAMS (param1 = srccol)), &
COLMAP (targcol = lookup.param1);
```

ID *logical_name*
Defines an optional logical name for the procedure. For example, logical names for a procedure named `lookup` might be `lookup1`, `lookup2`, and so forth. Use this option to execute the procedure multiple times within a `MAP` statement. A procedure can execute up to 20 times per `MAP` statement. `ID` is not required when executing a procedure once.
The following example shows the use of the `ID` option to enable multiple executions of a stored procedure that gets values from a lookup table. The values are mapped to target columns.

```
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);
```

ID *logical_name*, QUERY ' *SQL_query* '
Executes a query.

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 can either return results with a `SELECT` statement or execute an `INSERT`, `UPDATE`, or `DELETE` statement. A `SELECT` statement should only return one row. If multiple rows are returned, only the first row is processed. Do not specify an `INTO ...` clause for any `SELECT` statements.The query must be valid, standard query language for the database against which it is being executed. Most queries require placeholders for input parameters. How parameters are specified within the query depends on the database type, as follows:

- 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 = ?'
  ```

The query must be contained on one line, within single quotes. Quotation marks are not required around a parameter name for any database.
The following examples illustrate the use of a `SQLEXEC` query for Oracle and SQL Server queries, respectively.
Oracle example:

```
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);
```

SQL Server example:

```
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);
```

PARAMS [OPTIONAL | REQUIRED] *parameter_name* = {*source_column* | *OGG_function*} | NOPARAMS

Defines whether or not the procedure or query accepts parameters and, if yes, maps the parameters to the input source. Either a `PARAMS` clause or `NOPARAMS` must be used.

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.

***parameter_name* = {*source_column* | *OGG_function*}**

Maps the name of a parameter to a column or function that provides the input. The following data types are supported by `SQLEXEC` for input and output parameters.

- Numeric data types
- Date data types
- Character data types

parameter_name is one of the following:

- For a stored procedure, it is the name of any parameter in the procedure that can accept input.
- For an Oracle query, it is the name of any input parameter in the query *excluding* the leading colon. For example, `:vemplid` would be specified as `vemplid` in the `PARAMS` clause. Oracle permits naming an input parameter any logical name.

  ```
  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));
  ```

- For a non-Oracle query, it is `Pn`, where *n* is the number of the parameter within the statement, starting from 1. For example, in a query with two parameters, the *parameter_name* entries are `p1 and p2`. Consider whether the database requires the `p` to be upper or lower case.

```
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));
```

source_column is the name of a source column that provides the input. By default, if the specified column is not present in the log (because the record only contains the values of columns that were updated) the parameter assumes any default value specified by the procedure or query for the parameter.
OGG_function is the name of an Oracle GoldenGate column-conversion function that executes to provide the input. See "Table and Column Mapping Functions".

To pass output values from the stored procedure or query as input to a FILTER or COLMAP clause, use the following syntax:

```
{procedure_name | logical_name}.parameter
```

Where:

- *procedure_name* is the actual name of a stored procedure, which must match the value given for SPNAME in the SQLEXEC statement. Use this argument only if executing a procedure one time during the course of the Oracle GoldenGate run.
- *logical_name* is the logical name specified with the ID option of SQLEXEC. Use this argument to pass input values from either a query or an instance of a stored procedure when the procedure executes multiple times within a MAP statement.
- *parameter* is the name of a parameter or RETURN_VALUE if extracting returned values. By default, output values are truncated at 255 bytes per parameter. If output parameters must be longer, use the MAXVARCHARLEN option.

> **Note:**
>
> As an alternative to the preceding syntax, you can use the @GETVAL function. See "@GETVAL" for more information.

The following examples apply to a set of Oracle source and target tables and a lookup table. These examples show how parameters for the tables are passed for a single instance of a stored procedure and multiple instances of a stored procedure.
Source table cust:

```
custid                    Number
current_residence_state   Char(2)
birth_state               Char(2)
```

Target table cust_extended:

```
custid                          Number
current_residence_state_long    Varchar(30)
birth_state_long                Varchar(30)
```

Lookup table state_lookup

```
abbreviation     Char(2)
long_name        Varchar(30)
```

The following example shows the use of a stored procedure that executes once to get a value from the lookup table. When processing records from the `cust` table, Oracle GoldenGate executes the `lookup` stored procedure before executing the column map. The `long_name` parameter in the procedure accepts input from the `birth_state` source column.The value is mapped to the target column `birth_state_long` 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);
```

The following example shows the use of the `ID` option to enable 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);
```

`AFTERFILTER | 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.

`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.

The following is an example using `BEFOREFILTER`.

```
SQLEXEC (SPNAME check, NOPARAMS, BEFOREFILTER)
```

`ALLPARAMS [OPTIONAL | REQUIRED]`
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`.

`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.

The following is an example using `OPTIONAL`.

```
SQLEXEC (SPNAME lookup,
PARAMS (long_name = birth_state, short_name = state),
ALLPARAMS OPTIONAL)
```

ERROR {IGNORE | REPORT | RAISE | FINAL | FATAL}

Use `ERROR` to define a response to errors associated with the stored procedure or query. Without explicit error handling, the Oracle GoldenGate process abends on errors. Make certain your procedures return errors to the process and specify the responses with `ERROR`. With Oracle GoldenGate 21c, the functionality allows you to specify different behaviors based on the type SQL error.

The following example demonstrates an abend ER process when SQL error code 1403 or 1405 is detected when executing stored procedure lookup. All other errors are reported and replication continues.

```
SQLEXEC ( SPNAME lookup,
  PARAMS( long_name = birth_state, short_name = state),
  ERROR REPORT, ERROR FATAL (1403, 1405) );
```

IGNORE

Causes Oracle GoldenGate to ignore all errors associated with the stored procedure or query and continue processing. Any resulting parameter extraction results in "column missing" conditions. This is the default.

REPORT

Ensures that all errors associated with the stored procedure or query are reported to the discard file. The report is useful for tracing the cause of the error. It includes both an error description and the value of the parameters passed to and from the procedure or query. Oracle GoldenGate continues processing after reporting the error.

RAISE

Handles errors according to rules set by a `REPERROR` parameter. Oracle GoldenGate continues processing other stored procedures or queries associated with the current `MAP` statement before processing the error.

FINAL

Is similar to `RAISE` except that when an error associated with a procedure or query is encountered, remaining stored procedures and queries are bypassed. Error processing is invoked immediately after the error.

FATAL

Causes Oracle GoldenGate to abend immediately upon encountering an error associated with a procedure or query.

EXEC {MAP | ONCE | TRANSACTION | SOURCEROW}

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.

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. `MAP` is the default.

The following example shows the incorrect use of the default of `MAP`. Because `MAP` is the default, it need not be explicitly listed in the `SQLEXEC` statement. In this example, a source table is mapped in separate `MAP` parameters to two different target tables. In this case, the results are valid only for the first mapping. The results of the procedure `lookup` are expired by the time the second `MAP` parameter executes, and the second `MAP` results in a "column missing" condition. To implement this correctly so that each `MAP` returns valid results, `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);
```

ONCE

Executes the procedure or query once during the course of the Oracle GoldenGate run, upon the first invocation of the associated `MAP` statement. The results remain valid for as long as the process remains running.
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);
```

TRANSACTION

Executes the procedure or query once per source transaction. The results remain valid for all operations of the transaction.
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);
```

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.
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);
```

MAXVARCHARLEN *bytes*

Use `MAXVARCHARLEN` to specify the maximum byte length allocated for the output value of any parameter in a stored procedure or query. Beyond this maximum, the output values are truncated. The default is 255 bytes without an explicit `MAXVARCHARLEN` clause. The valid range of values is from 50 to 32767 bytes.
The following example limits the byte length of output values to 100.

```
MAXVARCHARLEN 100
```

PARAMBUFSIZE *bytes*

Use `PARAMBUFSIZE` to specify the maximum number of bytes allowed for the memory buffer that stores `SQLEXEC` parameter information, including both input and output parameters. The default is 10,000 bytes without an explicit `PARAMBUFSIZE` clause. The valid range of values is from 1000 to 2000000 bytes. Oracle GoldenGate issues a warning whenever the memory allocated for parameters is within 500 bytes of the maximum.
The following example increases the buffer to 15,000 bytes.

```
PARAMBUFSIZE 15000
```

TRACE {ALL | ERROR}
Use TRACE to log SQLEXEC input and output parameters to the report file.
The following is a sample report 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
```

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.

STATOPTIONS

Valid For

Extract and Replicat

Description

Use the STATOPTIONS parameter to specify the information that is 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]
[, THREADS (threadID[, threadID][, ...][, thread_range[, thread_range][, ...])]
```

REPORTDETAIL | NOREPORTDETAIL
Valid for Replicat. Controls the reporting of statistics for operations that were not applied to the target because they were discarded as the result of collision handling.

REPORTDETAIL
Returns statistics for the discarded operations. These operations are reported in the regular `STATS REPLICAT` statistics (inserts, updates, and deletes performed) and as discard statistics if `STATS REPLICAT` is issued with the `DETAIL` option. 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. `REPORTDETAIL` is the default.

NOREPORTDETAIL
Turns off the reporting of statistics for discarded operations.

REPORTFETCH | NOREPORTFETCH
Valid for Extract. Controls the reporting of statistics for the amount of row fetching performed by Extract, such as the fetches that are triggered by a `FETCHCOLS` clause or fetches that must be performed when not enough information is in the transaction record.

REPORTFETCH
Reports statistics for row fetching. The output is as follows:

- `row fetch attempts`: The number of times Extract attempted to fetch a column value from the database when it could not obtain the value from the transaction log.
- `fetch failed`: The number of `row fetch attempts` that failed.
- row fetch by key: (Valid for Oracle) The number of row fetch attempts that were made by using the primary key.

NOREPORTFETCH
Turns off the reporting of fetch statistics. `NOREPORTFETCH` is the default.

RESETREPORTSTATS | NORESETREPORTSTATS
Valid for Extract and Replicat. Controls whether or not statistics generated by the `REPORT` parameter are reset when a new report is created. `RESETREPORTSTATS` resets the statistics from one report to the other. `NORESETREPORTSTATS` continues the statistics from one report to another and is the default, see REPORT. Report rollover is controlled by the `REPORTROLLOVER` parameter, see REPORTROLLOVER.

THREADS (*threadID*[, *threadID*][, ...][, *thread_range*[, *thread_range*][, ...])
Enables the selected `STATOPTIONS` options for the specified threads of a coordinated Replicat.

***threadID*[, *threadID*][, ...]**
Specifies a thread ID or a comma-delimited list of threads in the format of `threadID, threadID, threadID`.

[, *thread_range*[, *thread_range*][, ...]
Specifies a range of threads in the form of `threadIDlow-threadIDhigh` or a comma-delimited list of ranges in the format of `threadIDlow-threadIDhigh, threadIDlow-threadIDhigh`.

A combination of these formats is permitted, such as `threadID, threadID, threadIDlow-threadIDhigh`.

Examples

This example includes fetch details of a coordinated Replicat.

```
STATOPTIONS REPORTFETCH
```

This example resets the statistics from one report to another for thread 0 of a coordinated Replicat .

```
STATOPTIONS RESETREPORTSTATS THREADS 0
```

TABLE | MAP

Valid For

`TABLE` is valid for Extract. You can use `TABLE` with Replicat only with the `EVENTACTIONS` parameter. `MAP` is valid for Extract in certain situations and Replicat. See MAP for Extract for details.

Description

The `TABLE` and `MAP` parameters control the selection, mapping, and manipulation of the objects that are to be affected by an Oracle GoldenGate process. These parameters work as follows:

- Use the `TABLE` parameter in an Extract parameter file to specify one or more objects that are to be captured from the data source by the Extract process. `TABLE` options specify processing work such as filtering and token definitions that must be performed before Extract writes the captured data to the Oracle GoldenGate trail.
- List the `TABLE` parameter after listing the `EXTFILE`, `EXTTRAIL`, `RMTFILE`, or `RMTTRAIL` parameter of the Extract. To write multiple trails within the same Extract, create a separate `TABLE` parameter after each trail specification.
- Use the `MAP` parameter in the Replicat parameter file to map the data from the source objects to the appropriate target objects. `MAP` options specify processing work such as filtering, conversion, and error handling that must be performed before the data is applied to the target objects. Each target object that you want to synchronize with a source object must be associated with that source object by means of a `MAP` parameter. Multiple source-target relationships can be specified by means of a wildcard.

`TABLE` and `MAP` are valid for initial load configurations and for online processes configured to support the replication of transactional changes.

You can process the following objects with `TABLE` and `MAP`:

- Index Organized Tables
- Materialized views
- Tables

To specify a sequence for capture by Extract, use the `SEQUENCE` parameter.

> **Note:**
>
> Oracle GoldenGate supports replication of actual data values of Oracle materialized views.

You can use one or more `TABLE` or `MAP` statements in a parameter file, with or without wildcards, to specify all of the objects that you want to process.

You can exclude objects from a wildcarded `TABLE` or `MAP` statement with the TABLEEXCLUDE and MAPEXCLUDE parameters. Additional exclusion parameters are CATALOGEXCLUDE, SCHEMAEXCLUDE, and EXCLUDEWILDCARDOBJECTSONLY.

Default

None

Syntax for TABLE

For tables, you can use all of the `TABLE` options. For non-table objects, use `TABLE` only to specify an object for capture.

```
TABLE source_table[, TARGET target_table]
[, ATTRCHARSET (charset)]
[, CHARSET character_set]
[, COLCHARSET character_set]
[, COLMAP (column_mapping)]
[, {COLS | COLSEXCEPT} (column_list)]
[, {DEF | TARGETDEF} template]
[, EVENTACTIONS action]
[, EXITPARAM 'parameter']
[, {FETCHCOLS | FETCHCOLSEXCEPT} (column_list)]
[, {FETCHMODCOLS | FETCHMODCOLSEXCEPT} (column_list)]
[, FETCHBEFOREFILTER]
[, FILTER (filter_clause)]
[, GETBEFORECOLS (column_specification)]
[, KEYCOLS (columns)]
[, SQLEXEC (SQL_specification)]
[, SQLPREDICATE 'WHERE where_clause']
[, TOKENS (token_definition)]
[, TRIMSPACES | NOTRIMSPACES]
[, TRIMVARSPACES | NOTRIMVARSPACES]
[, WHERE (clause)]
[, container.]schema.table PARTITIONOBJID ptn_object_ID [, ptn_object_ID]
;
```

Syntax for MAP

```
MAP source_table, TARGET target_table
[, MOD_COMPARE_COLS( tgt_col = source )]
[, COLMAP (column_mapping)]
[, COMPARECOLS (column_specification)]
[, COORDINATED]
[, {DEF | TARGETDEF} template]
[, EXCEPTIONSONLY]
[, EXITPARAM 'parameter']
[, EVENTACTIONS (action)]
[, FILTER (filter_clause)]
[, HANDLECOLLISIONS | NOHANDLECOLLISIONS]
[, INSERTALLRECORDS]
[, INSERTAPPEND | NOINSERTAPPEND]
[, KEYCOLS (columns)]
[, MAPALLCOLUMNS | NOMAPALLCOLUMNS]
[, MAPEXCEPTION (exceptions_mapping)]
```

```
[, MAPINVISIBLECOLUMNS | NOMAPINVISIBLECOLUMNS]
[, REPERROR (error, response)]
[, RESOLVECONFLICT (conflict_resolution_specification)]
[, SQLEXEC (SQL_specification)]
[, THREAD (thread_ID)]
[, THREADRANGE (thread_range[, column_list])]
[, TRIMSPACES | NOTRIMSPACES]
[, TRIMVARSPACES | NOTRIMVARSPACES]
[, WHERE (clause)]
[, container.]schema.table PARTITIONOBJID ptn_object_ID [, ptn_object_ID]
;
```

TABLE and MAP Options

The following table summarizes the options that are available for the TABLE and MAP parameters. Note that not all options are valid for both parameters.

Table 2-11 Summary of TABLE and MAP Syntax Components

Component	Description	Valid For
MAP MOD_COMPARE_COL(tgt_col = source [,...])	This is a Replicat only parameter. Assigns specified source value to target column's before image as key value, and the value is used for WHERE clause.	MAP
TABLE *source_table*[, TARGET *target*]	Specifies the source object in a TABLE statement for Extract and an optional mapping to a target object. Use in the Extract parameter file.	TABLE
MAP *source_table*, TARGET *target_table*	Specifies the source-target object mapping for the Replicat process. Use in the Replicat parameter file.	MAP
ATTRCHARSET *(charset)*	specifies the source character set information at UDT attribute level.	TABLE
COLCHARSET *character_set*	Specifies any supported character set.	TABLE
COLMAP *(column_mapping)*	Maps records between different source and target columns.	TABLE and MAP

Table 2-11 (Cont.) Summary of TABLE and MAP Syntax Components

Component	Description	Valid For
{COLS | COLSEXCEPT} (*column_list*)	Selects or excludes columns for processing.	TABLE
COMPARECOLS (*column_specification*)	Specifies columns to use for conflict detection and resolution.	TABLE and MAP
COORDINATED	Forces a transaction to be processed as a barrier transaction.	MAP
{DEF| TARGETDEF} *template*	Specifies a source-definitions or target-definitions template.	TABLE and MAP
EXCEPTIONSONLY	Specifies that the MAP statement is an exceptions MAP statement.	MAP
EVENTACTIONS (*action*)	Triggers an action based on a record that satisfies a specified filter rule.	TABLE and MAP
EXITPARAM '*parameter*'	Passes a parameter in the form of a literal string to a user exit.	TABLE and MAP
FETCHBEFOREFILTER	Directs the FETCHCOLS or FETCHCOLSEXCEPT action to be performed before a filter is executed.	TABLE
{FETCHCOLS | FETCHCOLSEXCEPT} (*column_list*)	Enables the fetching of column values from the source database when the values are not in the transaction record.	TABLE
{FETCHMODCOLS | FETCHMODCOLSEXCEPT} (*column_list*)	Forces column values to be fetched from the database when the columns are present in the transaction log.	TABLE

Table 2-11 (Cont.) Summary of TABLE and MAP Syntax Components

Component	Description	Valid For
FILTER (*filter_clause*)	Selects records based on a numeric value. FILTER provides more flexibility than WHERE.	TABLE and MAP
GETBEFORECOLS (*column_specification*)	Forces before images of columns to be captured and written to the trail.	TABLE
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.	MAP
INSERTALLRECORDS	Applies all row changes as inserts.	MAP
INSERTAPPEND | NOINSERTAPPEND	Controls whether or not Replicat uses an Oracle APPEND hint for INSERT statements.	MAP
KEYCOLS (*columns*)	Designates columns that uniquely identify rows.	TABLE and MAP
MAPALLCOLUMNS| NOMAPALLCOLUMNS	Controls whether or not Replicat obtains non-key columns.	NA
MAPEXCEPTION (*exceptions_mapping*)	Specifies that the MAP statement contains exceptions handling for wildcarded tables.	MAP

Table 2-11 (Cont.) Summary of TABLE and MAP Syntax Components

Component	Description	Valid For
MAPINVISIBLECOLUMNS | NOMAPINVISIBLECOLUMNS	Controls whether or not Replicat includes invisible columns in Oracle target tables for default column mapping. For invisible columns in Oracle target tables that use explicit column mapping, they are always mapped so do not require this option.	MAP
REPERROR (*error*, *response*)	Controls how Replicat responds to errors when executing the MAP statement.	MAP
RESOLVECONFLICT (*conflict_resolution_specification*)	Specifies rules for conflict resolution.	MAP
SQLEXEC (*SQL_specification*)	Executes stored procedures and queries.	TABLE and MAP
SQLPREDICATE 'WHERE *where_clause*'	Enables a WHERE clause to select rows for an initial load.	TABLE
THREAD (*thread_ID*)	Valid for Replicat in coordinated mode. Specifies that the MAP statement will be processed by the specified Replicat thread.	MAP
THREADRANGE (*thread_range*, *column_list*)	Valid for Replicat in coordinated mode. Specifies that the MAP statement will be processed by the specified range of Replicat threads.	MAP
TOKENS (*token_definition*)	Defines user tokens.	TABLE
TRIMSPACES | NOTRIMSPACES	Controls whether trailing spaces are trimmed or not when mapping CHAR to VARCHAR columns.	TABLE and MAP

Table 2-11 (Cont.) Summary of TABLE and MAP Syntax Components

Component	Description	Valid For
TRIMVARSPACES | NOTRIMVARSPACES	Controls whether trailing spaces are trimmed or not when mapping VARCHAR to CHAR or VARCHAR columns.	TABLE and MAP
WHERE (*clause*)	Selects records based on conditional operators.	TABLE and MAP
;	(Semicolon) Terminates the TABLE or MAP statement and is required.	TABLE and MAP
PARTITIONOBJID	Available for Integrated Extract. This option is used to specify the object IDs of the partitions to be captured for partitioned tables.	TABLE and MAP

MAP MOD_COMPARE_COL(tgt_col = source [,...])

tgt_col must be target table column name, and should be the key column to take effect properly. source can be source table column, constant value (number or string), column mapping function or SQLEXEC results. For example, source col1 is mapped to target col1. source col1 before image value is 1, which is a dummy value because it is masked by DBA for security. Replicat can query actual before image value of target col1 by SQLEXEC. Using MOD_COMPARE_COLS(), Replicat replaces dummy source col1 value by SQLEXEC result, so that UPDATE or DELETE operation works properly.

TABLE *source_table*[, TARGET *target*]

TABLE is valid in an Extract parameter file.

Use TABLE to specify a source object for which you want Extract to capture data. Specify the fully qualified two-part or three-part name of the object, such as *schema.table* or *catalog.schema.table*. You can use a wildcard to specify multiple objects with one TABLE statement. To specify object names and wildcards correctly, see Using Wildcards in Database Object Names and Using Wildcards in Command Arguments in *Oracle GoldenGate Microservices Documentation*.

Use the TARGET option only when Extract must refer to a target definitions file (specified with the TARGETDEFS parameter) to perform conversions or when the COLMAP option is used to map columns. Otherwise, it can be omitted from a TABLE parameter. Column mapping with COLMAP and conversion work usually are performed on the target system to minimize the impact of replication activities on the source system, but can be performed on the source system if required. For example, column mapping and conversion can be performed on the source

system in a configuration where there are multiple sources and one target. In this scenario, it may be easier to manage one target definitions file rather than managing a definitions file for each source database, especially if there are frequent application changes that require new definitions files to be generated.

Using TARGET in a TABLE parameter identifies the metadata of the extracted data based on 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. Do not use three-part names if TARGET specifies tables in a target Oracle container database. Replicat can only connect to one container or catalog, so it is assumed that the container or catalog portion of the name is the same as the one that Replicat logs into (as specified with USERID, USERIDALIAS, or TARGETDB, depending on the database).

If no other TABLE syntax options are required to process the specified source data, you can use a simple TABLE statement, making sure to terminate it with a semicolon.

```
TABLE sales.customers;
```

The following shows the use of a wildcard to specify multiple tables:

```
TABLE sales.*;
```

The preceding TABLE statements direct Extract to capture all supported column data for the specified objects and write it to the trail without performing any filtering, conversion, or other manipulation.

MAP *source_table*, TARGET *target_table*

MAP is valid in a Replicat parameter file. You can also use MAP in a Extract parameter file to change the name of the transactions that Oracle GoldenGate stores for the table. See

Use MAP to specify a source object, and use TARGET to specify the target object to which Replicat applies the replicated source data. Together, the MAP and TARGET clause comprise a *mapping*.

- For MAP *source_table*, specify the source object. Specify the fully qualified two-part or three-part name of the object, such as *schema.table* or *catalog.schema.table*. You can use a wildcard to specify multiple source objects.
- For TARGET *target_table*, specify a two-part name, even if the target is a container database. Replicat can only connect to one container or catalog, so it is assumed that the container or catalog portion of the name is the same as the one that Replicat logs into (as specified with USERID, USERIDALIAS, or TARGETDB, depending on the database). You can use a wildcard to specify multiple target objects.

The following shows the use of a wildcard to specify multiple tables. Note that the TARGET clause does not include the tab prefix before the wildcard. That specification would be invalid, because the wildcard would be resolved as sales.tabtab1, sales.tabtab2, and so forth.

```
MAP sales.tab*, TARGET sales.*;
```

If no filtering, mapping, or other work is required for the objects, you can use simple MAP statements like the following, making sure to terminate each one with a semicolon.

```
MAP sales.customers, TARGET sales.customers;
MAP fin.*, TARGET fin.*;
```

ATTRCHARSET *(charset)*

ATTRCHARSET is valid for TABLE.

Use the ATTRCHARSET clause to specify the source character set information at UDT attribute level. It overrides the character set defined in the trail file or specified by SOURCECHARSET, CHARSET, or COLCHARSET parameters.

Valid values are character set names and valid UDT attribute names. Wildcard attribute names are supported. For example:

```
TABLE SCHEMA.T*,
  ATTRCHARSET(WE8DEC, col*.attr1, col1.attr*.attr3);
```

COLCHARSET *character_set*

COLCHARSET is valid for TABLE.

Use the COLCHARSET clause to specify any supported character set. See COLCHARSET for more information.

COLMAP *(column_mapping)*

COLMAP is valid for TABLE and MAP.

Use COLMAP to:

- Map individual source columns to target columns when the source and target columns have different names.
- Specify default column mapping when the source and target names are identical.

COLMAP provides instructions for selecting, translating, and moving column data from a source column to a target column.

> **Note:**
>
> To create *global* rules for column mapping across all tables in subsequent MAP statements, use the COLMATCH parameter.

Getting More Information About Configuring Column Mapping

To use COLMAP, related configuration considerations must be taken into account, such as whether source and target column structures are identical or different and whether global column mapping parameters may be sufficient.

Syntax

```
COLMAP (
[USEDEFAULTS, ]
target_column = source_expression [BINARYINPUT]
[, ...]
)
```

USEDEFAULTS
Automatically maps source and target columns that have the same name if they were not specified in an explicit column mapping. The data types are translated automatically, as needed, based on the local data-definitions file. USEDEFAULTS eliminates the need for an explicit column mapping if those columns have the same name and the data does not require any filtering or conversion.
Specify USEDEFAULTS before explicit column mappings in the COLMAP clause.

target_column = source_expression
Defines an explicit source-target column mapping.

target_column
Specifies the name of the target column. For supported characters in column names, see Supported Character Sets.

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 single quotes, such as `'ABCD'`
- An expression using an Oracle GoldenGate column-conversion function, such as `@STREXT (COL1, 1, 3)`. See "Table and Column Mapping Functions" for more information.

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. he source input is handled as binary input, and replacement of data values is suppressed.

Example 1

```
MAP ggs.tran, TARGET ggs.tran2, COLMAP (loc2 = loc, type2 = type);
```

Example 2

```
TABLE ggs.tran, COLMAP (SECTION = @STRCAT('\u00a7', SECTION ));
```

{COLS | COLSEXCEPT} (*column_list*)

`COLS` and `COLSEXCEPT` are valid for `TABLE`.

Use `COLS` and `COLSEXCEPT` to control the columns for which data is captured.

- `COLS` specifies columns that contain the data that you want to capture. When `COLS` is used, all columns that are not in the `COLS` list are ignored by Oracle GoldenGate.
- `COLSEXCEPT` specifies columns to exclude from being captured. When `COLSEXCEPT` is used, all columns that are not in the `COLSEXCEPT` list are captured by Oracle GoldenGate. For tables with numerous columns, `COLSEXCEPT` may be more efficient than listing each column with `COLS`.

> **Caution:**
>
> Do *not* exclude key columns, and do *not* use `COLSEXCEPT` to exclude columns that contain data types that are not supported by Oracle GoldenGate. `COLSEXCEPT` does not exclude unsupported data types.

To use `COLS`, the following is required:

- The table must have one or more key columns, or a substitute key must be defined with the `KEYCOLS` option. See "`KEYCOLS (columns)`".

- The key columns or the columns specified with KEYCOLS must be included in the column list that is specified with COLS. Otherwise, they will not be captured, and an error will be generated during processing.

Without a primary key, a unique key, or a KEYCOLS clause in the TABLE statement, Oracle GoldenGate uses all of the columns in the table, rendering COLS unnecessary.

> **Note:**
>
> Do not use this option for tables that are processed in pass-through mode by a data-pump Extract group.

Syntax

```
{COLS | COLSEXCEPT} (column [, ...])
```

column
The name of a column. To specify multiple columns, create a comma-delimited list, for example:

```
COLS (name, city, state, phone)
```

> **Note:**
>
> If the database only logs values for columns that were changed in an update operation, a column specified for capture with COLS might not be available. To make those columns available, use the FETCHCOLS option in the TABLE statement or enable supplemental logging for the column.

Example

The COLS clause in this example captures *only* columns 1 and 3, whereas the COLSEXCEPT clause captures all columns *except* columns 1 and 3.

```
TABLE hq.acct, COLS (col1, col3);
TABLE hq.sales, COLSEXCEPT (col1, col3);
```

COMPARECOLS (*column_specification*)

COMPARECOLS is valid for MAP.

Use COMPARECOLS to specify the columns that Replicat uses to detect and resolve update or delete conflicts when configured with the RESOLVECONFLICT option of MAP in a multi-master configuration. A conflict is a mismatch between the before image of a record in the trail and the correct data in the target table.

To use COMPARECOLS, the before image must be available in the trail record by means of the GETBEFORECOLS parameter in the Extract TABLE statement. The specified columns must exist in the target database and also be part of the Replicat configuration (satisfy the TARGET specification with or without a COLMAP clause).

Only scalar data types are supported by COMPARECOLS as comparison columns. A scalar data type can be used in a WHERE clause, has a single, atomic value and no internal components. Scalar data types supported by Oracle GoldenGate include the following, but not LOBs.

- Numeric data types
- Date data types
- Character data types

Some examples of non-scalar data types are spatial data, user-defined data types, large objects (LOB), XML, reference data types, and RAW. A row being considered for CDR can include non-scalar data so long as the conflict is not in the non-scalar data itself.

To specify conflict resolution routines, use the `RESOLVECONFLICT` option of `MAP`. `COMPARECOLS` and `RESOLVECONFLICT` can be in any order in the `MAP` statement.

Getting More Information About Configuring the CDR Feature

See Automatic Conflict Detection and Resolution or Manual Conflict Detection and Resolution for more information about configuring conflict detection and resolution.

Syntax

```
COMPARECOLS(
{ON UPDATE | ON DELETE}
{ALL | KEY | KEYINCLUDING (col[,...]) | ALLEXCLUDING (col[,...]) }
[,...]
)
```

{ON UPDATE | ON DELETE}
Specifies whether the before image of the specified columns should be compared for updates or deletes. You can use `ON UPDATE` only, `ON DELETE` only, or both. If using both, specify them within the same `COMPARECOLS` clause. See the example for how to use both.

{ALL | KEY | KEYINCLUDING (*col*[,...]) | ALLEXCLUDING (*col*[,...])}
Specifies the columns for which a before image is captured.

ALL
Compares using all columns in the target table. An error is generated if any corresponding before images are not available in the trail. Using `ALL` imposes the highest processing load for Replicat, but allows conflict-detection comparisons to be performed using all columns for maximum accuracy.

KEY
Compares only the primary key columns. This is the fastest option, but does not permit the most accurate conflict detection, because keys can match but non-key columns could be different.

KEYINCLUDING
Compares the primary key columns and the specified column or columns. This is a reasonable compromise between speed and detection accuracy.

ALLEXCLUDING
Compares all columns except the specified columns. For tables with numerous columns, `ALLEXCLUDING` may be more efficient than `KEYINCLUDING`. Do *not* exclude key columns.

Example 1

In the following example, the key columns plus the `name`, `address`, and `salary` columns are compared for conflicts.

```
MAP src, TARGET tgt
COMPARECOLS (
```

```
ON UPDATE KEYINCLUDING (name, address, salary),
ON DELETE KEYINCLUDING (name, address, salary));
```

Example 2
In the following example, the `comment` column is ignored and all other columns are compared for conflicts.

```
MAP src, TARGET tgt
COMPARECOLS (ON UPDATE ALLEXCLUDING (comment))
```

COORDINATED

`COORDINATED` is valid for `MAP`. This option is valid when Replicat is in coordinated mode.

Use the `COORDINATED` option to force transactions made on objects in the same `MAP` statement to be processed as barrier transactions. It causes all of the threads across all `MAP` statements to synchronize to the same trail location. The synchronized position is the beginning of the transaction that contains a record that satisfies a `MAP` that contains the `COORDINATED` keyword. The transaction is then applied atomically by a single thread, which is either the thread with the lowest thread ID among the currently running threads or a dedicated thread with the ID of 0 if `USEDEDICATEDCOORDINATIONTHREAD` is specified in the parameter file.

`THREAD` and `THREADRANGE` clauses specified in conjunction with `COORDINATED` are ignored because the record will not be applied by the designated thread(s). The `COORDINATED` keyword results in temporarily suspending parallelism so that the target tables are in a consistent state before the force-coordinated transaction is applied. After this point, parallel execution commences again.

Replicat by default coordinates transactions in which the primary key is updated, transactions that perform DDL, and certain `EVENTACTIONS` actions. `COORDINATED` provides for explicit coordination.

See About Coordinated Replicat for more information.

Syntax

```
COORDINATED
```

Example

The following is an example of the use of the `COORDINATED` option. In this example, business rules require that the target tables be in a consistent state before Replicat executes transactions that include `SQLEXEC` operations on the objects specified in the `MAP` statement. Parallelism must be temporarily converted to serial SQL processing in this case.

Given the following `MAP` statement, if another thread inserts into `t2` a record with a value of 100 for `col_val` before the insert to `t1` is performed by thread 1, then the `SQLEXEC` will delete the row. If other threads are still processing the record that has the value of 100, the `SQLEXEC` fails. The results of this `MAP` statement are, therefore, not predictable.

```
MAP u1.t1, TARGET u2.t1 SQLEXEC (ID test2, QUERY ' delete from u2.t2 where col_val =100
', NOPARAMS)), THREAD(1);
```

Conversely, when `COORDINATED` is used, all of the threads synchronize at a common point, including the one processing the `col_val=100` record, thereby removing the ambiguity of the results.

```
MAP u1.t1, TARGET u2.t1 SQLEXEC (ID test2, QUERY ' delete from u2.t2 where col_val =100
', NOPARAMS)), THREAD(1), COORDINATED;
```

{DEF| TARGETDEF} ***template***

DEF and TARGETDEF are valid for TABLE and MAP.

Use DEF and TARGETDEF to specify the name of a definitions template that was created by the DEFGEN utility.

- DEF specifies a source-definitions template.
- TARGETDEF specifies a target-definitions template.

A template is based on the definitions of a specific table. It enables new tables that have the same definitions as the original table to be added to the Oracle GoldenGate configuration without running DEFGEN for them, and without having to stop and start the Oracle GoldenGate process. The definitions in the template are used for definitions lookups.

Syntax

```
{DEF | TARGETDEF} template
```

template
The name of one of the following definitions templates generated by the DEFGEN utility:

- Use DEF to specify a source-definitions template generated by the DEF option of the TABLE parameter in the DEFGEN parameter file.
- Use TARGETDEF to specify a target-definitions template generated by the TARGETDEF option of the TABLE parameter in the DEFGEN parameter file.

The definitions contained in the template must be identical to the definitions of the table or tables that are specified in the same TABLE or MAP statement.
Case-sensitivity of the template name is observed when the name is specified the same way that it is stored in the database. Make certain that the template name is specified the same way in both the DEF or TARGETDEF clause in this TABLE or MAP statement, and in the DEFGEN parameter file that created the template.

Example 1
This example shows a case-insensitive template name.

```
MAP acct.cust*, TARGET acct.cust*, DEF custdef;
```

Example 2
This example shows a case-sensitive template name when the database requires quotes to enforce case-sensitivity.

```
TABLE acct.cust*, DEF "CustDef";
```

Example 3
This example shows a case where both DEF and TARGETDEF are used.

```
MAP acct.cust*, TARGET acc.cust*, DEF custdef, TARGETDEF tcustdef;
```

EXCEPTIONSONLY

EXCEPTIONSONLY is valid for MAP.

Use EXCEPTIONSONLY in an exceptions MAP statement intended for error handling. The exceptions MAP statement must follow the MAP statement for which errors are anticipated. The exceptions MAP statement executes only if an error occurs for the last record processed in the preceding regular MAP statement.

To use EXCEPTIONSONLY, use a REPERROR statement with the EXCEPTION option either within the regular MAP statement or at the root of the parameter file. See "REPERROR" for more information.

> **Note:**
>
> If using the Oracle GoldenGate Conflict Detection and Resolution (CDR) feature, a REPERROR with EXCEPTION is not needed. CDR automatically sends all operations that cause errors to the exceptions MAP statement.

The exceptions MAP statement must specify the same source table as in the regular MAP statement, but the target table in the exceptions MAP statement must be an exceptions table.

> **Note:**
>
> See "MAPEXCEPTION *(exceptions_mapping)*" to support wildcarded object names.

Syntax

```
EXCEPTIONSONLY
```

```
EVENTACTIONS (action)
```

EVENTACTIONS is valid for TABLE and MAP. Some options apply only to one or the other parameter and are noted as such in the descriptions.

Use EVENTACTIONS to cause the 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, known as the *event marker system* (or *event marker infrastructure*) to customize processing based on database events. For example, you can suspend a process to perform a transformation or report statistics. The event marker feature is supported for the replication of data changes, but not for initial loads.

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 "TABLE for Replicat" for more information.

You may need to combine two or more actions to achieve your goals. When multiple actions are combined, the entire EVENTACTIONS statement is parsed first, and then the specified options execute in order of precedence. The following list shows the order of precedence. The actions listed before Process the record occur before the record is written to the trail or applied to the target (depending on the process). Actions listed after Process the record are executed after the record is processed.

TRACE
LOG
CHECKPOINT BEFORE
DISCARD
SHELL
ROLLOVER
(Process the record)

```
IGNORE
REPORT
SUSPEND
ABORT
CHECKPOINT AFTER
FORCESTOP
STOP
```

To prevent the event record itself from being processed in the normal manner, use the `IGNORE` or `DISCARD` option. Because `IGNORE` and `DISCARD` are evaluated before the record itself, they prevent the record from being processed. Without those options, `EVENTACTIONS` for Extract writes the record to the trail, and `EVENTACTIONS` for Replicat applies that operation 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 files empty of transaction data.

You should also take into account that when the `GETUPDATEBEFORES` parameter is in effect, two records are generated for `UPDATE` operations: a record that contains the before image and a record that contains the after image. An event action is triggered for each of those records when the operation qualifies as an event record. You can use the `BEFOREAFTERINDICATOR` token of the `GGHEADER` column-conversion function as a filter in a `FILTER` clause to qualify the records so that the event action triggers only once, either on the before record or the after record, but not both.

The following example filters on the `BEFORE` indicator. The `EVENTACTION` issues the `ECHO` shell command to output the string 'Triggered on `BEFORE`' to the event log when a `BEFORE` record is encountered.

```
TABLE qasource.test, &
FILTER(@STRFIND('BEFORE', @GETENV('GGHEADER' , 'BEFOREAFTERINDICATOR')) > 0), &
EVENTACTIONS ( shell ('echo --== Triggered on BEFORE ==-- '), LOG);
```

The following shows the result of the event action:

```
013-03-06 17:59:31  INFO    OGG-05301  Shell command output: '--== Triggered
on AFTER ==--'
```

The following example does the same thing, but for the `AFTER` indicator.

```
TABLE qasource.test, &
FILTER(@STRFIND('AFTER', @GETENV('GGHEADER' , 'BEFOREAFTERINDICATOR')) > 0), &
EVENTACTIONS ( shell ('echo --== Triggered on AFTER ==-- '), LOG);
```

Syntax

```
EVENTACTIONS (
[STOP | SUSPEND | ABORT | FORCESTOP]
[IGNORE [RECORD | TRANSACTION [INCLUDEVENT]]
[DISCARD]
[LOG [INFO | WARNING]]
[REPORT]
[ROLLOVER]
[SHELL 'command' |
```

```
  SHELL ('command', VAR variable = {column_name | expression}
  [, ...]) ]
[TRACE[2] file [TRANSACTION] [DDL[INCLUDE] | DDLONLY] [PURGE | APPEND]]
[CHECKPOINT [BEFORE | AFTER | BOTH]]
[, ...]
)
```

STOP

Valid in `TABLE` for Extract and in `MAP` for Replicat.

Brings the process to a graceful stop when the specified event record is encountered. The process waits for other operations within event transaction 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`.

SUSPEND

Valid in `TABLE` for Extract and in `MAP` for Replicat.

Pauses the process so that it retains the active context of the current run and can still respond to `SEND` commands that are issued in Admin Client. When a process is suspended, the `INFO` command shows it as `RUNNING`, and the `RBA` field shows the last checkpoint position.

To resume processing, issue the `SEND` command with the `RESUME` option.

To use the `CHECKPOINT BEFORE` option in conjunction with `SUSPEND`, the event record must be the start of a transaction for the `SUSPEND` to take place. That way, if the process is killed while in the suspended state, the event record with the `SUSPEND` action is the first record to be reprocessed upon restart. If both `CHECKPOINT BERORE` and `SUSPEND` are specified, but the event record is not the start of a transaction, the process abends before `SUSPEND` can take place.

To use the `CHECKPOINT AFTER` option in conjunction with `SUSPEND`, the `RESUME` command must be issued before the checkpoint can take place, and the event record must be a `COMMIT` record. If the process is killed while in a `SUSPEND` state, the process reprocesses the transaction from the last checkpointed position upon restart.

`SUSPEND` cannot be combined with `ABORT` but can be combined with all other options.

ABORT

Valid in `TABLE` for Extract and in `MAP` for Replicat.

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

Valid in `TABLE` for Extract and in `MAP` for Replicat.

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 [RECORD | TRANSACTION [INCLUDEVENT]]
Valid in TABLE for Extract and in MAP for Replicat.
Ignores some or all of the transaction, depending on the selected action.

- RECORD is the default. It forces the process to ignore only the specified event record, but not the rest of the transaction. No warning or message is written to the log, but the Oracle GoldenGate statistics are updated to show that the record was ignored.
- Use TRANSACTION to ignore the entire transaction that contains the record that triggered the event. If TRANSACTION is used, the event record must be the first one in the transaction. When ignoring a transaction, the event record is also ignored by default. TRANSACTION can be shortened to TRANS.
- Use INCLUDEEVENT with TRANSACTION to propagate the event record to the trail or to the target, but ignore the rest of the associated transaction.

IGNORE can be combined with all other EVENTACTIONS options except ABORT and DISCARD.
An IGNORE action is processed after all the qualification, filtering, mapping, and user-exit operations are processed. The record or transaction is ignored in the final output phase and prevents the record or transaction from being written to the output target (the trail in the case of Extract or the database in the case of Replicat). Therefore, in certain expressions, for example those that include SQLEXEC operations, the SQLEXEC will be executed before the IGNORE is processed. This means that, while the record is not written to the trail or target database, all of the effects of processing the record through qualification, filtering, mapping and user-exit will occur.
This action is not valid for DDL records. Because DDL operations are autonomous, ignoring a record is equivalent to ignoring the entire transaction.

DISCARD
Valid in TABLE for Extract and in MAP for Replicat.
Causes the process to:

- write the specified event record to the discard file.
- update the Oracle GoldenGate statistics to show that the record was discarded.

The process resumes processing with the next record in the trail.
DISCARD can be combined with all other EVENTACTIONS options except IGNORE.

LOG [INFO | WARNING]
Valid in TABLE for Extract and in MAP for Replicat.
Causes the process to log the event when the specified event record is encountered. The message is written to the report file, to the Oracle GoldenGate error log, and to the system event log.
Use the following options to specify the severity of the message:

- INFO specifies a low-severity informational message. This is the default.
- WARNING specifies a high-severity warning message.

LOG can be combined with all other EVENTACTIONS options except ABORT. If using ABORT, LOG is not needed because ABORT logs a fatal error before the process exits.

REPORT
Valid in TABLE for Extract and in MAP for Replicat.

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 in TABLE 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. ROLLOVER cannot be combined with ABORT because ROLLOVER does not cause the process to write a checkpoint, and 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*'
Valid in TABLE for Extract and in MAP for Replicat.
Causes the process to execute the specified shell command when the event record is encountered. SHELL '*command*' executes a basic shell command. The command string is taken at its literal value and sent to the system that way. The command is case-sensitive. Enclose the command string within single quote marks, for example:

```
EVENTACTIONS (SHELL 'echo hello world! > output.txt')
```

If the shell command is successful, the process writes an informational message to the report file and to the event log. Success is based upon the exit status of the command in accordance with the UNIX shell language. In that language, zero indicates success.
If the system call is not successful, the process abends with a fatal error. In the UNIX shell language, non-zero equals failure. Note that the error message relates only to the execution of the SHELL command itself, and not the exit status of any subordinate commands. For example, SHELL can execute a script successfully, but commands in that script could fail.
SHELL can be combined with all other EVENTACTIONS options.

SHELL ('*command*', VAR *variable* = {*column_name* | *expression*} [, ...])
Valid in TABLE for Extract and in MAP for Replicat.
Causes the process to execute the specified shell command when the event record is encountered and supports parameter passing. The command and the parameters are case-sensitive.
When SHELL is used with arguments, the entire command and argument strings must be enclosed within parentheses, for example:

```
EVENTACTIONS (SHELL
('Current timestamp: $1  SQLEXEC result is $2 ',VAR $1 = @GETENV('JULIANTIMESTAMP'),
VAR $2 = mytest.description));
```

The input is as follows:

command
Is the command, which is passed literally to the system.

VAR
Is a required keyword that starts the parameter input.

variable
Is the user-defined name of the placeholder variable where the run-time variable value will be substituted. Extra variables that are not used in the command are ignored. Note that any literal in the `SHELL` command that matches a `VAR` variable name is replaced by the substituted `VAR` value. This may have unintended consequences, so test your code before putting it into production.

column_name
Can be the before or after (current) image of a column value.

expression
can be the following, depending on whether column data or DDL is being handled.

- Valid expressions for column data:
  - The value from a `TOKENS` clause in a `TABLE` statement.
  - A return value from any Oracle GoldenGate column-conversion function.
  - A return value from a `SQLEXEC` query or procedure.
- Valid expressions for DDL:
  - Return value from `@TOKEN` function (Replicat only).
  - Return value from `@GETENV` function.
  - Return value from other functions that do not reference column data (for example, `@DATENOW`).
  - Return value from `@DDL` function.

***TRACE[2] file* [TRANSACTION] [DDL[INCLUDE] | DDLONLY] [PURGE | APPEND]**
Valid in `TABLE` for Extract and in `MAP` for Replicat.
Causes process trace information to be written to a trace file when the specified event record is encountered. `TRACE` provides step-by-step processing information. `TRACE2` identifies the code segments on which the process is spending the most time.
By default (without options), standard DML tracing without consideration of transaction boundaries is enabled until the process terminates.

- *file* specifies the name of the trace file and must appear immediately after the `TRACE` keyword. You can specify a unique trace file, or use the default trace file that is specified with the standalone `TRACE` or `TRACE2` parameter.

  The same trace file can be used across different `TABLE` or `MAP` statements in which `EVENTACTIONS TRACE` is used. If multiple `TABLE` or `MAP` statements specify the same trace file name, but the `TRACE` options are not used consistently, preference is given to the options in the last resolved `TABLE` or `MAP` that contains this trace file.
- Use `TRANSACTION` to enable tracing only until the end of the current transaction, instead of when the process terminates. For Replicat, transaction boundaries are based on the source transaction, not the typical Replicat grouped or batched target transaction. `TRANSACTION` can be shortened to `TRANS`. This option is valid only for DML operations.
- `DDL[INCLUDE]` traces DDL and also DML transactional data processing. Either `DDL` or `DDLINCLUDE` is valid.
- `DDLONLY` traces DDL but does not trace DML transactional data.

  These options are valid only for Replicat. By default DDL tracing is disabled.

- Use PURGE to truncate the trace file before writing additional trace records, or use APPEND to write new trace records at the end of the existing records. APPEND is the default.

TRACE can be combined with all other EVENTACTIONS options except ABORT.
To disable tracing to the specified trace file, issue the GGSCI SEND *process* command with the TRACE OFF *file_name* option.

CHECKPOINT [BEFORE | AFTER | BOTH]
Valid in TABLE for Extract and in MAP for Replicat.
Causes the process to write a checkpoint when the specified event record is encountered. Checkpoint actions provide a context around the processing that is defined in TABLE or MAP statements. This context has a begin point and an end point, thus providing synchronization points for mapping the functions that are performed with SQLEXEC and user exits.

BEFORE
BEFORE for an Extract process writes a checkpoint before Extract writes the event record to the trail. BEFORE for a Replicat process writes a checkpoint before Replicat applies the SQL operation that is contained in the record to the target.
BEFORE requires the event record to be the first record in a transaction. If it is not the first record, the process will abend. Use BEFORE to ensure that all transactions prior to the one that begins with the event record are committed.
When using EVENTACTIONS for a DDL record, note that since each DDL record is autonomous, the DDL record is guaranteed to be the start of a transaction; therefore the CHECKPOINT BEFORE event action is implied for a DDL record.
CHECKPOINT BEFORE can be combined with all EVENTACTIONS options.

AFTER
AFTER for Extract writes a checkpoint after Extract writes the event record to the trail.
AFTER for Replicat writes a checkpoint after Replicat applies the SQL operation that is contained in the record to the target.
AFTER flags the checkpoint request as an advisory, meaning that the process will only issue a checkpoint at the next practical opportunity. For example, in the case where the event record is one of a multi-record transaction, the checkpoint will take place at the next transaction boundary, in keeping with the Oracle GoldenGate data-integrity model.
When using EVENTACTIONS for a DDL record, note that since each DDL record is autonomous, the DDL record is guaranteed to be the end (boundary) of a transaction; therefore the CHECKPOINT AFTER event action is implied for a DDL record.
CHECKPOINT AFTER can be combined with all EVENTACTIONS options except ABORT.

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 Extract processes any trail record that has name = abc, it ignores the record.

```
TABLE fin.cust, &
WHERE (name = 'abc'), &
EVENTACTIONS (ignore);
```

Example 2
Based on the compatibility and precedence rules of `EVENTACTIONS` options, `DISCARD` takes higher precedence than `ABORT`, so in this example the event record gets written to the discard file before the process abends.

```
MAP fin.cust, TARGET fin.cust2, &
WHERE (name = 'abc'), &
EVENTACTIONS (DISCARD, ABORT);
```

Example 3
The following example executes a `SHELL` action. It gets the result of a `SQLEXEC` query and pairs it with the current timestamp.

```
TABLE src.tab &
SQLEXEC (id mytest, query 'select description from lookup &
where pop = :mycol2', params (mycol2 = col2) ), &
EVENTACTIONS (SHELL ('Current timestamp: $1  SQLEXEC result is $2 ', &
VAR $1 = @GETENV('JULIANTIMESTAMP'), VAR $2 = mytest.description));
```

The shell command that results from this example could be similar to the following:

```
'Current timestamp: 212156002704718000  SQLEXEC result is test passed'
```

Example 4
The following example shows how invalid results can occur if a placeholder name conflicts with literal text in the command string. In this example, a placeholder named `$1` is associated with a column value, and the `SHELL` command echoes a literal string that includes `$1`.

```
MAP src.tab1, TARGET targ.tab1 &
EVENTACTIONS (SHELL ('echo Extra charge for $1 is $1', VAR $1 = COL1));
```

This is the unintended result, assuming the column value is `gift wrap`:

```
'Extra charge for gift wrap is gift wrap'
```

Changing the placeholder variable to `$col` results in the correct output:

```
MAP src.tab1, TARGET targ.tab1 &
EVENTACTIONS (SHELL ('echo Extra charge for $col is $1', VAR $col = COL1));
'Extra charge for gift wrap is $1'
```

The following shows similar potential for unintended results:

```
MAP src.tab1, TARGET targ.tab1 &
EVENTACTIONS (SHELL ('Timestamp: $1  Price is $13 > out.txt ', &
VAR $1 = @GETENV('JULIANTIMESTAMP')));
```

The redirected output file might contain a string like this (notice the second timestamp contains an appended value of `3`):

```
'Timestamp: 212156002704718000 Price is 2121560027047180003'
```

The intended result is this:

```
'Timestamp: 212156002704718000 Price is $13'
```

Example 5
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 is generated just before the first tab1 event record is applied to the target. It contains all of the tracing information from that point forward until Replicat terminates or unless tracing is turned off with the GGSCI SEND REPLICAT command.
- Because the second MAP statement contains the TRANSACTION option, the trace2.txt file is generated just before the first tab2 event record is applied to the target, but the tracing stops automatically at the conclusion of the transaction that contains the tab2 event record.

Example 6
The following shows how EVENTACTIONS with SUSPEND can be used.

- *Case 1*: You are replicating DDL, and you want to ensure that there is enough space in the target database to create a new table. Use EVENTACTIONS with SUSPEND in the MAP statement that maps the CREATE TABLE DDL operation, and then execute a SQL statement in that MAP statement to query the amount of space remaining in a tablespace. If there is enough space, use SEND REPLICAT with RESUME to resume processing immediately; if not, leave Replicat suspended until a DBA can add the space, and then use SEND REPLICAT with RESUME to resume processing.
- *Case 2*: You want to fix unique key violations when they occur on any table. Because Replicat is processing thousands of tables, you do not want to stop the process each time there is a violation, because this would cause Replicat to spend time rebuilding the object cache again upon restart. By using EVENTACTIONS with SUSPEND, you can simply suspend processing until the problem is fixed.
- *Case 3*: At the end of the day, you suspend Replicat to run daily reports, and then resume processing immediately without stopping and restarting the process.

EXITPARAM '*parameter*'

EXITPARAM is valid for TABLE and MAP.

Use EXITPARAM to pass a parameter to the EXIT_PARAMS function of a user exit routine whenever a record from the TABLE or MAP statement is encountered.

Syntax

EXITPARAM '*parameter string*'

'parameter string'
A parameter that is a literal string. Enclose the parameter within single quotes. You can specify up to 100 characters for the parameter string.

FETCHBEFOREFILTER

FETCHBEFOREFILTER is valid for TABLE.

Use FETCHBEFOREFILTER to fetch columns that are specified with FETCHCOLS or FETCHCOLSEXCEPT before a FILTER operation is executed. Fetching before the filter 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. Specify FETCHBEFOREFILTER before FILTER in the parameter file.

Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

Syntax

```
FETCHBEFOREFILTER
```

Example

```
TABLE hr.salary, FETCHCOLS (sal_level),
FETCHBEFOREFILTER,
FILTER (sal_level >= 8)
;
```

{FETCHCOLS | FETCHCOLSEXCEPT} (*column_list*)

FETCHCOLS and FETCHCOLSEXCEPT are valid for TABLE. These options are only valid for the primary Extract.

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 only logs the values of columns that were changed in an update operation, but you need to ensure that other column values required for FILTER operations are available.

- FETCHCOLS fetches the specified columns.
- FETCHCOLSEXCEPT fetches all columns except the specified columns. For tables with numerous columns, FETCHCOLSEXCEPT may be more efficient than listing each column with FETCHCOLS.

FETCHCOLS and FETCHCOLSEXCEPT are valid for all databases that are supported by Oracle GoldenGate.

For an Oracle Database, Oracle GoldenGate fetches the values from the undo tablespace through Oracle's Flashback Query mechanism. The query provides a read-consistent image of the columns as of a specific time or SCN. For more information about how Oracle GoldenGate uses Flashback Query.

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.

Syntax

```
{FETCHCOLS | FETCHCOLSEXCEPT} (column [, ...])
```

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 (*).

Example

The FETCHCOLS clause in this example fetches *only* columns 1 and 3, whereas the FETCHCOLSEXCEPT clause fetches all columns *except* columns 1 and 3.

```
TABLE hq.acct, FETCHCOLS (col1, col3);
TABLE hq.sales, FETCHCOLSEXCEPT (col1, col3);
```

{FETCHMODCOLS | FETCHMODCOLSEXCEPT} (*column_list*)

FETCHMODCOLS and FETCHMODCOLSEXCEPT are valid for TABLE. These options are only valid for the primary Extract.

Use FETCHMODCOLS and FETCHMODCOLSEXCEPT to force column values to be fetched from the database even if the columns are present in the transaction log. These Depending on the database type, a log record can contain all of the columns of a table or only the columns that changed in the given transaction operation.

- FETCHMODCOLS fetches the specified columns.
- FETCHMODCOLSEXCEPT fetches all columns that are present in the transaction log, except the specified columns. For tables with numerous columns, FETCHMODCOLSEXCEPT might be more efficient than listing each column with FETCHMODCOLS.

FETCHMODCOLS and FETCHMODCOLSEXCEPT are valid for all databases that are supported by Oracle GoldenGate.

Observe the following usage guidelines:

- Do not use FETCHMODCOLS and FETCHMODCOLSEXCEPT for key columns.

Syntax

```
{FETCHMODCOLS | FETCHMODCOLSEXCEPT} (column [, ...])
```

(*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 (*).

Example

The FETCHMODCOLS clause in this example fetches *only* columns 1 and 3, whereas the FETCHMODCOLSEXCEPT clause fetches all columns *except* columns 1 and 3.

```
TABLE hq.acct, FETCHMODCOLS (col1, col3);
TABLE hq.sales, FETCHMODCOLSEXCEPT (col1, col3);
```

FILTER (*filter_clause*)

FILTER is valid for TABLE and MAP.

Use FILTER to select or exclude records based on a numeric value. A filter expression can use conditional operators, Oracle GoldenGate column-conversion functions, or both.

> **Note:**
>
> To filter based on a string, use one of the Oracle GoldenGate string functions. See "Table and Column Mapping Functions" for more information about these functions. You can also use the WHERE option. See "WHERE (*clause*)".

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`

Enclose literals in single quotes. Specify case-sensitive column names as they are stored in the database, and enclose them in double quotes if the database requires quotes to enforce case-sensitivity (such as Oracle).

Oracle GoldenGate supports `FILTER` for columns that have a multi-byte character set.

Syntax

```
FILTER (
[, ON INSERT | ON UPDATE| ON DELETE]
[, IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE]
, filter_clause
[, RAISEERROR error_number]
)
```

filter_clause
Selects records based on an expression, such as:

```
FILTER ((PRODUCT_PRICE*PRODUCT_AMOUNT) > 10000))
```

You can use the column-conversion functions of Oracle GoldenGate in a filter clause, as in:

```
FILTER (@COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)
```

Enclose literals in single quotes. Specify case-sensitive column names as they are stored in the database, and enclose them in double quotes if the database requires quotes to enforce case-sensitivity (such as Oracle).
Oracle GoldenGate does not support `FILTER` for columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.
The maximum size of the filter clause is 5,000 bytes.

`ON INSERT | ON UPDATE | ON DELETE`
Restricts record filtering to the specified operation(s). Separate operations with commas, for example:

```
FILTER (ON UPDATE, ON DELETE,
@COMPUTE (PRODUCT_PRICE*PRODUCT_AMOUNT)>10000)
```

The preceding example executes the filter for `UPDATE` and `DELETE` operations, but not `INSERT` operations.

`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)
```

The preceding example executes the filter on `UPDATE` and `DELETE` operations, but ignores `INSERT` operations.

`RAISEERROR` *`error`*
Raises a user-defined error number if the filter fails. Can be used as input to the `REPERROR` parameter to invoke error handling. Make certain that the value for *`error`* is outside the range of error numbers that is used by the database or by Oracle GoldenGate. For example: `RAISEERROR 21000`.

`GETBEFORECOLS (`*`column_specification`*`)`

`GETBEFORECOLS` is valid for `TABLE`.

Use `GETBEFORECOLS` to specify columns for which you want before image to be captured and written to the trail upon an update or delete operation. Use `GETBEFORECOLS` when using the Oracle GoldenGate Conflict Detection and Resolution (CDR) feature in a bi-directional or multi-master configuration. Also use it when using conversion functions or other processing features that require the before image of a record.

For updates, the before image of the specified columns is included in the trail whether or not any given column is modified. In addition to the columns specified in the `GETBEFORECOLS` clause, an Oracle database will also log the before image of other columns that are modified. For other supported databases, you can use the `GETUPDATEBEFORES` parameter to force the inclusion of the before values of other columns that are modified.

> **Note:**
>
> `GETUPDATEBEFORES` overrides `GETBEFORECOLS` if both are used in the same parameter file.

To use this parameter, supplemental logging must be enabled for any database that does not log before values by default.

GETBEFORECOLS overrides COMPRESSUPDATES and COMPRESSDELETES if used in the same parameter file.

This parameter is valid for all databases except DB2 z/OS. For DB2 z/OS on all platforms that are supported by Oracle GoldenGate, use the GETUPDATEBEFORES parameter instead of GETBEFORECOLS.

Syntax

```
GETBEFORECOLS(
{ON UPDATE | ON DELETE}
{ALL | KEY | KEYINCLUDING (col[,...])  | KEYANDMOD | | ALLEXCLUDING (col[,...]) }
[,...]
)
```

{ON UPDATE | ON DELETE}
Specifies whether the before image of the specified columns should be captured for updates or deletes. You can use ON UPDATE only, ON DELETE only, or both. If using both, specify them within the same GETBEFORECOLS clause. See the example for how to use both.

{ALL | KEY | KEYINCLUDING (*col*[,...]) | KEYANDMOD | ALLEXCLUDING (*col*[,...])}
Specifies the columns for which a before image is captured.

ALL
Captures a before image of all supported data type columns in the target table, including the primary key; all unsupported columns are skipped and logged in the Extract or Replicat parameter file as an information message. This imposes the highest processing load for Extract, but allows conflict-detection comparisons to be performed using all columns for maximum accuracy.

KEY
Capture before image only for the primary key. This is the fastest option, but does not permit the most accurate conflict detection, because keys can match but non-key columns could be different. KEY is the default.

KEYINCLUDING
Capture before image of the primary key and also the specified column or columns. This is a reasonable compromise between speed and detection accuracy.

KEYANDMOD
Use this option as an extension of the key option for both Extract and Replicat. For update DMLs on the source, Extract logs the key and modified columns. Replicat on the target will use the KEY and MODIFIED columns during conflict detection in a WHERE clause. With Oracle databases, the modified column is always used for conflict detection by default and this parameter makes it explicit.

ALLEXCLUDING
Capture before image of all columns except the specified columns. For tables with numerous columns, ALLEXCLUDING may be more efficient than KEYINCLUDING. Do *not* exclude key columns.

Example

In the following example, the before images for the key column(s) plus the name, address, and salary are always written to the trail file on update and delete operations.

```
TABLE src,
GETBEFORECOLS (
ON UPDATE KEYINCLUDING (name, address,  salary),
ON DELETE KEYINCLUDING (name, address, salary));
```

HANDLECOLLISIONS | NOHANDLECOLLISIONS

HANDLECOLLISIONS and NOHANDLECOLLISIONS are valid for MAP.

Use HANDLECOLLISIONS and NOHANDLECOLLISIONS to control whether or not Oracle GoldenGate reconciles the results of an initial load with replicated transactional changes that are made to the same tables. When Oracle GoldenGate applies replicated changes after the load is finished, HANDLECOLLISIONS causes Replicat to overwrite duplicate records in the target tables and provides alternate handling of errors for missing records.

HANDLECOLLISIONS and NOHANDLECOLLISIONS can be used globally for all MAP statements in the parameter file or as an ON/OFF switch for groups of tables specified with MAP statements, and they can be used within a MAP statement. When used in a MAP statement, they override the global specifications.

See "HANDLECOLLISIONS | NOHANDLECOLLISIONS" for syntax and usage.

INSERTALLRECORDS

INSERTALLRECORDS is valid for MAP.

Use the INSERTALLRECORDS parameter to convert all mapped operations to INSERT operations on the target. INSERTALLRECORDS can be used at the root level of the parameter file, within a MAP statement, and within a MAPEXCEPTION clause of a MAP statement.

See "INSERTALLRECORDS" for syntax and usage.

INSERTAPPEND | NOINSERTAPPEND

INSERTAPPEND is valid for MAP.

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

See "INSERTAPPEND | NOINSERTAPPEND" for syntax and usage.

KEYCOLS (*columns*)

KEYCOLS is valid for TABLE and MAP.

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 an appropriate unique index is not available for the table. You can also use KEYCOLS to specify additional columns to use in the row identifier that Replicat uses. Without the availability of a key or KEYCOLS clause, Replicat uses all columns of the table to build its WHERE clause, essentially performing a full table scan.

The columns of a key rendered by KEYCOLS must uniquely identify a row, and they must match the columns that are used as a key on the source table. The source table must contain at least as many key or index columns as the KEYCOLS key specified for 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 KEYCOL columns.

When defining a substitute key with KEYCOLS, observe the following guidelines:

- If the source and target tables both lack keys or unique indexes, use a KEYCOLS clause in the TABLE parameter and in the MAP parameter, and specify matching sets of columns in each KEYCOLS clause.
- If either of the tables lacks a key or unique index, use KEYCOLS for that table. Specify columns that match the actual key or index columns of the other table. If a matching set cannot be defined with KEYCOLS, you must use KEYCOLS for the source table (TABLE parameter) and for the target table (MAP parameter). Specify matching sets of columns that contain unique values. KEYCOLS overrides a key or unique index.
- If the target table has a larger key than the source table does (or if it has more unique-index columns), use KEYCOLS in the TABLE statement to specify the source columns that match the extra target columns. You must also include the actual source key or index columns in this KEYCOLS clause. Using KEYCOLS in this way ensures that before images are available to Replicat in case the non-key columns are updated on the source.

When using KEYCOLS, make certain that the specified columns are configured for logging so that they are available to Replicat in the trail records. For an Oracle database, you can enable the logging by using the COLS option of the ADD TRANDATA command.

On the target tables, create a unique index on the KEYCOLS-defined key columns. An index improves the speed with which Oracle GoldenGate locates the target rows that it needs to process.

Do not use KEYCOLS for tables being processed in pass-through mode by a data-pump Extract group.

Additional Considerations for KEYCOLS when using Parallel Replicat or Integrated Replicat:

- When using KEYCOLS with ALLOWDUPTARGETMAP, the key columns must be the same for each mapped table. For example, if you map HR.EMP to HR.EMP_TARGET and HR.EMP_BACKUP and if you specify KEYCOLS, they must be the same for both HR.EMP_TARGET and HR.EMP_BACKUP.
- When using KEYCOLS to map from multiple source tables to the same target table, the MAP statements must use the same set of KEYCOLS.

Syntax

```
KEYCOLS (column [, ... ])
```

column
Defines a column to be used as a substitute primary key. If a primary or unique key exists, those columns must be included in the KEYCOLS specification. To specify multiple columns, create a comma-delimited list as in:

```
KEYCOLS (id, name)
```

The following column-types are **not** supported in KEYCOLS:

- Oracle column types **not** supported by KEYCOLS:

  Virtual columns, UDTs, function-based columns, and any columns that are explicitly excluded from the Oracle GoldenGate configuration.

- SQL Server, DB2 LUW, DB2 z/OS, MySQL, and Teradata:

  Columns that contain a timestamp or non-materialized computed column, and any columns excluded from the Oracle GoldenGate configuration. For SQL Server Oracle GoldenGate enforces the total length of data in rows for target tables without a primary key to be below 8000 bytes.

Example

```
TABLE hr.emp, KEYCOLS (id, first, last, birthdate);
```

MAPEXCEPTION (*exceptions_mapping*)

MAPEXCEPTIONS is valid for MAP.

Use MAPEXCEPTION as part of an exceptions MAP statement intended for error handling. MAPEXCEPTION maps failed operations that are flagged as exceptions by the REPERROR parameter to an *exceptions table*. Replicat writes the values of these operations along with other information to the exceptions table.

You can use MAPEXCEPTION within the same MAP statement that includes the source-target table mapping and other standard MAP options. The source and target table names can include wildcards.

When using MAPEXCEPTION, use a REPERROR statement with the EXCEPTION option either within the same MAP statement or at the root of the Replicat parameter file. See "EXCEPTIONSONLY" and "REPERROR".

Syntax

```
MAPEXCEPTION (TARGET exceptions_table, INSERTALLRECORDS [, exception_MAP_options])
```

TARGET *exceptions_table*
The fully qualified name of the exceptions table. Standard Oracle GoldenGate rules for object names apply to the name of the exceptions table.

exception_MAP_options
Any valid options of the MAP parameter that you want to apply to the exceptions handling.

INSERTALLRECORDS
Applies all exceptions to the exceptions table as INSERT operations. This parameter is required when using MAPEXCEPTION.

Example

This is an example of how to use MAPEXCEPTION for exceptions mapping. The MAP and TARGET clauses contain wildcard source and target table names. Exceptions that occur when processing any table with a name beginning with TRX will be captured to the fin.trxexceptions table using the specified mapping.

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

MAPALLCOLUMNS | NOMAPALLCOLUMNS

MAPALLCOLUMNS and NOMAPALLCOLUMNS are valid for MAP.

Use MAPALLCOLUMNS to obtain unmapped columns (non-key). When this option is specified, Extract or Replicat checks if all source columns are directly mapped to the target without the column mapping function. If any source columns isn't mapped, then the Extract and/or Replicat abends.

See "MAPALLCOLUMNS| NOMAPALLCOLUMNS

MAPINVISIBLECOLUMNS | NOMAPINVISIBLECOLUMNS

MAPINVISIBLECOLUMNS and NOMAPINVISIBLECOLUMNS are valid for MAP.

Use MAPINVISIBLECOLUMNS and NOMAPINVISIBLECOLUMNS to control whether or not Replicat includes invisible columns in Oracle target tables for default column mapping. For invisible columns in Oracle target tables that use explicit column mapping, they are always mapped so do not require this option.

MAPINVISIBLECOLUMNS and NOMAPINVISIBLECOLUMNS can be used in two different ways. When specified at a global level, one parameter remains in effect for all subsequent MAP statements, until the other parameter is specified. When used within a MAP statement, they override the global specifications

See "MAPINVISIBLECOLUMNS | NOMAPINVISIBLECOLUMNS" for syntax and usage.

REPERROR (*error*, *response*)

REPERROR is valid for MAP.

Use REPERROR to specify an error and a response that together control how Replicat responds to the error when executing the MAP statement. You can use REPERROR at the MAP level to override and supplement global error handling rules set with the REPERROR parameter at the root level of the parameter file. Multiple REPERROR statements can be applied to the same MAP statement to enable automatic, comprehensive management of errors and interruption-free replication processing.

For syntax and descriptions, see "REPERROR".

RESOLVECONFLICT (*conflict_resolution_specification*)

RESOLVECONFLICT is valid for MAP.

Use RESOLVECONFLICT in a bi-directional or multi-master configuration to specify how Replicat handles conflicts on operations made to the tables in the MAP statement.

Multiple resolutions can be specified for the same conflict type and are executed in the order listed in RESOLVECONFLICT. Multiple resolutions are limited to INSERTROWEXISTS and UPDATEROWEXISTS conflicts only.

RESOLVECONFLICT can be used multiple times in a MAP statement to specify different resolutions for different conflict types.

The following are the data types and platforms that are supported by RESOLVECONFLICT.

- RESOLVECONFLICT supports all databases that are supported by Oracle GoldenGate for Windows and UNIX.
- To use RESOLVECONFLICT, the database must reside on a Windows, Linux, or UNIX system (including those running on NonStop OSS).
- CDR supports data types that can be compared with simple SQL and without explicit conversion. See the individual parameter options for details.

- Do not use `RESOLVECONFLICT` for columns that contain LOBs, abstract data types (ADT), or user-defined types (UDT).

Syntax

```
RESOLVECONFLICT (
{INSERTROWEXISTS | UPDATEROWEXISTS | UPDATEROWMISSING |
   DELETEROWEXISTS | DELETEROWMISSING}
( {DEFAULT | resolution_name},
  {USEMAX (resolution_column) | USEMAXEQ (resolution_column) | USEMIN
(resolution_column) | USEMINEQ (resolution_column) | USEDELTA |
     DISCARD | OVERWRITE | IGNORE}
 )
[, COLS (column[,...])]
)
```

INSERTROWEXISTS | UPDATEROWEXISTS | UPDATEROWMISSING | DELETEROWEXISTS | DELETEROWMISSING
The type of conflict that this resolution handles.

INSERTROWEXISTS
An inserted row violates a uniqueness constraint on the target.

UPDATEROWEXISTS
An updated row exists on the target, but one or more columns have a before image in the trail that is different from the current value in the database.

UPDATEROWMISSING
An updated row does not exist in the target.

DELETEROWEXISTS
A deleted row exists in the target, but one or more columns have a before image in the trail that is different from the current value in the database.

DELETEROWMISSING
A deleted row does not exist in the target.

DEFAULT | *resolution_name*

DEFAULT
The default column group. The resolution that is associated with the `DEFAULT` column group is used for all columns that are not in an explicitly named column group. You must define a `DEFAULT` column group.

resolution_name
A name for a specific column group that is linked to a specific resolution type. Supply a name that identifies the resolution type. Valid values are alphanumeric characters. Avoid spaces and special characters, but underscores are permitted, for example:

```
delta_res_method
```

Use either a named resolution or `DEFAULT`, but not both.

USEMAX (*resolution_column*) | USEMAXEQ (*resolution_column*) | USEMIN (*resolution_column*) | USEMINEQ (*resolution_column*) | USEDELTA | DISCARD | OVERWRITE | IGNORE
The conflict-handler logic that is used to resolve the conflict. Valid resolutions are:

USEMAX
If the value of *resolution_column* in the trail record is greater than the value of the column in the database, the appropriate action is performed.

- (INSERTROWEXISTS conflict) Apply the trail record, but change the insert to an update to avoid a uniqueness violation, and overwrite the existing values.
- (UPDATEROWEXISTS conflict) Apply the trail record as an update.

USEMAXEQ
If the value of *resolution_column* in the trail record is greater than or equal to the value of the column in the database, the appropriate action is performed.

- (INSERTROWEXISTS conflict) Apply the trail record, but change the insert to an update to avoid a uniqueness violation, and overwrite the existing values.
- (UPDATEROWEXISTS conflict) Apply the trail record as an update.

USEMIN
If the value of *resolution_column* in the trail record is less than the value of the column in the database, the appropriate action is performed:

- (INSERTROWEXISTS conflict) Apply the trail record, but change the insert to an update to avoid a uniqueness violation, and overwrite the existing values.
- (UPDATEROWEXISTS conflict) Apply the update from the trail record.

USEMINEQ
If the value of *resolution_column* in the trail record is less than or equal to the value of the column in the database, the appropriate action is performed:

- (INSERTROWEXISTS conflict) Apply the trail record, but change the insert to an update to avoid a uniqueness violation, and overwrite the existing values.
- (UPDATEROWEXISTS conflict) Apply the update from the trail record.

resolution_column
The name of a NOT NULL column that serves as the resolution column. This column must be part of the column group that is associated with this resolution. The value of the resolution column compared to the current value in the target database determines how a resolution should be applied. The after image of the resolution column is used for the comparison, if available; otherwise the before image value is used. Use a column that can be compared through simple SQL:

- NUMERIC
- DATE
- TIMESTAMP
- CHAR/NCHAR
- VARCHAR/ NVARCHAR

To use a latest-timestamp resolution, use a timestamp column as the *resolution_column* and set the timestamp column to the current time when a row is inserted or updated. If possible, define the resolution column with the SYSTIMESTAMP data type, which supports fractional seconds. When comparisons are performed with sub-second granularity, there is little need for tie-breaking conflict handlers that resolve cases where the value of the resolution column is identical in both trail and target. If you ensure that the value of the

timestamp column can only increase or only decrease (depending on the resolution), then USEMAX and USEMIN does not lead to data divergence.

> **Note:**
>
> Do not use a primary key column as the resolution column in a USEMAX statement for the UPDATEROWEXISTS conflict. Otherwise, Replicat abends with an error similar to the following:
>
> ```
> 2013-04-04 10:18:38  ERROR   OGG-01922  Missing  RESOLUTION COLUMN NAME while
> mapping to target table "FIN"."ACCT".
> ```

USEDELTA
(UPDATEROWEXISTS conflict only) Add the difference between the before and after values in the trail record to the current value of the column in the target database. If any of the values is NULL, an error is raised. Base USEDELTA on columns that contain NUMERIC data types. USEDELTA is useful in a multi-node configuration when a row is getting simultaneously updated on multiple nodes. It propagates only the difference in the column values to the other nodes, so that all nodes become synchronized.

DISCARD
(Valid for all conflict types) Retain the current value in the target database, and write the data in the trail record to the discard file.
Use DISCARD with caution, because it can lead to data divergence.

OVERWRITE
(Valid for all conflict types except DELETEROWMISSING) Apply the trail record as follows:

- (INSERTROWEXISTS conflict) Apply the trail record but change the insert to an update to avoid a uniqueness violation, and overwrite the existing values.
- (UPDATEROWEXISTS conflict) Apply the update from the trail record.
- (UPDATEROWMISSING conflict) Apply the trail record but convert the missing UPDATE to an INSERT by using the modified columns from the after image and the unmodified columns from the before image. To convert an update to an insert, the before image of all columns of the row must be available in the trail. Use supplemental logging if the database does not log before images by default, and specify ALL for the Extract GETBEFORECOLS parameter.
- (DELETEROWEXISTS conflict) Apply the delete from the trail record, but use only the primary key columns in the WHERE clause.

Use OVERWRITE with caution, because it can lead to data divergence.

IGNORE
(Valid for all conflict types) Retain the current value in the target database, and ignore the trail record: Do not apply to the target table or a discard file.

COLS (*column*[, ...])
A non-default column group. This is a list of columns in the target database (after mapping) that are linked to, and operated upon by, a specific resolution type. If no column group is specified for a conflict, then all columns are affected by the resolution that is specified for the given conflict.

Alternatively, you can specify a DEFAULT column group, which includes all columns that are not listed in another column group. See the DEFAULT option.
You can specify multiple column groups, each with a different resolution. For example, you could use OVERWRITE for col2 and col3, and you could use USEDELTA for col4. No column in any group can be in any other group. Conflicts for columns in different column groups are resolved separately according to the specified resolution, and in the order listed.
Column groups work as follows:

- For INSERTROWEXISTS and UPDATEROWEXISTS conflicts, you can use different column groups to specify more than one of these conflict types and resolutions per table. Conflicts for columns in different column groups are resolved separately, according to the conflict resolution method specified for the column group.
- For UPDATEROWMISSING, DELETEROWEXISTS, and DELETEROWMISSING, you can use only one column group, and all columns of the table must be in this column group (considered the *default* column group).

Examples

Example 1
This example demonstrates all conflict types with USEMAX, OVERWRITE, DISCARD.

```
MAP fin.src, TARGET fin.tgt,
    COMPARECOLS (ON UPDATE ALL, ON DELETE ALL),
    RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (last_mod_time)),
    RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (last_mod_time)),
    RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)),
    RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)),
    RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD)),
    );
```

Example 2
This example demonstrates UPDATEROWEXISTS with USEDELTA and USEMAX.

```
MAP fin.src, TARGET fin.tgt,
    COMPARECOLS
    (ON UPDATE KEYINCLUDING (address, phone, salary, last_mod_time),
    ON DELETE KEYINCLUDING (address, phone, salary, last_mod_time)),
    RESOLVECONFLICT (
    UPDATEROWEXISTS,
    (delta_res_method, USEDELTA, COLS (salary)),
    (DEFAULT, USEMAX (last_mod_time)));
```

Example 3
This example demonstrates UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE.

```
MAP fin.src, TARGET fin.tgt,
    COMPARECOLS
    (ON UPDATE ALLEXCLUDING (comment)),
    RESOLVECONFLICT (
    UPDATEROWEXISTS,
    (delta_res_method, USEDELTA, COLS (salary, balance)),
    (max_res_method, USEMAX (last_mod_time), COLS (address, last_mod_time)),
    (DEFAULT, IGNORE));
```

SQLEXEC (*SQL_specification*)

SQLEXEC is valid for TABLE and MAP.

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

with the database to perform any work that is supported by the database. This work can be part of the synchronization process, such as retrieving values for column conversion, or it can be independent of extracting or replicating data, such as executing a stored procedure that executes an action within the database.

See "SQLEXEC" for syntax and usage.

SQLPREDICATE 'WHERE *where_clause*'

SQLPREDICATE is valid for TABLE.

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 faster selection method for initial loads than the WHERE or FILTER options. It affects the SQL statement directly and does not require Extract to fetch all records before filtering them.

For Oracle tables, 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.

SQLPREDICATE also enables you to select data based on a timestamp or other criteria to filter the rows that are extracted and loaded to the target table. SQLPREDICATE can be used for ORDER BY clauses or any other type of selection clause.

Make certain that the WHERE clause contains columns that are part of a key or index. Otherwise, Extract performs a full table scan, which reduces the efficiency of the SELECT statement.

SQLPREDICATE is valid for Oracle, DB2 LUW, DB2 on z/OS, DB2 for i, and SQL Server databases. Do not use SQLPREDICATE for an Extract group that is configured to synchronize transactional changes. It is only appropriate for an initial load Extract, because it re quires a SELECT statement that selects records directly from tables.

Syntax

```
TABLE source_table, SQLPREDICATE 'WHERE where_clause';
```

WHERE
This is a required keyword.

where_clause
A valid SQL WHERE clause that selects records from the source tables.

Example

```
TABLE hr.emp, SQLPREDICATE 'WHERE state = 'CO' and city = 'DENVER''
```

THREAD (*thread_ID*)

THREAD is valid for MAP. This option is valid when Replicat is in coordinated mode.

Use THREAD to specify that all of the object or objects in the same MAP statement are to be processed by the specified Replicat thread. The specified thread handles filtering,

manipulation, delivery to the target, error handling, and other work that is configured for those objects. Wildcards can be used in the TARGET clause when THREAD is used.

All tables that have referential dependencies among one another must be mapped in the same thread. For example, if tables scott.cust and scott.ord have a foreign-key relationship, the following is a possible mapping:

```
MAP scott.cust, TARGET scott.cust, THREAD (5);
MAP scott.ord, TARGET scott.ord, THREAD (5);
```

The thread with the lowest thread ID always processes barrier transactions if the THREAD or THREADRANGE option is omitted. Additionally, and work that is not explicitly assigned to a thread is processed through this thread. For example, if there are threads with IDs ranging from 1 to 10, barrier and non-assigned transactions are performed by thread 1.

To process a MAP statement among multiple threads, see *THREADRANGE (thread_range, column_list)*. THREAD and THREADRANGE are mutually exclusive options. Do not use them together in the same MAP statement.

Syntax

```
THREAD (thread_ID)
```

thread_ID
A numerical identifier for the thread that will process this MAP statement. Valid values are 1 through the value that was specified with the MAXTHREADS option of the ADD REPLICAT command that created this group. You can use the INFO REPLICAT command to verify the maximum number of threads allowed for a Replicat group. When specifying thread IDs, the following must be true:

- The *total number* of threads specified across all MAP statements of a Replicat group cannot exceed the value of MAXTHREADS.
- No single *thread_ID* value in the Replicat group can be higher than the value of MAXTHREADS. For example, if MAXTHREADS is 25, there cannot be a *thread_ID* of 26 or higher.

If MAXTHREADS was not used, the default maximum number of threads is 25.

Examples

The following examples show some ways to use the THREAD option.

Example 1
In this example, thread 1 processes table cust.

```
MAP scott.cust, TARGET scott.cust, THREAD (1);
```

Example 2
In this example, thread 1 processes all of the tables in the scott schema.

```
MAP scott.*, TARGET scott.*, THREAD (1);
```

Example 3
In this example, the `orders` table is partitioned among two `MAP` statements through the use of `FILTER (filter_clause)` and the `@RANGE` function. For more information about `@RANGE`, see "@RANGE".

```
MAP scott.orders, TARGET scott.orders, FILTER (@RANGE (1, 2, OID)), THREAD
(1);
MAP scott.orders, TARGET scott.orders, FILTER (@RANGE (2, 2, OID)), THREAD
(2);
```

`THREADRANGE (thread_range, column_list)`

`THREADRANGE` is valid for `MAP`. This option is valid when Replicat is in coordinated mode.

Use `THREADRANGE` to specify that the workload of the target table is to be partitioned evenly among a range of Replicat threads, based on the value of a specified column or columns. For example, if the partitioning is based on the value of a column named `ID`, and the `THREADRANGE` value is 1-3, then thread 1 processes rows with `ID` values from 1 through 10, thread 2 processes rows with `ID` values from 11 through 20, and thread 3 processes rows with `ID` values from 21 through 30. The partitioning may not be as absolutely even as shown in the preceding example, depending on the initial calculation of the workload, but it is coordinated so that same row is always processed by the same thread. Each specified thread handles filtering, manipulation, error handling, delivery to the target, and other work for its range of rows.

Partitioning a table across a range of threads may improve apply performance for very large tables or tables that frequently incur long-running transactions or heavy volume, but can be used in other cases, as well. You can process more than one table through the same range of threads.

A wildcarded `TARGET` clause can be used when `THREADRANGE` is used if the optional column list is omitted. When using a column list, use separate explicit `MAP` statements for each table that is using the same thread range.

To process a `MAP` statement with one specific thread, see `THREAD (thread_ID)`. `THREAD` and `THREADRANGE` are mutually exclusive options. Do not use them together in the same `MAP` statement.

Do not specify tables that have referential dependencies among one another in a thread range. Use the `THREAD` option and process all of those tables with the same thread.

Do not use `THREADRANGE` to partition sequences. If coordination is required, for example when a sequence is part of a `SQLEXEC` operation, partition the sequence work to one thread with the `THREAD` option.

The thread with the lowest thread ID always processes barrier transactions if the `THREAD` or `THREADRANGE` option is omitted. Additionally, and work that is not explicitly assigned to a thread is processed through this thread. For example, if there are threads with IDs ranging from 1 to 10, barrier and non-assigned transactions are performed by thread 1.

> **Note:**
>
> The columns specified in a list of columns must exist in the trail file. You can control this using KEYCOLS in the Extract to include this column, or by using FETCHCOLS in the Extract for the column, or by ensuring that the column is part of the supplemental log group and then using LOGALLSUPCOLS.

Syntax

```
THREADRANGE (lowID-highID, [column[, column][, ...]])
```

lowID
The lowest thread identifier of this range. Valid values are 1 through 500.

highID
The highest thread identifier of this range, which must be a higher number than *lowID*. Valid values are *lowID*+1 through 500. The number of threads in the range cannot exceed the value that was specified with the MAXTHREADS option of the ADD REPLICAT command. If MAXTHREADS was not used, the default maximum number of threads is 25.

[*column*[, *column*][, ...]]
Optional. Specifies one or more unique columns on which to base the row partitioning. To specify multiple columns, use a comma-delimited list, such as col1, col2, col3. When this option is omitted, the partitioning among the threads is based by default on the following columns, in the order of preference shown:

- Primary key
- KEYCOLS clause in the same MAP statement
- All of the columns of the table that are supported by Oracle GoldenGate for use as a key.

Example

The following example divides the orders and order_lines tables between the same two threads, based on the value of the OID column.

```
MAP scott.orders, TARGET scott.orders, THREADRANGE (1-2, OID);
MAP scott.order_lines, TARGET scott.order_lines, THREADRANGE (1-2, OID);
```

TOKENS (*token_definition*)

TOKENS is valid for TABLE.

Use TOKENS to define a user token and associate it with data. Tokens enable you to extract and store data within the user token area of a trail record header. Token data can be retrieved and used in many ways to customize the way that Oracle GoldenGate delivers data. For example, you can use token data in column maps, stored procedures called by SQLEXEC, or macros.

To use the defined token data in target tables, use the @TOKEN column-conversion function in the COLMAP clause of a Replicat MAP statement. The @TOKEN function maps the name of a token to a target column.

Do not use this option for tables being processed in pass-through mode by a data-pump Extract group.

The character set of token data is not converted. The token must be in the character set of the source database for Extract and in the character set of the target database for Replicat.

Syntax

```
TOKENS (token_name = token_data [, ...])
```

token_name
A name of your choice for the token. It can be any number of valid characters and is not case-sensitive. Multi-byte names are not supported.

token_data
Any valid character string of up to 2000 bytes. The data can be either a literal that is enclosed within single quotes (or double quotes if NOUSEANSISQLQUOTES is in use) or the result of an Oracle 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'));
```

TRIMSPACES | NOTRIMSPACES

TRIMSPACES and NOTRIMSPACES are valid for TABLE and MAP.

Use TRIMSPACES and NOTRIMSPACES at the root level of a parameter file or within a TABLE or MAP statement to control whether or not trailing spaces in a source CHAR column are truncated when applied to a target CHAR or VARCHAR column. The default is TRIMSPACES.

See "TRIMSPACES | NOTRIMSPACES" for syntax and usage.

TRIMVARSPACES | NOTRIMVARSPACES

TRIMVARSPACES and NOTRIMVARSPACES are valid for TABLE and MAP.

Use TRIMVARSPACES and NOTRIMVARSPACES at the root level of a parameter file or within a TABLE or MAP statement to control whether or not trailing spaces in a source VARCHAR column are truncated when applied to a target CHAR or VARCHAR column. The default is NOTRIMVARSPACES.

See "TRIMVARSPACES | NOTRIMVARSPACES" for syntax and usage.

WHERE (*clause*)

WHERE is valid for TABLE and MAP.

Use WHERE to select records based on a conditional statement. WHERE does not support the following:

- Columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.
- The evaluation of the before image of a primary key column in the conditional statement as part of a primary key update operation.

Enclose literals in single quotes. Specify case-sensitive column names as they are stored in the database, and enclose them in double quotes if the database requires quotes to enforce case-sensitivity (such as Oracle).

Getting More Information about Record Filtering

Syntax

```
WHERE (clause)
```

clause
Selects records based on a condition, such as:

```
WHERE (branch = 'NY')
```

Table 2-12 shows permissible `WHERE` operators.

Table 2-12 Permissible WHERE Operators

Operator	Example
Column names	`PRODUCT_AMT` `"Product_Amt"`
Numeric values	`-123, 5500.123`
Literal strings enclosed in single quotes	`'AUTO', 'Ca'`
Column tests	`@NULL`, `@PRESENT`, `@ABSENT` (column is null, present or absent in the record). These tests are built into Oracle GoldenGate.
Comparison operators	`=, <>, >, <, >=, <=`
Conjunctive operators	`AND, OR`
Grouping parentheses	Use open and close parentheses for logical grouping of multiple elements.

Example

The following `WHERE` example returns all records when the `AMOUNT` column is over 10,000 and does not cause a record to be discarded when `AMOUNT` is absent.

```
WHERE (amount = @PRESENT AND amount > 10000)
```

`PARTITIONOBJID`

Valid for Integrated Extract.

`PARTITIONOBJID` is used to specify the object IDs of the partitions to be captured for partitioned tables, and applies to Integrated Extract. For an IO table (with or without overflow area), index segment object ID should be used for partition level filtering. In this case, `PARTITIONOBJID` in the `MAP` or `TABLE` statement specifies the index segment object IDs of the partitions to be extracted.

Syntax

```
MAP/TABLE [container.]schema.table PARTITIONOBJID ptn_object_ID [, ptn_object_ID]
```

The following restrictions apply:

- Wildcarded table names are not allowed for a MAP/TABLE parameter that contains PARTITIONOBJID.
- DDL Capture and replication is not supported when using PARTITIONOBJID.

Syntax for IO table TABLE statement:

```
TABLE [container.]schema.table PARTITIONOBJID index_segment_object_ID [,
index_segment_object_ID]
```

Syntax for IO table MAP statement:

```
MAP [container.]schema.table PARTITIONOBJID index_segment_object_ID [,
index_segment_object_ID]
```

TABLE for DEFGEN

Valid For

DEFGEN

Description

Use the TABLE parameter in a DEFGEN parameter file to identify a source table or tables for which you want to run the utility.

You can output definitions for objects that are in different containers in an Oracle container database to the same definitions file. All table attributes must be identical, such as case sensitivity, character set, and the use of the full three-part name. For example, you cannot use two-part names (stripped of their container or catalog by the NOCATALOG parameter) and three-part names in the same definitions file.

Default

None

Syntax

```
TABLE [catalog.]owner.table[, DEF template];
```

[*catalog*.]*owner.table*
The Oracle container database if applicable, and the owner and name of the table. This parameter accepts wildcards. Oracle GoldenGate automatically increases the internal storage to track up to 100,000 wildcard entries.
Oracle GoldenGate preserves the case of the table name. Some databases require a name to be within double quotes to enforce case-sensitivity. Other case-sensitive databases do not require double quotes to enforce case-sensitivity, but the names must be specified the way they are stored in the database.

DEF *template*
Creates a definitions template based on the definitions of the specified table. A template enables new tables that have the same definitions as the specified table to be added during an Oracle GoldenGate process run, without the need to run DEFGEN for them first, and without the need to stop and start the Oracle GoldenGate process to update its definitions cache. To use a template that is generated by DEFGEN, specify it with the DEF or TARGETDEF option of the TABLE or MAP statement. To retain case-sensitivity, specify the template name the way you would specify any case-sensitive object in the database. This option is not supported for initial loads.

;
Terminates the TABLE statement.

Examples

Example 1

```
TABLE fin.account;
```

Example 2

```
TABLE fin.acc*;
```

Example 3

```
TABLE fin."acct1", DEF "acctdefs";
```

TABLE for Replicat

Valid For

Replicat

Description

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.

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. If Replicat is in coordinated mode, all actions are processed through the thread with the lowest thread ID.

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 so that Replicat can 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.

Syntax

See "TABLE | MAP" for descriptions of the following syntax options.

```
TABLE table_spec,
[, SQLEXEC (SQL_specification), BEFOREFILTER]
[, FILTER (filter_clause)]
[, WHERE (where_clause)]
{, EVENTACTIONS ({IGNORE | DISCARD} [action])}
;
```

Example

The following example enables Replicat tracing for an order transaction that contains an insert operation for a specific order number (`order_no = 1`). The trace information is written to the `order_1.trc` trace file. The `MAP` parameter specifies the mapping of the source table to the target table.

```
MAP sales.order, TARGET rpt.order;
TABLE sales.order,
FILTER (@GETENV ('GGHEADER', 'OPTYPE') = 'INSERT' AND @STREQ (order_no, 1), &
EVENTACTIONS (TRACE order_1.trc TRANSACTION);
```

TABLEEXCLUDE

Valid For

Extract

Description

Use the `TABLEEXCLUDE` parameter with the `TABLE` and `SEQUENCE` parameters to explicitly exclude tables and sequences from a wildcard specification. The positioning of `TABLEEXCLUDE` in relation to parameters that specify files or trails determines its effect. Parameters that specify trails or files are: `EXTFILE`, `RMTFILE`, `EXTTRAIL`, `RMTTRAIL`. The parameter works as follows:

- When a `TABLEEXCLUDE` specification is placed before any `TABLE` or `SEQUENCE` parameters, and also before the parameters that specify trails or files, it applies globally to all trails or files, and to all `TABLE` and `SEQUENCE` parameters.
- When a `TABLEEXCLUDE` specification is placed after a parameter that specifies a trail or file, it is effective only for that trail or file and only for the `TABLE` or `SEQUENCE` parameters that are associated with it. Multiple trail or file specifications can be made in a parameter file, each followed by a set of `TABLE`, `SEQUENCE`, and `TABLEEXCLUDE` specifications.

`TABLEEXCLUDE` is evaluated before evaluating the associated `TABLE` or `SEQUENCE` parameter. Thus, the order in which they appear does not make a difference.

When using wildcards, be careful not to place them such that all objects are excluded, leaving nothing to capture. For example, the following captures nothing:

```
TABLE cat1.schema*.tab*;
TABLEEXCLUDE cat1.*.*
```

The default for resolving wildcards is `WILDCARDRESOLVE DYNAMIC`. Therefore, if a table that is excluded with `TABLEEXCLUDE` is renamed to a name that satisfies a wildcard, the data will be captured. The `DYNAMIC` setting enables new table names that satisfy a wildcard to be resolved as soon as they are encountered and included in the Oracle GoldenGate configuration immediately. For more information, see WILDCARDRESOLVE.

See also the EXCLUDEWILDCARDOBJECTSONLY parameter.

Default

None

Syntax

```
TABLEEXCLUDE [container. | catalog.]owner.{table | sequence}
```

container.
If the database requires three-part names, specifies the name or wildcard specification of the Oracle container that contains the object to exclude.

owner
Specifies the name or wildcard specification of the owner, such as the schema, of the object to exclude.

table | sequence
The name or wildcard specification of the object to exclude.

Example

In this example, `test.tab*` specifies that all tables beginning with `tab` in schema `test` are to be excluded from all trail files. Table `fin.acct` is excluded from trail `ee`. Table `fin.sales` is excluded from trail `ff`.

```
TABLEEXCLUDE  test.tab*
    EXTTRAIL ./dirdat/ee
TABLE pdb1.*.*;
TABLEEXCLUDE pdb1.fin.acct
    EXTTRAIL ./dirdat/ff
TABLE pdb2.*.*;
TABLEEXCLUDE pdb2.fin.sales
```

TARGETDDLNOTIFY | NOTARGETDDLNOTIFY

Valid For

Replicat

Description

This parameter controls whether or not Replicat uses DDL notification to synchronize its target table metadata cache. If it's not supported by the database, then the parameter is ignored. It is supported by all types of Replicat (classic, coordinated, integrated, parallel and parallel integrated). It applies to all tables mapped by the Replicat.

It is ON by default if the target Oracle database supports it.

`NOTARGETDDLNOTIFY` turns it OFF.

Default

`ON`

Syntax

```
TARGETDDLNOTIFY
```

Examples

For Oracle database 23ai and Oracle GoldenGate 23ai Replicat, the target table DDL notification is enabled by default. The following example shows how to turn off target table DDL notification for Replicat in the Replicat parameter file:

```
NOTARGETDDLNOTIFY
```

TARGETDEFS

Valid For

Primary Extract

Description

Use the `TARGETDEFS` parameter to specify a target-definitions 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.

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 specific application.

To generate the target-definitions file, use the `DEFGEN` utility. Transfer the file to the source or intermediary system before starting Extract.

Default

None

Syntax

```
TARGETDEFS file
```

file
The relative or fully qualified path name of the target-definitions file.

Examples

Example 1

```
TARGETDEFS C:\repodbc\sales.def
```

Example 2

```
TARGETDEFS /ggs/dirdef/ODBC/tandem_defs
```

TCPSOURCETIMER | NOTCPSOURCETIMER

Valid For

Extract

Description

Use the TCPSOURCETIMER and NOTCPSOURCETIMER parameters to manage the timestamps of replicated operations for reporting purposes within the Oracle GoldenGate environment.

TCPSOURCETIMER and NOTCPSOURCETIMER are global parameters and apply to all TABLE statements in the Extract parameter file.

Default

```
TCPSOURCETIMER
```

Syntax

```
TCPSOURCETIMER | NOTCPSOURCETIMER
```

TCPSOURCETIMER
Adjusts the timestamp of data records when they are sent to other systems, making it easier to interpret synchronization lag. This is the default.

NOTCPSOURCETIMER
Retains the original timestamp value. Use NOTCPSOURCETIMER when using timestamp-based conflict resolution in a bidirectional configuration and when using a user token that refers to 'GGHEADER', 'COMMITTIMESTAMP' of the @GETENV column-conversion function.

TRACE | TRACE2

Valid For

Extract and Replicat

Description

Use the TRACE and TRACE2 parameters to capture Extract or Replicat processing information to help reveal processing bottlenecks. 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. .

Contact Oracle Support for assistance if the trace reveals significant processing bottlenecks.

Default

No tracing

Syntax

```
TRACE | TRACE2
[, DDL[INCLUDE] | DDLONLY]
[, [FILE] file_name]
[, THREADS (threadID[, threadID][, ...][, thread_range[, thread_range][, ...])]
```

TRACE
Provides step-by-step processing information.

TRACE2
Identifies the code segments on which Extract or Replicat is spending the most time.

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. This is the default. Either `DDL` or `DDLINCLUDE` is valid.

DDLONLY
Traces DDL but does not trace transactional data.

[FILE] *file_name*
The relative or fully qualified name of a file to which Oracle GoldenGate logs the trace information. The `FILE` keyword is optional, but must be used if other parameter options will follow the file name, for example:

```
TRACE FILE file_name DDLINCLUDE
```

If no other options will follow the file name, the `FILE` keyword can be omitted, for example:

```
TRACE DDLINCLUDE file_name
```

THREADS (*threadID*[, *threadID*][, ...][, *thread_range*[, *thread_range*][, ...])
Enables tracing only for the specified thread or threads of a coordinated Replicat. Tracing is only performed for threads that are active at runtime.

***threadID*[, *threadID*][, ...]**
Specifies a thread ID or a comma-delimited list of threads in the format of `threadID, threadID, threadID`.

[, *thread_range*[, *thread_range*][, ...]
Specifies a range of threads in the form of `threadIDlow-threadIDhigh` or a comma-delimited list of ranges in the format of `threadIDlow-threadIDhigh, threadIDlow-threadIDhigh`.

A combination of these formats is permitted, such as `threadID, threadID, threadIDlow-threadIDhigh`.
If the Replicat is in coordinated mode and `TRACE` is used with a `THREADS` list or range, a trace file is created for each currently active thread. Each file name is appended with its associated thread ID. This method of identifying trace files by thread ID does not apply when `SEND REPLICAT` is issued by `groupname` with `threadID` (as in `SEND REPLICAT fin003 TRACE...`) or when only one thread is specified with `THREADS`.
Contact Oracle Support for assistance if the trace reveals significant processing bottlenecks.

Examples

Example 1
The following traces to a file named `trace.trc`. If this is a coordinated Replicat group, the tracing applies to all active threads.

```
TRACE /home/ggs/dirrpt/trace.trc
```

Example 2
The following enables tracing for only thread 1. In this case, because only one thread is being traced, the trace file will not have a *threadID* extension. The file name is `trace.trc`.

```
TRACE THREADS(1) FILE ./dirrpt/trace.trc
```

Example 3
The following enables tracing for threads 1,2, and 3. Assuming all threads are active, the tracing produces files `trace001`, `trace002`, and `trace003`.

```
TRACE THREADS(1-3) FILE ./dirrpt/trace.trc
```

TRAILBYTEORDER

Valid For

GLOBALS, Extract

Description

> **Note:**
>
> `TRAILBYTEORDER` is automatically handled by Oracle GoldenGate, and should only be used if the default settings are not working.

Use the `TRAILBYTEORDER` parameter in the `GLOBALS` file to set the byte format of the metadata in the trails or files created with the `EXTFILE`, `EXTTRAIL`, `RMTFILE`, and `RMTTRAIL` parameters. By default, Extract always writes the trail metadata in big endian byte order, regardless of the byte order of the source or target machine.

This parameter affects only the metadata of the trail records. It does not affect the column data.

When used in the `GLOBALS` file, `TRAILBYTEORDER` affects all of the files or trails in the same Oracle GoldenGate instance. To specify the byte order of a specific trail or file, use the `TRAILBYTEORDER` option of the associated `EXTFILE`, `RMTFILE`, `EXTTRAIL`, or `RMTTRAIL` parameter in the Extract parameter file. In cases where Extract writes to multiple trails or files on different platforms, `TRAILBYTEORDER` in the Extract parameter file enables the correct byte ordering of each one. When `TRAILBYTEORDER` is used as an Extract parameter, it overrides any `TRAILBYTEORDER` specification in the `GLOBALS` file.

`TRAILBYTEORDER` reduces the overhead of conversion work when the source and target machines both use little endian. In this case, because the default without `TRAILBYTEORDER` is `BIGENDIAN`, the conversion work must be performed from little endian to big endian (to write to trail) and then from big endian to little endian to read the trail on the target. `TRAILBYTEORDER` prevents unnecessary conversions by allowing you to specify the byte order that is used by both the source and target machines (`LITTLEENDIAN`) as the byte order of the trail.

In the case where the source byte order is big endian and the target is little endian, where some conversion is required, you can decide whether the conversion takes place at the source or at the target. To perform the conversion on the source, set `TRAILBYTEORDER` to `LITTLEENDIAN`. The trail is converted to little endian, and no conversion is needed on the target. To perform the conversion on the target, leave the default set to `BIGENDIAN`. If the target system of the trail is big endian, `TRAILBYTEORDER` is not needed, because the default is big endian.

Use the `NATIVEENDIAN` option for a primary Extract or a data pump if the byte order of the source machine is not known, but you want to keep that format and do not want conversion performed on the source. If nothing is specified with `TRAILBYTEORDER`, a data pump writes the trail using the same byte order as the input trail, which may not be the desired format.

`TRAILBYTEORDER` is valid for files that have a `FORMAT RELEASE` version of at least 12.1. For older versions, this parameter is ignored.

Do not use TRAILBYTEORDER when replicating data to a NonStop system. On the NonStop platform, Oracle GoldenGate only supports LITTLEENDIAN, the default.

To identify the byte order of the metadata in a trail, use the ENV command of the Logdump utility.

Default

BIGENDIAN

Syntax

```
TRAILBYTEORDER {BIGENDIAN | LITTLEENDIAN | NATIVEENDIAN}
```

BIGENDIAN
Formats the trail metadata in big endian.

LITTLEENDIAN
Formats the trail metadata in little endian.

NATIVEENDIAN
Formats the trail metadata in the default byte order of the local system. Enables you to make certain the output trail is converted to the native format of the source machine.

Example

```
TRAILBYTEORDER LITTLEENDIAN
```

TRAILCHARSET

Valid For

Replicat

Description

> **Note:**
>
> This parameter has been replaced by the SOURCECHARSET parameter but may still be retained in existing parameter files for backward compatibility.

Use the TRAILCHARSET parameter to supply a character set for the source data if the trail is written by an Extract version that is earlier than 11.2.1.0.0. In the earlier versions, the source character set is not stored in the trail.

When TRAILCHARSET is used, Replicat uses the specified character set as the source character set when converting character-type columns to the target character set. Replicat issues a warning message when it uses the TRAILCHARSET character set.

By default, Replicat performs character set conversion. This feature is controlled by the CHARSETCONVERSION (default) and NOCHARSETCONVERSION parameters. To use TRAILCHARSET, NOCHARSETCONVERSION cannot be used.

Default

Character set of the operating system

Syntax

```
TRAILCHARSET source_charset [, REPLACEBADCHAR];
```

source_charset
The ICU character-set identifier or an Oracle character-set identifier of the source database. For Oracle databases, Oracle GoldenGate converts an Oracle identifier to the corresponding ICU identifier for conversion to the character set that is specified with the `NLS_LANG` specification in the `SETENV` parameter in the Replicat parameter file.

`REPLACEBADCHAR`
Prevents Replicat from abending when a conversion attempt fails. The failed character is replaced with a replacement character for each target character set. The replacement character is pre-defined in each character set.

Examples

Example 1

```
TRAILCHARSET ISO-8859-9;
```

Example 2

```
TRAILCHARSET windows-932, REPLACEBADCHAR;
```

Example 3

```
TRAILCAHRSET EUC-CN;
```

TRAILCHARSETASCII

Valid For

Extract for DB2 on z/OS; not valid for Extract or Replicat.

Description

Use `TRAILCHARSETASCII` to cause character data to be written to the trail file in the local ASCII code page of the DB2 subsystem from which data is to be captured.

- Specification of this parameter on a single-byte DB2 z/OS subsystem causes character data from non-Unicode tables to be written to the trail file in the installed ASCII single-byte CCSID. Data from EBCDIC tables is converted to this ASCII CCSID.
- Specification of this parameter on a multi-byte DB2 z/OS subsystem causes Extract to process only ASCII and Unicode tables. Extract abends with an error if it encounters EBCDIC tables. Data from ASCII tables is written to the trail file in the installed ASCII mixed CCSID.

Either `TRAILCHARSETASCII` or `TRAILCHARSETEBCDIC` is required if the target is a multi-byte system. To replicate both ASCII and EBCDIC tables to a multi-byte DB2 z/OS target, process each character set with an Extract process for the EBCDIC tables.

Default

Character data is written in the character set of the host table.

Syntax

```
TRAILCHARSETASCII
```

TRAILCHARSETEBCDIC

Valid For

Extract for DB2 on z/OS; not valid for Extract or Replicat.

Description

Use `TRAILCHARSETEBCDIC` to cause character data to be written to the trail file in the local EBCDIC code page of the DB2 subsystem from which data is to be captured.

- Specification of this parameter causes all character data to be written to the trail file in the EBCDIC code page of the job in which Extract is running.
- Specification of this parameter on a single-byte DB2 z/OS subsystem causes character data from non-Unicode tables to be written to the trail file in the installed EBCDIC single-byte CCSID. Data from ASCII tables is converted to this EBCDIC CCSID.
- Specification of this parameter on a multi-byte DB2 z/OS subsystem causes Extract to process only EBCDIC and Unicode tables. Extract abends with an error if it encounters ASCII tables. Data from EBCDIC tables is written to the trail file in the installed EBCDIC mixed CCSID.

Either `TRAILCHARSETASCII` or `TRAILCHARSETEBCDIC` is required if the target is a multi-byte system. To replicate both ASCII and EBCDIC tables to a multi-byte DB2 z/OS target, process each character set with an Extract process for the EBCDIC tables.

Default

Character data is written in the character set of the host table.

Syntax

```
TRAILCHARSETEBCDIC
```

TRANLOGOPTIONS

Valid For

Extract

Description

Use the `TRANLOGOPTIONS` parameter to control the way that Extract interacts with the transaction log or with the API that passes transaction data, depending on the database or capture mode. You can use multiple `TRANLOGOPTIONS` statements in the same parameter file, or you can specify multiple options within the same `TRANLOGOPTIONS` statement, if permissible for those options.

Use a given `TRANLOGOPTIONS` option only for the database or databases for which it is intended.

Default

None

Syntax

```
TRANLOGOPTIONS {
[ALLOWTABLECOMPRESSION][ALTLOGDEST path | REMOTE]
[ALWAYSONREADONLYROUTING]
[{DBLOGREADERBUFSIZE size}][ASYNCTRANSPROCESSING buffer_size]
[BUFSIZE size]
[CHECKPOINTRETENTIONTIME days][DB2APIRETRY     retry_count]
[DB2ZV11COMPATIBILITYMODE][DICTIONARY_CACHE_SIZE value]
[DLFAILOVER_TIMEOUT seconds]
[DISABLESOFTEOFDELAY]
[EXCLUDEFILTERTABLE table]
[EXCLUDETAG [tag | NULL] | [EXCLUDETAG +]
[EXCLUDETRANS transaction]
[EXCLUDEUSER user]
[EXCLUDEUSERID Oracle_uid]
[FAILOVERTARGETDESTID n][FETCHPARTIALJSON][FETCHPARTIALLOB][FETCHPARTIALXML]
[FORCEFETCHLOB]
[GETCTASDML | NOGETCTASDML][HANDLEDLFAILOVER [STANDBY_WARNING value |
STANDBY_ABEND value]]
[IFILOCKSECONDS (seconds)]
[IGNOREDATACAPTURECHANGES | NOIGNOREDATACAPTURECHANGES][INCLUDEAUX
(AUX_specification)]
[INCLUDEREGIONID | INCLUDEREGIONIDWITHOFFSET]
[INCLUDETAG tag]
[ENABLE_PROCEDURAL_REPLICATION Y]
[ENABLE_AUTO_CAPTURE | DISABLE_AUTO_CAPTURE]
[LOB_CHUNK_SIZE size][MAXAUTOCMTTRANSSIZE (range, default)][MININGUSER {/ |
user}[, MININGPASSWORD password]
   [algorithm ENCRYPTKEY {key_name | DEFAULT}] [SYSDBA]
[MININGUSERALIAS alias [DOMAIN domain]]
[MIXEDENDIAN [ON|OFF]]
[MANAGECDCCLEANUP | NOMANAGECDCCLEANUP ]
[MANAGESECONDARYTRUNCATIONPOINT | NOMANAGESECONDARYTRUNCATIONPOINT ]
[PERFORMANCEPROFILE HIGH|MEDIUM|LOW_RES][QUERYTIMEOUT seconds]
[QUERYRETRYCOUNT seconds]
[READQUEUESIZE size]
[READTIMEOUT milliseconds]
[REDO_TRANSPORT_LAG_THRESHOLD seconds]
[REDO_TRANSPORT_LAG_TIMEOUT value]
[REQUIRELONGDATACAPTURECHANGES | NOREQUIRELONGDATACAPTURECHANGES]
[SOURCE_OS_TIMEZONE timezone]
[SKIPUNKNOWNEVENT]
[SUPPRESSNOOOPUPDATES][TRACKSCHEMACHANGES ]
[TRANCOUNT integer]
[TSLOOKUPBEGINLRI | TSLOOKUPENDLRI]
[VALIDATEINLINESFLOB]
[USE_ROOT_CONTAINER_TIMEZONE]
[USENATIVEOBJSUPPORT | NOUSENATIVEOBJSUPPORT]
[VERSIONCHECK DYNAMIC | IMMEDIATE]
}
```

ALWAYSONREADONLYROUTING

Valid for SQL Server

The ALWAYSONREADONLYROUTING parameter allows Extract for SQL Server to route its read-only processing to an available read-intent Secondary when connected to an Always On availability group listener.

ALTLOGDEST *path* | *REMOTE*
Valid for MySQL.
Specifies the location of the MySQL log index file. Extract looks for the log files in this location instead of the database default location. ALTLOGDEST can be used when the database configuration does not include the full path name to the logs or when there are multiple MySQL installations on the machine. Extract reads the log index file to find the binary log file that it needs to read. When ALTLOGDEST is used, Extract assumes that the logs and the index are in the same location.
Supply the full path name to the directory.
On Windows, enclose the path within double quotes if the path contains any spaces, such as in the following example.

```
TRANLOGOPTIONS ALTLOGDEST "C:\Program Files\MySQL\MySQL Server
5.7\log\binlog.index"
```

On Linux system:

```
TRANLOGOPTIONS ALTLOGDEST "/mnt/rdbms/mysql/data/logs/binlog.index"
```

When capturing against a remote MySQL database, use the REMOTE option instead of the index file path. From remote capture, specify the following in the Extract parameter file.

```
TRANLOGOPTIONS ALTLOGDEST REMOTE
```

For more information on using the REMOTE option, see Setting Logging Parameters.

ASYNCTRANSPROCESSING *buffer_size*
Valid for Extract in integrated capture mode for Oracle.
Controls whether integrated capture runs in asynchronous or synchronous processing mode, and controls the buffer size when Extract is in asynchronous mode. The minimum is 1 and the maximum is 1024; the default is 300.

ASYNCTRANSPROCESSING *buffer_size*
In asynchronous transaction processing mode, there are two threads of control:

- One thread groups logical change records (LCR) into transactions, does object-level filtering, and does partial rollback processing,
- The other thread formats committed transactions, performs any user-specified transformations, and writes to the trail file.

The transaction buffer is the buffer between these two threads and is used to transfer work from one thread to the other. The default transaction buffer size is 300 committed transactions, but is adjusted downward by the Oracle GoldenGate memory manager if its cache memory is close to being exhausted.

NOASYNCTRANSPROCESSING
Disables asynchronous processing and causes Extract to operate in synchronous mode. In this mode, one thread performs all capture work.

BUFSIZE *size*
Valid for DB2 LUW, and DB2 z/OS. Valid for DB2 for i from Oracle GoldenGate 19c and higher. Valid for Oracle database from Oracle GoldenGate 21c and higher.
Controls the maximum size, in bytes, of the buffers that are allocated to contain the data that is read from the transaction log.
High values increase capture 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.
For Oracle database, the DDL operation record size is limited by Oracle GoldenGate internal record capture buffer size. The DDL size can be up to the Oracle RDBMS size limit. Although Oracle database 21c allows creating DDL greater than 10MB, the maximum internal record capture buffer size is limited to 10MB.
The default buffer size is determined by the source of the redo data. The following are the valid ranges and default sizes, in bytes:
DB2 LUW:

- Minimum: 8,192
- Maximum: 10,000,000
- Default: 204,800
- 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.

DB2 z/OS and DB2 for i:

- Minimum: 36KB (36864)
- Maximum: 32MB (33554432)
- Default: 2MB (2097152)
- 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.
- Each Extract uses a fixed 32bytes of ECSA on the DB2 z/OS system that the Extract connects to. This doesn't apply to DB2 for i.

CHECKPOINTRETENTIONTIME *days*
Valid for Extract in integrated mode only for Oracle.
Controls the number of days that Extract retains checkpoints before they are purged. Partial days can be specified using decimal values. For example, 8.25 specifies 8 days and 6 hours. When the checkpoint of an Extract in integrated capture mode is purged, LogMiner data dictionary information for the archived redo log file that corresponds to the checkpoint is purged, and the `first_scn` value of the capture process is reset to the SCN value corresponding to the first change in the next archived redo log file. The default is seven days and the minimum is 0.00001.

DB2APIRETRY *number of retries*
If Extract receives an error from the DB2 log reading API `db2ReadLog()`, then for certain errors the API call is retried. Use the `DB2APIRETRY` to change the number of retries. The default number of retries is set to 3. SQL code for which the API is retried is `SQLCODE -30108`.

DB2ZV11COMPATIBILITYMODE
Valid for Extract for DB2 z/OS.
When using Oracle GoldenGate to extract from DB2 z/OS version 11 in some compatibility modes, the Extract process may not programmatically determine the actual database version and an OGG-00551 or OGG-00804 error occurs. Use this option in your Extract parameter file to manually set the correct database version.

DICTIONARY_CACHE_SIZE *value*
Use this option to tune dictionary cache size from Extract. The default value is 5000. If PERFORMANCEPROFILE is set to HIGH, then the default value is 10000.

DLFAILOVER_TIMEOUT *seconds*
Valid for Extract in integrated mode for Oracle.
Provides a configurable timeout in seconds to allow for standby database reinstatement post-role transition. It is used in conjunction with HANDLEDLFAILOVER to allow Integrated Extract to start up immediately after a role transition. At the end of the timeout period, if the standby database is still not available, then Extract will terminate.
The default is 300 seconds. You can also use centiseconds or milliseconds.

DISABLESOFTEOFDELAY
Valid for Extract only in integrated mode for Oracle and DB2 LUW.
Use DISABLESOFTEOFDELAY in the Extract parameter file to set that the wait time takes effect when the an EOF status is reported with no records to return.

[EXCLUDETAG [*tag* | NULL] | [EXCLUDETAG +]
Use EXCLUDETAG *tag* to direct the Extract process to ignore the individual records that are tagged with the specified redo tag. Compare with older versions, new trail file contains tag tokens, which would not introduce problems for older trail readers.
Use EXCLUDETAG + to direct the Extract process to ignore the individual records that are tagged with any redo tag.
The EXCLUDETAG is used to exclude changes that were earlier tagged either by Replicat using the DBOPTIONS SET TAG option or within the Oracle database session using the dbms_xstream.set_tag procedure.

Example
The following are examples of how to use tag specifiers with EXCLUDETAG.
To exclude all tagged changes:

```
TRANLOGOPTIONS EXCLUDETAG +
```

To exclude specific tagged changes:

```
TRANLOGOPTIONS EXCLUDETAG 00
TRANLOGOPTIONS EXCLUDETAG 0952
```

Considerations while using EXCLUDETAG and INCLUDETAG Parameters
While using EXCLUDETAG and INCLUDETAG parameters with TRANLOGOPTIONS and DDLOPTIONS commands, consider the following:

- If the TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG are specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are not specified, then the TRANLOGOPIIONS EXCLUDETAG/INCLUDETAG parameters apply to both DML and DDL operations.
- If the TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG options are specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are also specified, then the TRANLOGOPIIONS EXCLUDETAG/INCLUDETAG apply to DML operations, and the DDLOPTIONS EXCLUDETAG/INCLUDETAG apply to DDL operations.
- If the TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG are not specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are specified, then the DDLOPTIONS EXCLUDETAG/INCLUDETAG applies to DDL operations, and there is no tag filtering for DML operations.

- If TRANLOGOPTIONS EXCLUDETAG/INCLUDETAG are not specified and DDLOPTIONS EXCLUDETAG/INCLUDETAG are also not specified, then the default option DDLOPTIONS EXCLUDETAG + is applicable, which excludes all tagged DDL operations.
- For DDLOPTIONS when specifying both EXCLUDETAG and INCLUDETAG, then EXCLUDETAG should come first.

EXCLUDETRANS *transaction*
Valid for Integrated Extract for Oracle.
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.
For more information about bidirectional synchronization, see Configuring Bi-Directional Replication in *Oracle GoldenGate Microservices Documentation*.

EXCLUDEUSER *user*
Valid for DB2 LUW, DB2 for z/OS, DB2 for i, and Oracle.
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.
You can use EXCLUDEUSER and EXCLUDEUSERID in the same parameter file. Do not use wildcards in either parameter.
The user name must be valid. Oracle GoldenGate queries the database to get the associated user ID and maps the numeric identifier back to the user name. For this reason, if the specified user is dropped and recreated while name resolution is set to the default of DYNAMICRESOLUTION, EXCLUDEUSER remains valid. If the same transaction 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, see DYNAMICRESOLUTION .

- **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 must include the IGNOREAPPLOPS parameter for EXCLUDEUSER to operate correctly unlike all other supported databases. EXLCUDEUSER is not supported for multitenant source databases.

Example
The following Oracle example filters for two users (one by name and one by user ID). The transactions generated by these users will be handled according to the GETREPLICATES or IGNOREREPLICATES rules, and a new transaction buffer size is specified.

```
TRANLOGOPTIONS EXCLUDEUSER ggsrep, EXCLUDEUSERID 90, BUFSIZE 100000
```

To use EXCLUDEUSER with multitenant, you must specify the PDB.USERNAME. The following example excludes any DML operation made by PDBXYZ.SCOTT:

```
TRANLOGOPTIONS EXCLUDEUSER PDBXYZ.SCOTT
```

EXCLUDEUSERID *Database_uid*
Valid for Extract for Oracle.
Specifies the database 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. This parameter is not valid for multitenant Extracts. Use tagging and EXCLUDETAG instead.
Usage is the same as that of EXCLUDEUSER.
Oracle_uid is a non-negative integer with a maximum value of 2147483638. There are several system views that can be queried to get the user ID. The simplest one is the ALL_USERS view. Oracle GoldenGate does not validate the user ID. If the user that is associated with the specified user ID is dropped and recreated, a new user ID is assigned; therefore, EXCLUDEUSERID becomes invalid for that user.

FAILOVERTARGETDESTID *n*
Valid for Extract for Oracle.
When using Oracle GoldenGate Extract processes in an Oracle Data Guard configuration, the GoldenGate Extract process must remain behind the redo that has been applied to the Oracle Data Guard standby database. The FAILOVERTARGETDESTID parameter is used to identify the LOG_ARCHIVE_DEST_n initialization parameter which points to the standby, that is the failover target which Extract must remain behind. This parameter is used in combination with HANDLEDLFAILOVER to control whether Extract will throttle its writing of trail data based on the apply progress of the Oracle Data Guard standby database. Note that the FAILOVERTARGETDESTID is not needed if the Data Guard configuration has Fast Start Failover (FSFO) enabled. The minimum value is 0, the maximum is 32 and the default 0.

Example
To determine the correct value for the TRANLOGOPTIONS FAILOVERTARGETDESTID Extract parameter, connect to the database from which Extract is extracting data from, and issue the following command.

```
SQL> show parameters log_archive_dest
        NAME TYPE VALUE

------------------------------------------------------------------------------
         log_archive_dest_1 string location=USE_DB_RECOVERY_FILE_DEST,
         valid_for=(ALL_LOGFILES, ALL_ROLES)

         log_archive_dest_2 string service="ggs2d",
         ASYNC NOAFFIRM delay=0 optional compression =disable max_failure=0
         max_connections=1 reopen=300 db_unique_name="GGS2D" net_timeout=30,
         valid_for=(online_logfile,all_roles)
```

The Extract parameter TRANLOGOPTIONS FAILOVERTARGETDESTID will be set to 2 because that is the Standby database Oracle GoldenGate should stay behind. The first entry (log_archive_dest_1) is for the local archive logs for that database, and the second is for the standby database.

It would be set to 2 because that is the Standby database Oracle GoldenGate should stay behind. The first entry (`log_archive_dest_1`) is for the local archive logs for that database, and the second is for the standby database.0

FETCHPARTIALJSON
Valid for Extract for MySQL
Use this option in the Extract parameter file to directly fetch data from the table, if there are partial updates to the `JSON` data type columns of a table.

> **Note:**
>
> Processing `JSON` column data updates depends on the value of the MySQL server variable, `binlog_row_value_options`, the value of which needs to be set as `PARTIAL_JSON` and the Extract parameter file includes the `FETCHPARTIALJSON` parameter.

FETCHPARTIALLOB
Valid for Extract in integrated capture mode for Oracle.
Use this option when replicating to a heterogeneous target or in other conditions where the full LOB image is required. It causes Extract to fetch the full LOB object, instead of using the partial change object from the redo record. By default, the database logmining server sends Extract a whole or partial LOB, depending on whether all or part of the source LOB was updated. To ensure the correct snapshot of the LOB, the Oracle Flashback feature must be enabled for the table and Extract must be configured to use it. The Extract `FETCHOPTIONS` parameter controls fetching and must be set to `USESNAPSHOT` (the default in the absence of `NOUSESNAPSHOT`). Without a Flashback snapshot, Extract fetches the LOB from the table, which may be a different image from the point in time when the redo record was generated.

FETCHPARTIALXML
Valid for Extract in integrated capture mode Oracle.
Use this option when replicating to a heterogeneous target or in other conditions where the full LOB image is required. It causes Extract to fetch the full XML document, instead of using the partial change image from the redo record. By default, the database logmining server sends Extract a whole or partial XML document, depending on whether all or part of the source XML was updated. To ensure the correct snapshot of the XML, the Oracle Flashback feature must be enabled for the table and Extract must be configured to use it. The Extract `FETCHOPTIONS` parameter controls fetching and must be set to `USESNAPSHOT` (the default in the absence of `NOUSESNAPSHOT`). Without a Flashback snapshot, Extract fetches the XML document from the table, which may be a different image from the point in time when the redo record was generated.

EXCLUDEFILTERTABLE *table*
Valid for Extract for MySQL, PostgreSQL, and SQL Server.
Use this option to identify a source transaction for filtering. If a source transaction includes any operation for the specified `EXCLUDEFILTERTABLE`, then that transaction is identified as a replicated transaction. Transaction filtering is based on the `GETREPLICATES`/`IGNOREREPLICATES` and `GETAPPLOPS`/`IGNOREAPPLOPS` parameters.
When a Replicat uses a checkpoint table, it writes a recovery record in the checkpoint table at the end of each transaction that it applies. Considering that all transactions applied by the Replicat contain an update to the checkpoint table, the Extract ignores the entire transaction applied by the Replicat, which prevents data looping. For PostgreSQL and SQL Server, ensure that `TRANDATA` has been added for the checkpoint table.

If using a parallel Replicat in a bidirectional replication, then multiple filter tables are supported using the `TRANLOGOPTIONS EXCLUDEFILTERTABLE` option. Multiple filter tables allow the `TRANLOGOPTIONS EXCLUDEFILTERTABLE` to be specified multiple times with different table names or wildcards.
You can include single or multiple `TRANLOGOPTIONS EXCLUDEFILTERTABLE` entries in the Extract parameter file. In the following example, multiple `TRANLOGOPTIONS EXCLUDEFILTERTABLE` entries are included in the Extract parameter file with explicit object names and wildcards.

```
TRANLOGOPTIONS EXCLUDEFILTERTABLE ggs.chkpt2
TRANLOGOPTIONS EXCLUDEFILTERTABLE ggs.chkpt_RABC_*
```

For information about creating a checkpoint table, see Add a Primary Extract. To specify object names and wildcards correctly, see Using Wildcards in Command Arguments in *Oracle GoldenGate Microservices Documentation*.

`FORCEFETCHLOB`
Valid for Extract for Oracle.
Overrides the default behavior of capturing LOB data from the redo log. Causes LOBs to be fetched from the database by default.

> **Caution:**
>
> If a value gets deleted before the fetch occurs, Extract writes a null to the trail. If a value gets updated before a fetch, Extract writes the updated value. To prevent these inaccuracies, try to keep Extract latency low. The Oracle GoldenGate documentation provides guidelines for tuning process performance. Also, see Interactions Between Fetches from a Table and DDL for instructions on setting fetch options.

`GETCTASDML | NOGETCTASDML`
Enables Create Table As Select (CTAS) functionality. When `GETCTASDML` is enabled, CTAS DMLs are sent from LogMiner and replicated on the target. This option is enabled by default. Execution of the CTAS DDL is suppressed on the target. This parameter cannot be enabled while using the DDL metadata trigger. Trail files produced with the CTAS functionality enabled cannot be consumed by a Replicat version lower than 12.1.2.1.0.
Use `GETCTASDML` to allow CTAS to replay the inserts of the CTAS thus preserving OIDs during replication. This parameter is only supported with Integrated Dictionary and any downstream Replicat must be 12.1.2.1 or greater to consume the trail otherwise, there may be divergence.

`HANDLEDLFAILOVER [ STANDBY_WARNING value | STANDBY_ABEND value ]`
Valid for Extract for Oracle
`STANDBY_WARNING` and `STANDBY_ABEND` valid for Oracle Database 21c and higher.
Controls whether Extract will throttle its writing of trail data based on the apply progress of the Fast Start Failover standby database. It is intended to keep Extract at a safe point behind any data loss failover.
When using this for data loss in a Data Guard configuration without Fast Start Failover (FSFO), you must set the `FAILOVERTARGETDESTID` Extract parameter to identify the archive log destination ID to where the standby can be connected.
Extract is found to be in a stalled state when Extract queries the standby database apply SCN information (`SELECT applied_scn FROM v$archive_dest where dest_id=n`) and this SCN is less than Extract processing LCR SCN. In this case, Extract will not process the LCR and waits until the `applied_scn` becomes greater than or equal to Extract processing LCR SCN.

STANDBY_WARNING *value*
The amount of time before a warning message is written to the Extract report file, if Extract is stalled. The default is 60 seconds.

STANDBY_ABEND *value*
The amount of time before Extract abends, if Extract is stalled. The default is 30 minutes.

If both STANDBY_WARNING and STANDBY_ABEND are specified, STANDBY_ABEND should always be greater than STANDBY_WARNING.

IFILOCKSECONDS *seconds*
Valid for Db2 z/OS
Sets the interval in seconds, for which the Extract holds the implicit locks held in the database by the calls to IFCID 0306. The locks can affect the ability to perform certain database operations such as REORGS. The default value is 20 seconds,with minimum and maximum values as 1 second and 300 seconds, respectively.

> **Note:**
>
> If the IFILOCKSECONDS parameter is set for a longer duration, other database operations such as REORGS, can be impacted due to internal locking caused by the Extract IFI calls. Therefore, if any lock contention occurs with relation to an Extract, either set the lock timeout for the operation to a duration longer than the value of the IFILOCKSECONDS parameter, or shut down the Extract.

IGNOREDATACAPTURECHANGES | NOIGNOREDATACAPTURECHANGES
Valid for Db2 LUW
Controls whether or not Extract captures tables for which DATA CAPTURE CHANGES is not set. 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.

INCLUDEREGIONID | INCLUDEREGIONIDWITHOFFSET
Valid for Extract for Oracle.
These options support the Oracle data type TIMESTAMP WITH TIME ZONE specified as TZR (which represents the time zone region, such as US/Pacific). By default, Extract abends on TIMESTAMP WITH TIME ZONE if it includes a time zone region. These options enable you to handle this timestamp based on the target database type.
When Extract detects that the source data type is TIMESTAMP and there is a region ID mapping token, Replicat applies the timestamp as follows:

- A TIMESTAMP WITH TIME ZONE with TZR is applied if the target Oracle version supports it.
- A timestamp with a UTC offset is applied to a heterogeneous database, or to an earlier version of Oracle that does not support TIMESTAMP WITH TIME ZONE with TZR.

INCLUDEREGIONID
Valid for Integrated Extract for Oracle.
The INCLUDEREGIONID is deprecated for Oracle GoldenGate 19c (19.1.0). From Oracle GoldenGate 19c (19.1.0) onward, TIMESTAMP WITH TIME ZONE with region ID data is included by default including initial load.

Use when replicating from an Oracle source to an Oracle target of the same version or later. When `INCLUDEREGIONID` is specified, Extract adds a column index and the two-byte `TMZ` value as a time-zone mapping token and outputs it to the trail in the UTC format of `YYYY-MM-DD HH:MI.SS.FFFFFF +00:00`.

`INCLUDEREGIONIDWITHOFFSET`
Valid for Integrated Extract for Oracle.
Use this option to convert region ID to hour and minutes offset value (+06:00 as example). If the option is not specified, then the timestamp is always written to the trail file in UTC and the time zone is always +00:00.
If you need to preserve the time zone value in hour and minutes instead of UTC, then this option can be used.
In the following cases, the option is forced to turn on to preserve the TIMEZONE value in hour and minutes offset:

- Old trail file format because Replicat does not support region ID.
- `XML`, `TEXT`, and `SQL` format because they don't support region ID.

`INCLUDETAG` *`tag`*
Valid for integrated Extract.
Use `INCLUDETAG` *`tag`* to include specific changes trail files. The tag value can be up to 2000 hexadecimal digits (0-9 A-F).

> **Note:**
>
> FFFF and + (plus symbol) are not supported for tag usage.

To avoid conflicts, don't use `INCLUDETAG` in conjunction with `EXCLUDETAG`.
Example: `tranlogoptions includetag 00`

`LOB_CHUNK_SIZE`
Valid for SQL Server, PostgreSQL.
If you have huge LOB data sizes, then you can adjust the `LOB_CHUNK_SIZE` from the default of 4000 bytes, to a higher value up to 65535 bytes, so that the fetch size is increased, reducing the trips needed to fetch the entire LOB
Example: `TRANLOGOPTIONS LOB_CHUNK_SIZE 8000`
(PostgreSQL) Specifies the size of chunk for the `LOB` (`CLOB`/`BLOB`) data that will be used to push in `COM`. It's unit is in bytes. The minimum and maximum `lob_chunk_size` values lies between 4000 to 65535 bytes.

`INTEGRATEDPARAMS (`*`parameter value`*` [, ...])`
Valid for Extract in integrated capture mode for Oracle Standard or Enterprise Edition 12*c* or later.
Passes parameters and values to the Oracle Database logmining server when Extract is in integrated capture mode. The input must be in the form of *`parameter value`*, as in:

```
TRANLOGOPTIONS INTEGRATEDPARAMS (downsream_real_time_mine Y)
```

Valid *`parameter`* specifications and their values are the following:

`max_sga_size`
Specifies the amount of SGA memory that is used by the database logmining server. Can be a positive integer in megabytes. The default is 1 GB if `streams_pool_size` is greater than 1 GB; otherwise, it is 75% of `streams_pool_size`.

parallelism
Specifies the number of processes supporting the database logmining server. Can be a positive integer. The default is 2.

downstream_real_time_mine
Specifies whether or not integrated capture mines a downstream mining database in real-time mode. A value of `Y` specifies real-time capture and requires standby redo logs to be configured at the downstream mining database. A value of `N` specifies capture from archived logs shipped to the downstream mining database. The default is `N`.

enable_procedural_replication
Enables procedural replication at capture. Procedural replication is disabled by default. A value of `Y` enables procedural replication. Once this option is turned on for an Extract, it remains on. The parameter value can not be toggled back.

ENABLE_AUTO_CAPTURE | DISABLE_AUTO_CAPTURE
Set this option to enable auto capture mode, which would deliver LCRs of tables enabled for automatic capture. This option can be set when the source database's Oracle binary version is 21c or higher.

MANAGESECONDARYTRUNCATIONPOINT | NOMANAGESECONDARYTRUNCATIONPOINT
Valid for PostgreSQL.

`MANAGESECONDARYTRUNCATIONPOINT` is the default setting and controls the `restart_lsn` of the replication slot for the specific Extract.
`NOMANAGESECONDARYTRUNCATIONPOINT` does not move the Extract's replication slot and is typically only used for development and testing purposes where an Extract needs to be repositioned to an earlier LSN from the Extract's recovery LSN. If used, the PostgreSQL write-ahead log will continue to grow and consume disk space.

MAXAUTOCMTTRANSSIZE (*range*, *default*)
Valid for DB2 for i only
Provides the range of the maximum autocommited transaction size.
DB2 for i autocommited records (journal entry has CCID equal to 0) do not have a commit record in the journal and therefore Oracle GoldenGate must create an implicit transaction to include these records in the trail. The default allows for a single record to be included in a single transaction, which maintains the accuracy of the indicated IO Time for each record because the IO time is based on the commit for the transaction.
This parameter sets the maximum number of records that will be included in an implicitly created transaction, but the number could be less if any other type of entry is seen in the journal before the maximum is reached. This behavior avoids issues with overlap for checkpoints on records that belong to explicitly committed records.
Setting the value for this parameter to 1 (the defulat) will provide an accurate IO time for each record in the trail for records that are autocommitted (have a CCID of 0 in the journal entry), at the potential expense of throughput for the Extract. The value of this parameter also affects the maximum potential size of a cached transaction for these records in memory. Setting it to a lower value causes the transaction memory to be lower if the Extract is able to store the maximum number of entries per implicit transaction. By definition there can only be one such implicit transaction in memory at any given time since any other transaction records will cause an immediate commit to the trail of any records in an implicit transaction already in memory.
The default range is between 1-10000 and the default value is 1.

MININGUSER {/ | *useridalias*
[*algorithm* ENCRYPTKEY {*key_name* | DEFAULT}] [SYSDBA]]
Valid for Oracle GoldenGate Extract for Oracle.

Specifies login credentials for Extract to log in to a downstream Oracle mining database to interact with the logmining server. Can be used instead of the MININGUSERALIAS option if an Oracle GoldenGate credential store is not being used.
This user must:

- Have the privileges granted in OGG_CAPTURE and OGG_APPLY role for Oracle GoldenGate 23ai and later. For Oracle database 21c, the privileges granted in dbms_goldengate_auth.grant_admin_privilege must be applied.
- Be the user that issues the MININGDBLOGIN or MININGDBLOGIN USERIDALIAS and REGISTER EXTRACT or UNREGISTER EXTRACT commands for the Extract group that is associated with this MININGDBLOGIN USERIDALIAS.
- Not be changed.

/
Directs Oracle GoldenGate to use an operating-system login for Oracle, not a database user login. Use this argument only if the database allows authentication at the operating-system level. Bypassing database-level authentication eliminates the need to update Oracle GoldenGate parameter files if application passwords frequently change.
To use this option, the correct user name must exist in the database, in relation to the value of the Oracle OS_AUTHENT_PREFIX initialization parameter. The value set with OS_AUTHENT_PREFIX is concatenated to the beginning of a user's operating system account name and then compared to the database name. Those two names must match. When OS_AUTHENT_PREFIX is set to ' ' (a null string), the user name must be created with IDENTIFIED EXTERNALLY. For example, if the OS user name is ogg, you would use the following to create the database user:

```
CREATE USER ogg IDENTIFIED EXTERNALLY;
```

When OS_AUTHENT_PREFIX is set to OPS$ or another string, the user name must be created in the format of:

```
OS_AUTHENT_PREFIX_value OS_user_name
```

For example, if the OS user name is ogg, you would use the following to create the database user:

```
CREATE USER ops$ogg IDENTIFIED BY oggpassword;
```

user
Specifies the name of the mining database user or a SQL*Net connect string.

password
The user's password. Use when database authentication is required to specify the password for the database user. If the password was encrypted by means of the ENCRYPT PASSWORD command, supply the encrypted password; otherwise, use the clear-text password. If the password is case-sensitive, type it that way. If either the user ID or password changes, the change must be made in the Oracle GoldenGate parameter files, including the re-encryption of the password if necessary.

algorithm
Specifies the encryption algorithm that was used to encrypt the password with ENCRYPT PASSWORD. Can be one of:

```
AES128
AES192
AES256
```

ENCRYPTKEY {*key_name* | DEFAULT}
Specifies the encryption key that was specified with `ENCRYPT PASSWORD`.

- `ENCRYPTKEY` *`key_name`* specifies the logical name of a user-created encryption key in the `ENCKEYS` lookup file. Use if `ENCRYPT PASSWORD` was used with the `KEYNAME` *`key_name`* option.
- `ENCRYPTKEY DEFAULT` directs Oracle GoldenGate to use a random key. Use if `ENCRYPT PASSWORD` was used with the `KEYNAME DEFAULT` option.

SYSDBA
Specifies that the user logs in as `sysdba`.

MININGUSERALIAS *alias*
Valid for Extract for Oracle.
Specifies the alias for the login credentials that Extract uses to log in to a downstream Oracle mining database to interact with the logmining server. Can be used instead of `MININGUSER` if an Oracle GoldenGate credential store is being used.
This alias must be:

- Associated with a database user login credential that is stored in the local Oracle GoldenGate credential store. This user must have the privileges granted in `dbms_goldengate_auth.grant_admin_privilege` for Oracle database 21c and lower and the `OGG_CAPTURE` and `OGG_APPLY` role for Oracle database 23ai and higher.
- The user that issues the `MININGDBLOGIN` or `MININGDBLOGINALIAS` and `REGISTER EXTRACT` or `UNREGISTER EXTRACT` commands for the Extract group that is associated with this `MININGUSERALIAS`.

This alias and user must not be changed while Extract is in integrated capture mode.

MIXEDENDIAN [ON|OFF]
Valid for Db2 LUW with Oracle GoldenGate primary Extract
Oracle GoldenGate Extract for Db2 LUW supports cross-endian capture where the database and Oracle GoldenGate are running on different byte order servers. Detection of byte order is automatic for Db2 LUW database version 10.5 and higher. If you need to disable auto-detection on Db2 LUW 10.5, then you can override it by specifying this parameter. For version 10.1, the parameter must be used in the Extract parameter file for the cross-endian capture. By default, the value is set to `OFF` for version 10.1.
Syntax:

```
TRANLOGOPTIONS MIXEDENDIAN [ON|OFF]
```

`ON`: If this is set, then the Extract assumes that the database and Oracle GoldenGate are running on servers with a different byte order and necessary byte reversal conversion is performed.
`OFF`: If this is set, then the Extract assumes that the database and Oracle GoldenGate are running on servers with the same byte order and no byte order reversal conversion is performed.

MANAGECDCCLEANUP | NOMANAGECDCCLENUP
Valid for SQL Server.

MANAGECDCCLEANUP is the default and recommended setting that instructs the Extract to validate the existence of the Oracle GoldenGate CDC Cleanup job or Purge Change Data task, depending on the architecture and version of Oracle GoldenGate.
For all Oracle GoldenGate Classic Architecture versions, and for Microservices versions prior to Oracle GoldenGate 21.4, use the ogg_cdc_cleanup_setup.bat/sh program to install the Oracle GoldenGate CDC Cleanup job and associated tables and stored procedures.
For Oracle GoldenGate Microservices 21.4 and later installations, create a Purge Change Data task from the Tasks page from the Configuration section of the Administration Service of the WebUI. The Purge Change Data task creates the required stored procedures and associated tables, and handles the purge function within the Oracle GoldenGate and not through the SQL Server Agent job.
The NOMANAGECDCCLEANUP option instructs Extract not to check for the existence of the Oracle GoldenGate CDC Cleanup job or Purge Change Data task. This is not a recommended option for production environments but can be used for testing an Extract without having to create the Oracle GoldenGate CDC Cleanup job or task.

PERFORMANCEPROFILE HIGH|MEDIUM|LOW_RES
Valid for Extract in Integrated Capture mode.

For tuning Integrated Capture.
It can be set to HIGH, MEDIUM (default), or LOW_RES. It helps achieve better performance by grouping the parameters that affect performance. Once the performance profile is set up, this option automatically configures the relevant parameters, to achieve the desired throughput and latency.

- The HIGH option allows high workload with a continuous throughput to be processed more efficiently in terms of the end-to end replication. The HIGH option increases the trail file buffer size to 4 MB and decreases the end-of-file and flush option values to 0.1 second.
- The MEDIUM option sets the trail file buffer size to 1 MB, and the values for end-of-file delay and flush to 1 second.
- The LOW_RES option is applicable when resources are low and has been added for memory or resource constrained deployment.

When HIGH option is enabled for low to medium intensity workload, it spikes the integrated Extract latency to several seconds. This is because the HIGH option increases the Extract's read buffer size to 8MB. However, the rule to flush the extract read buffer is either when the buffer is full or when there is no incoming records for a duration of 0.2 seconds. Therefore, any continuous workload with extract ingestion rate below 8MB will result in integrated Extract latency to exceed 1 second. Ensure that if the extract ingestion rates (or redo generation rates if 100% of redo is being captured) are below specific value, such as ~15 MB/sec to get ~0.5 second extract latency, do not use the HIGH option. If sub-second latency is required, it is recommended to lower the buffer size accordingly. For example, set the buffer size to one-third of the redo generation rate in MB/sec to get ~0.3 second maximum extract latency.

QUERYTIMEOUT *seconds*
Valid for SQL Server.
Specifies how long queries to SQL Server will wait for results before reporting a timeout error message. This option takes an integer value to represent the number of seconds. The default query timeout value is 300 seconds (5 minutes). The minimum value is 0 seconds (infinite timeout). The maximum is 2147483645 seconds.
The following example instructs SQL Server to wait 60 seconds for results before timing out.

```
TRANLOGOPTIONS QUERYTIMEOUT 60
```

QUERYRETRYCOUNT *seconds*
Valid for Extract for SQL Server and MySQL.
Specifies how many times to retry calls to the CDC stored procedure used by Extract, in case of a result set timeout.
QUERYRETRYCOUNT can be specified to retry multiple times. If all of the retry attempts fail, Extract abends with the normal connection timeout error message.
For SQL Server, the default is one retry attempt, after which the process abends.The minimum setting (0) is infinite, maximum is 1000, and default is 1.
For MySQL, the minimum and default setting is 50 and maximum is 1000. There is no infinite value. Any attempt to set the QUERYRETRYCOUNT to less than minimum, will be ignored with no error or warning.
The following example causes Extract to attempt its CDC stored procedure call 4 times:

```
TRANLOGOPTIONS QUERYRETRYCOUNT 4
```

The following example causes Extract to attempt its CDC stored procedure call 100 times:

```
TRANLOGOPTIONS QUERYRETRYCOUNT 100
```

READQUEUESIZE *size*
Valid for MySQL.
Specifies the internal queue size, in bytes, for transaction data. It can be increased to improve performance. Valid values are integers from 3 through 1500. The default is 256 bytes; start with the default and evaluate performance before adjusting upward.

REDO_TRANSPORT_LAG_THRESHOLD *seconds*
Valid for Integrated Extract in Downstream Mining Mode.
Monitors the network latency between a source database and target database when redo logs are shipped. If the latency exceeds the specified threshold then a warning appears in the report file and a subsequent information message appears when the lag drops to the normal level.
The default threshold value is 30 seconds. The minimum threshold value that can be specified is 15 seconds.
For more information, see Downstream Extract

REDO_TRANSPORT_LAG_TIMEOUT *value*
Valid for Integrated Extract in Downstream Mining Mode.
The value provided as input in this parameter option is the time period for which Extract will wait for redo from each thread. If all the threads have waited for the timeout (in seconds) and have not received any redo then Extract will abend.

REQUIRELONGDATACAPTURECHANGES | NOREQUIRELONGDATACAPTURECHANGES
Valid for 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 any of the following Oracle GoldenGate parameters that require the presence of before images for some or all column values: GETBEFOREUPATES, NOCOMPRESSUPDATES, and NOCOMPRESSDELETES. Both of those DATA CAPTURE settings prevent the logging of before values for LONGVAR columns. If those columns are not available to Extract, it can affect the integrity of the target data.

REQUIRELONGDATACAPTURECHANGES
Extract abends with an error.

NOREQUIRELONGDATACAPTURECHANGES
Extract issues a warning but continues processing the data record.

SOURCE_OS_TIMEZONE *timezone*
Valid for Extract in integrated capture mode for Oracle.
Specifies the system time zone of the source database. The system time zone of a database is usually given by the default time zone of its operating system, and can also be overridden by setting the `TZ` environment variable when the database is started. You should specify this option only if the source database and the Extract process use different system time zones. For example, in a downstream capture deployment where the source database and the Extract process run on different servers in different time zones.
You can specify the value of this option in a time zone region name or a UTC offset form and you must use the same form used by the source database. For example, if the source database uses a region name form like America/New_York, then you must specify America/New_York, US/Eastern, or EST5EDT. Alternately, if the source database uses a UTC offset form like -05:00, then you must use the syntax `(GMT)[+|-]hh[:mm]`. For example, GMT-05:00 or -5.

SKIPUNKNOWNEVENT
Valid for MySQL.
You can use this parameter in the Extract parameter file to enable skipping any unhandled or unknown event in the MySQL binary log. If this parameter is specified, then the Oracle GoldenGate for MySQL Extract continues processing without any error on finding an event that is not handled by the current Extract process.

SUPPRESSNOOOPUPDATES
Valid for Extract on Oracle Database.
You can control whether no-op updates are filtered or not in Integrated Extract. The default is no suppression.

TRACKSCHEMACHANGES
Valid for Db2 z/OS and MySQL

This parameter enables Extract to capture table level DDL statements and retain a history of the changes to be used to process DML when the log records refers to a table version that is earlier than the current version of the table. This would usually be before images of updates, but could be after images, inserts or deletes if the Extract is running in a lag situation from the log backlog. When Extract encounters appropriate DDL operations, it will note the version number of the DDL and update the DDL history table with the new information. The Extract will create a new TDR record that relates to the change in the trail as well. When Extract encounters prior versions of the table in the log, it will reference the DDL history to be able to correctly interpret the DDL for the older version of the table. The DDL change is not actually being replicated, and synchronization of any changes to the source table are still required to be manually executed by the user in the target database.
Syntax:
`TRANLOGOPTIONS TRACKSCHEMACHANGES`
This will enable table level DDL changes to be tracked by the Extract and the trail metadata updated as appropriate. To use `TRACKSCHEMACHANGES` properly, the table metadata must be at a known consistent state, which means that all the tables that need version tracking must be created and never altered or reorganized before using `TRACKSCHEMACHANGES` so that no prior table versions will appear in the transaction log for update or delete operations. The script `ddl_update.sh` has been provided to assist in the creation of an initial set of DDL history records for the database.
To use DDL processing, the database needs to be set up with a history table that will capture DDL changes of the various versions of all tables on the database system. Also system tables

need to be enabled for data capture changes. To create and maintain the history table the following UNIX shell scripts are provided:

- `ddl_create.sh` : This script is used to create the DDL history table. It also enables data capture changes for the following system tables:
  - `SYSIBM.SYSTABLES`
  - `SYSIBM.SYSCOLUMNS`
  - `SYSIBM.SYSINDEXES`
  - `SYSIBM.SYSKEYCOLUSE`

  Example:

  ```
  ./ddl_create.sh -f crt_ddl_hist.sql -s OGGSCHEMA
  ```

  In this example, the resulting file, `crt_ddl_hist.sql` must be processed by another program.

  ```
  ./ddl_create.sh -d DB2DSXY -u gguser -p ggpw -s OGGSCHEMA
  ```

  This will call a local Db2 to make a remote connection to a mainframe database to immediately create the DDL history table.
- `ddl_remove.sh`: This script is used to remove the DDL history table. However, the system tables are not altered.
- `ddl_update.sh`: This script should be run to establish an initial start point for the DDL version tracking using `TRACKSCHEMACHANGES`. It must be run after the creation of the DDL history table and should not be necessary to be run again unless the extract must be repositioned in such a way that table versions may be missed in the transaction log. Rerunning ddl_update.sh will only add information for tables that are not already present in the DDL history.
- `execsql.sh`: This script should not be run directly but must be available to the other scripts to be sourced. It provides common facilities for parsing command line and executing SQL or writing it to a file.

These scripts may be used to create the tables directly using a db2 connection or they may be used to create SQL files which may be run in a SQL processing program of choice. The files have been checked to be compatible with DB2 remote and `SPUFI`.
Following is a description of options accepted by these scripts:

- `-h` shows this usage help.
- `-d` *`dsn`* specifies the DB2 DSN to connect to.
- `-u` *`userid`* specifies the User ID to connect to the database with.
- `-p` *`password`* specifies the password to connect to the database with.
- `-s` *`ggschema`* specifies the name of the schema the DDL history table should be stored in. This schema should be the same as `GGSCHEMA` in GLOBALS.
- `-t` *`ddltable`* specifies the name of the table the DDL history table. `GGS_DDL_HIST` is the default.
- `-f` *`outfile`* specifies the name of a file to write the SQL statements to instead of executing them. If `-d`, `-u`, `-p` must all be specified if used or `-f`. Currently `-t` should not be used. `-f` must be used if only the `db2cli` command is available on the remote host since `db2cli` cannot run the SQL statements that are generated.

TRANCOUNT
Valid for SQL Server.
Allows adjustment of the number of transactions processed per call by Extract to pull data from the SQL Server change data capture staging tables. Based on your transaction workload, adjusting this value may improve capture rate throughput. The minimum value is 1, maximum is 100, and the default is 10.
Example:

```
TRANLOGOPTIONS TRANCOUNT 20
```

This example instructs Extract to fetch 20 transactions at a time from change data capture enabled tables.

[TSLOOKUPBEGINLRI | TSLOOKUPENDLRI]
Valid for Db2 LUW v 10.1 and later.
When you specify an LRI range using these parameters, Extract looks for the timestamp specified in the `ADD` or `ALTER EXTRACT` command within this range. This helps Extract to optimize the look up process for a particular timestamp in the database transaction log. The `TSLOOKUPBEGINLRI` parameter is mandatory while `TSLOOKUPENDLRI` is optional. Specifying only `TSLOOKUPENDLRI` without `TSLOOKUPBEGINLRI` is invalid that causes the Extract to abend. For example:

```
TRANLOGOPTIONS TSLOOKUPBEGINLRI 75200.666197, TSLOOKUPENDLRI  75207.666216
TRANLOGOPTIONS TSLOOKUPBEGINLRI 75200.666197
```

If the provided timestamp falls between the given LRI ranges or the provided timestamp falls after the `TSLOOKUPBEGINLRI` LRI timestamp then Extract starts from a record with timestamp equal to or nearest less than the provided timestamp.
If the provided timestamp falls before `TSLOOKUPBEGINLRI` LRI timestamp, Extract is started from the specified `TSLOOKUPBEGINLRI` LRI. If the provided timestamp falls after `TSLOOKUPENDLRI` timestamp, then Extract abends.

USENATIVEOBJSUPPORT | NOUSENATIVEOBJSUPPORT
Valid for Extract in integrated capture mode for Oracle.
Integrated Capture adds redo-based capture for User Defined Type (UDT) and `ANYDATA` data types. It is enabled by default and can only be enabled if the source database version is 12.1.0.1 or greater and the source database compatibility is 12.0.0.0.0 or greater. Replicat from Oracle GoldenGate release 12.1.2.1.0 must be used. To use Native Support, all of your Oracle databases and Oracle GoldenGate instances must be release 12.1.0.1 or greater to be compatible.
If redo-based capture is enabled but a UDT contains an unsupported attribute, Integrated Capture retries to capture the UDT using fetch. For limitations of support for capture, see XML Data Types. If you create object tables by using a `CREATE TABLE AS SELECT` (CTAS) statement, Integrated Capture must be configured to capture DML from CTAS operation in order to fully support object tables. For CTAS, refer to How Oracle GoldenGate Handles Derived Object Names in *Oracle GoldenGate Microservices Documentation*
The default is `USENATIVEOBJSUPPORT` if supported.

USE_ROOT_CONTAINER_TIMEZONE
Valid for Oracle integrated Extract only.
This parameter is for a CDB environment. Each PDB in a CDB can use a different database time zone. If the database time zone is available, Extract tries to get the time zone of a PDB from Integrated Dictionary. The time zone extraction requires a patch on the mining database. If the patch is not available, Extract sends a query to the PDB to get the time zone. If the database patch or a connection to the PDB is not available, and this parameter is specified,

Extract assumes that the PDB database time zone is the same as the root container database time zone.

VERSIONCHECK DYNAMIC | IMMEDIATE
This is valid for SQL Server.
Use this option when when you want Extract to validate CDC object versions of common stored procedures, such as OracleCDCExtract and OracleGGCreateProcs, at startup.
DYNAMIC (default) identifies the CDC object versions of table specified in the parameter file once per table *while* records are processed.
IMMEDIATE identifies CDC object version issues upfront, instead of while records are processed. Databases with large numbers of tables configured for capture require longer startup validation.

TRANSACTIONTIMEOUT

Valid For

Replicat

Description

Use the TRANSACTIONTIMEOUT parameter to prevent an uncommitted Replicat target transaction from holding locks on target tables and consuming database 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 range is 1–604800.

The following events could last long enough to trigger TRANSACTIONTIMEOUT:

- Network problems prevent trail data from being delivered to the target system.
- Running out of disk space on any system, preventing trail data from being written.
- Collector abends (a rare event).
- Extract abends or is terminated in the middle of writing records for a transaction.
- An Extract data pump abends or is terminated.
- There is a source system failure, such as a power outage or system crash.

How TRANSACTIONTIMEOUT Works

During normal operations, Replicat remembers the position in the trail of the beginning of the first source transaction in the current target transaction, in case the transaction must be abended and retried. When TRANSACTIONTIMEOUT is enabled, Replicat also saves the position of the first record of the current source transaction and will use that position as the logical end-of-file (EOF) if TRANSACTIONTIMEOUT is triggered.

When triggered, TRANSACTIONTIMEOUT does the following:

1. Aborts the current target transaction

2. Repositions to the beginning of the first source transaction in the aborted target transaction.
3. Processes all of the trail records up to the logical end-of-file position (the beginning of the last, incomplete source transaction).
4. Commits the transaction at logical EOF point.
5. Waits for new trail data before processing any more trail records.

`TRANSACTIONTIMEOUT` can be triggered multiple times for the same source transaction, depending on the nature of the problem that is causing the trail data to arrive slowly enough to trigger `TRANSACTIONTIMEOUT`.

Detecting 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
```

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
```

TRIMSPACES | NOTRIMSPACES

Valid For

Extract and Replicat

Description

Use the `TRIMSPACES` and `NOTRIMSPACES` parameters to control whether or not trailing spaces in a source `CHAR` column are truncated when applied to a target `CHAR` or `VARCHAR` column. `TRIMSPACES` and `NOTRIMSPACES` can be used at the root level of the parameter file as global `ON/OFF` switches for different sets of `TABLE` or `MAP` statements, and they can be used within an individual `TABLE` or `MAP` statement to override any global settings for that particular `MAP` or `TABLE` statement.

TRIMSPACES is applied only to single-byte white spaces (U+0020). Ideographic spaces (U+3000) are not supported.

For Extract, TRIMSPACES only has an effect if Extract is performing mapping within the TABLE statement (by means of a TARGET statement).

Default

TRIMSPACES

Syntax

```
TRIMSPACES | NOTRIMSPACES
```

Examples

Example 1
The following example uses TRIMSPACES and NOTRIMSPACES at the root level of the parameter file. The default of TRIMSPACES is in effect until the last MAP statement, to which NOTRIMSPACES applies.

```
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;
```

Example 2
The following example uses NOTRIMSPACES within a MAP statement to override the global default of TRIMSPACES. The default applies to the first two MAP statements, and then NOTRIMSPACES applies to 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;
```

TRIMVARSPACES | NOTRIMVARSPACES

Valid For

Extract and Replicat

Description

Use the TRIMVARSPACES and NOTRIMVARSPACES parameters to control whether or not trailing spaces in a source VARCHAR column are truncated when applied to a target CHAR or VARCHAR column. TRIMVARSPACES and NOTRIMVARSPACES can be used at the root level of the parameter file as global ON/OFF switches for different sets of TABLE or MAP statements, and they can be used within an individual TABLE or MAP statement to override any global settings for that particular MAP or TABLE statement.

The default is NOTRIMVARSPACES because the spaces in a VARCHAR column can be part of the data. Before using TRIMVARSPACES, make certain that trailing spaces are not required as part of the target data.

For Extract, TRIMVARSPACES only has an effect if Extract is performing mapping within the TABLE statement (by means of a TARGET statement).

Default

NOTRIMVARSPACES

Syntax

```
TRIMVARSPACES | NOTRIMVARSPACES
```

Examples

Example 1
The following example uses TRIMVARSPACES and NOTRIMVARSPACES at the root level of the parameter file. The default of NOTRIMVARSPACES is in effect until the last MAP statement, to which TRIMVARSPACES applies.

```
MAP fin.src1, TARGET fin.tgt1;
MAP fin.src2, TARGET fin.tgt2;
MAP fin.src3, TARGET fin.tgt3;
TRIMVARSPACES
MAP fin.src4, TARGET fin.tgt4;
```

Example 2
The following example uses TRIMVARSPACES within a MAP statement to override the global default of NOTRIMVARSPACES. The default applies to the first two MAP statements, and then TRIMVARSPACES applies to the last two targets.

```
MAP fin.src1, TARGET fin.tgt1;
MAP fin.src1, TARGET fin.tgt2;
MAP fin.src1, TARGET fin.tgt3, TRIMVARSPACES;
MAP fin.src1, TARGET fin.tgt4, TRIMVARSPACES;
```

UPDATEDELETES | NOUPDATEDELETES

Valid For

Replicat

Description

Use the UPDATEDELETES parameter to convert delete operations to update operations for all MAP statements that are specified after it in the parameter file. Use NOUPDATEDELETES to turn off UPDATEDELETES. These parameters are table-specific. One remains in effect for subsequent MAP statements until the other is encountered. The UPDATE WHERE clause uses the same key columns that the DELETE statement was going to use. So this works best on tables that have a primary key or unique key.

Because you can selectively enable or disable these parameters between MAP statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the UPDATEDELETES threads in one set of MAP statements, and specify the NOUPDATEDELETES threads in a different set of MAP statements.

When using UPDATEDELETES, use the NOCOMPRESSDELETES parameter. This parameter causes Extract to write all of the columns to the trail, so that they are available for updates.

Default

NOUPDATEDELETES

Syntax

```
UPDATEDELETES | NOUPDATEDELETES
```

Example

This example shows how you can apply UPDATEDELETES and NOUPDATEDELETES selectively to different MAP statements, each of which represents a different thread of a coordinated Replicat.

```
UPDATEDELETES
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
NOUPDATEDELETES
MAP sales.loc, TARGET sales.loc, THREAD (3);
```

UPDATEINSERTS | NOUPDATEINSERTS

Valid For

Replicat

Description

Use the UPDATEINSERTS parameter to convert insert operations to update operations for all MAP statements that are specified after it in the parameter file. Use NOUPDATEINSERTS to turn off UPDATEINSERTS. The UPDATE WHERE clause will use the same key columns as a regular UPDATE statement. So, it works best on tables that have a primary key or unique key.

Because you can selectively enable or disable these parameters between MAP statements, you can enable or disable them for different threads of a coordinated Replicat. Specify the UPDATEINSERTS threads in one set of MAP statements, and specify the NOUPDATEINSERTS threads in a different set of MAP statements.

Default

```
NOUPDATEINSERTS
```

Syntax

```
UPDATEINSERTS | NOUPDATEINSERTS
```

Example

This example shows how you can apply UPDATEINSERTS and NOUPDATEINSERTS selectively to different MAP statements, each of which represents a different thread of a coordinated Replicat.

```
UPDATEINSERTS
MAP sales.cust, TARGET sales.cust, THREAD (1);
MAP sales.ord, TARGET sales.ord, THREAD (2);
NOUPDATEINSERTS
MAP sales.loc, TARGET sales.loc, THREAD (3);
```

UPDATERECORDFORMAT

Valid For

Extract for all databases. This parameter can only be used in a `TRANLOG` Extract. It cannot be used in an Extract or distribution service.

Description

Use the `UPDATERECORDFORMAT` parameter to cause Extract to combine the before and after images of an `UPDATE` operation into a single record in the trail. It is valid for Extract in classic and integrated capture modes.

Before images are generated when the `GETUPDATEBEFORES`, `GETBEFORECOLS`, and `LOGALLSUPCOLS` parameters are used. (In the case of an update to a primary key, unique index, or user-specified `KEYCOLS` key, the before and after images are stored in the same record by default. `UPDATERECORDFORMAT` does not apply in these cases.) The `NOCOMPRESSUPDATES` parameter is required for non-Oracle databases.

When two records are generated for an update to a single row, it incurs additional disk I/O and processing for both Extract and Replicat. If supplemental logging is enabled on all columns, the unmodified columns may be repeated in both the before and after records. The overall size of the trail is larger, as well. This overhead is reduced by using `UPDATERECORDFORMAT`.

When `UPDATERECORDFORMAT` is used, Extract writes the before and after images to a single record that contains all of the information needed to process an `UPDATE` operation. In addition to improving the read performance of downstream processes, this enables column mapping functions to access the before and after column values at the same point in time, rather than having to cache the before image column values while reading the after values.

`UPDATERECORDFORMAT` takes effect for all `TABLE` statements in the parameter file.

If you specify both `UPDATERECORDFORMAT` and `FORMAT RELEASE 11.x` or earlier, then Extract will abend.

> **Note:**
>
> Many-columned tables can cause the trail record to reach its maximum size when `UPDATERECORDFORMAT` is used. The rest of the record is continued in one or more additional, chained record fragments. This has a minor effect on processing performance.

> **Note:**
>
> `INSERTALLRECORDS` only works with `UPDATERECORDFORMAT FULL`. So if you are using `INSERTALLRECORDS` in Replicat, you must set `UPDATERECORDFORMAT FULL` in the Extract.

Default

If you specify UPDATERECORDFORMAT, you have to set COMPACT or FULL. But for Oracle database, if neither of the parameters, logallsupcols or updaterecordformat, is specified, then logallsupcols and updaterecordformat compact is set by default.

If you specify the parameter, then you must specify the option full or compact.

For heterogeneous database, you have to set compact or full, if using the parameter. If not using the parameter, then updaterecordformat is not on at all and you get a single after image in the trail, unless you enable getupdatebefores, then you get two records in the trail and not a unified record.

Syntax

```
UPDATERECORDFORMAT [FULL | COMPACT]
```

FULL
Generates one trail record that contains the before and after images of an UPDATE, where the before image includes all of the columns that are available in the transaction record for both the before and after images. When viewed in the Logdump utility, this record appears as GGSUnifiedUpdate.

COMPACT
Generates one trail record that contains the before and after images of an UPDATE, where the before image includes all of the columns that are available in the transaction record, but the after image is limited to the primary key columns and the columns that were modified in the UPDATE. UPDATERECORDFORMAT COMPACT is recommended for configurations that include an integrated Replicat. This is the default.
When either FULL or COMPACT are viewed in the Logdump utility, the record appears as GGSUnifiedUpdate. The record contains the following:

- a header
- the length of the before image
- the before values of each column
- the after values of the primary key, unique index, or KEYCOLS columns
- the after values of the modified columns
- internal token data

Example

```
UPDATERECORDFORMAT COMPACT
```

USEDEDICATEDCOORDINATIONTHREAD

Valid For

Replicat (coordinated mode)

Description

Use USEDEDICATEDCOORDINATIONTHREAD to force Replicat to maintain a dedicated coordination thread to apply barrier transactions. The thread ID of this thread is always 0.

By default, Replicat uses the thread with the lowest thread ID to apply barrier transactions, but that thread also includes work that is mapped to it explicitly. By using a dedicated thread for barrier transactions, you can get an accurate view in Oracle GoldenGate statistics of the number of barrier events and exposes the amount of work that is performed serially. Coordinated Replicat statistics are written to the report file and also can be viewed with the `STATS REPLICAT` command.

`USEDEDICATEDCOORDINATIONTHREAD` applies to the Replicat group as a whole, across all `MAP` statements.

Syntax

```
USEDEDICATEDCOORDINATIONTHREAD
```

Example

```
USEDEDICATEDCOORDINATIONTHREAD
MAP u1.t1, TARGET u2.t1 SQLEXEC &
(ID test2, QUERY 'delete from u2.t2 where col_val =100 ',  &
NOPARAMS)), THREAD(1), COORDINATED;
```

USEIPV4 | USEIPV6

Valid For

GLOBALS

Description

Use the `USEIPV4` parameter to force the use of Internet Protocol version 4 (IPv4) by Oracle GoldenGate for TCP/IP connections. By default, Oracle GoldenGate uses IPv6 in dual-stack mode and this parameter forces the use of IPv4 only.

When `USEIPV4` is used, the entire network in which Oracle GoldenGate operates must be IPv4 compatible.

Use the `USEIPV6` parameter to force the use of Internet Protocol version 6 (IPv6) by Oracle GoldenGate for TCP/IP connections. By default, Oracle GoldenGate uses IPv6 in dual-stack mode but falls back to IPv4, and only then to IPv6. `USEIPV6` eliminates the IPv4 fallback step. The order of socket selection becomes:

- IPv6 dual-stack
- IPv6

When `USEIPV6` is used, the entire network in which Oracle GoldenGate operates must be IPv6 compatible.

Default

Disabled

Syntax

```
USEIPV4
```

```
USEIPV6
```

USERIDALIAS

Valid For

Extract, Replicat, DEFGEN, Admin Client.

Supported for

All supported databases for the release.

Description

Use the `USERIDALIAS` parameter to specify authentication for an Oracle GoldenGate process to use when logging into a database. The use of `USERIDALIAS` requires the use of an Oracle GoldenGate credential store. Specify `USERIDALIAS` before any `TABLE` or `MAP` entries in the parameter file. The privileges that are required for the `USERIDALIAS` user vary by database.

USERIDALIAS Compared to USERID

`USERIDALIAS` enables you to specify an alias, rather than a user ID and password, in the parameter file. The user IDs and encrypted passwords are stored in a credential store. `USERIDALIAS` supports databases running on Linux, UNIX, and Windows platforms.

`USERID` requires either specifying the clear-text password in the parameter file or encrypting it with the `ENCRYPT PASSWORD` command and, optionally, storing an encryption key in an `ENCKEYS` file. `USERID` supports a broad range of the databases that Oracle GoldenGate supports. In addition, it supports the use of an operating system login for Oracle databases.

USERIDALIAS Requirements Per Database Type

The usage of `USERIDALIAS` varies depending on the database type.

> **Note:**
>
> Login that requires a database user and password must be stored in the Oracle GoldenGate credential store.

The supported specified `USERID` in `USERIDALIAS` in Oracle GoldenGate with MySQL MA are:

- `user/db`
- `user@host/db`
- `user@host:port/db`

If an Oracle GoldenGate user is allowed to alter or add a `USERIDALIAS` that does not follow any of the preceding patterns, the user will see the error in pattern matching on the client such as the MA web interface client or Admin Client.

DB2 for LUW

Use `USERIDALIAS` parameter for all Oracle GoldenGate processes that connect to a Db2 LUW database using database authentication.

MySQL

Use `USERIDALIAS` for all Oracle GoldenGate processes that connect to a MySQL database.

Oracle

Use `USERIDALIAS` for Oracle GoldenGate processes that connect to an Oracle Database.

- Specify the alias of a database credential that is stored in the Oracle GoldenGate credential store.
- Special database privileges are required for the `USERIDALIAS` user when Extract is configured to use `LOGRETENTION`. These privileges might have been granted when Oracle GoldenGate was installed., see Configure Logging Properties for more information about `LOGRETENTION`.
- To use `USERIDALIAS` for an Extract group that is configured for integrated capture, the user must have privileges. If using Oracle Database 23ai and higher, then the user privileges can be granted with the `OGG_CAPTURE` and `OGG_APPLY` user roles. If using Oracle database 21c or lower, then the user must have privileges in the `dbms_goldengate_auth.grant_admin_privilege` procedure. The user must be the same one that issues `DBLOGIN` and `REGISTER EXTRACT` or `UNREGISTER EXTRACT` for the Extract group that is associated with this `USERIDALIAS`.

  For details on granting Oracle Database 23ai user privileges, see Grant User Privileges for Oracle Database 23ai and higher.

  For details on granting privileges for Oracle Database 21c and lower, see Grant User Privileges for Oracle Database 21c and lower

USERIDALIAS with Kerberos Authentication

Supported for Oracle Database only.

You need to set the `ALTER CREDENTIALSTORE` to use the `NOPASSWORD` keyword as the authentication is external. See ALTER CREDENTIALSTORE to know more.

SQL Server

Use `USERIDALIAS` if the ODBC data source connection that will be used by the Oracle GoldenGate process is configured to connect using SQL Server authentication.

- 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.

Teradata

Use `USERIDALIAS` for Oracle GoldenGate processes that connect to a Teradata database.

On a target Teradata system, use the `TARGETDB` parameter to specify the target ODBC data source.

Default

None

Syntax

```
USERIDALIAS alias [DOMAIN domain] [SYSDBA]
[, THREADS (threadID[, threadID][, ...][, thread_range[, thread_range]
[, ...])]
```

alias
Specifies the alias of a database user credential that is stored in the Oracle GoldenGate credential store.

DOMAIN *domain*
Specifies the credential store domain for the specified alias. A valid domain entry must exist in the credential store for the specified alias.

SYSDBA
(Oracle) Specifies that the user logs in as `sysdba`.

THREADS (*threadID*[, *threadID*][, ...][, *thread_range*[, *thread_range*][, ...])
Valid for Replicat. Links the specified credential to one or more threads of a coordinated Replicat. Enables you to specify different logins for different threads.

> ***threadID*[, *threadID*][, ...]**
> Specifies a thread ID or a comma-delimited list of threads in the format of `threadID, threadID, threadID`.
>
> **[, *thread_range*[, *thread_range*][, ...]**
> Specifies a range of threads in the form of `threadIDlow-threadIDhigh` or a comma-delimited list of ranges in the format of `threadIDlow-threadIDhigh, threadIDlow-threadIDhigh`.

A combination of these formats is permitted, such as `threadID, threadID, threadIDlow-threadIDhigh`.

TimesTen

Use `USERIDALIAS` for Oracle GoldenGate processes that connect to a TimesTen database.

On a target TimesTen system, use the `TARGETDB` parameter to specify the target ODBC data source.

Default

None

Syntax

```
USERIDALIAS alias [DOMAIN domain] [SYSDBA]
[, THREADS (threadID[, threadID][, ...][, thread_range[, thread_range][, ...])]
```

alias
Specifies the alias of a database user credential that is stored in the Oracle GoldenGate credential store.

DOMAIN *domain*
Specifies the credential store domain for the specified alias. A valid domain entry must exist in the credential store for the specified alias.

SYSDBA
(Oracle) Specifies that the user logs in as `sysdba`.

THREADS (*threadID*[, *threadID*][, ...][, *thread_range*[, *thread_range*][, ...])
Valid for Replicat. Links the specified credential to one or more threads of a coordinated Replicat. Enables you to specify different logins for different threads.

***threadID*[, *threadID*][, ...]**
Specifies a thread ID or a comma-delimited list of threads in the format of `threadID, threadID, threadID`.

[, *thread_range*[, *thread_range*][, ...]
Specifies a range of threads in the form of `threadIDlow-threadIDhigh` or a comma-delimited list of ranges in the format of `threadIDlow-threadIDhigh, threadIDlow-threadIDhigh`.

A combination of these formats is permitted, such as `threadID, threadID, threadIDlow-threadIDhigh`.

Examples

Example 1
The following supplies a credential for the user in the credential store that has the alias of `tiger1` in the domain of `east`.

```
USERIDALIAS tiger1 DOMAIN east
```

Example 2
The following supplies a credential for thread 3 of a coordinated Replicat.

```
USERIDALIAS tiger1 DOMAIN east THREADS (3)
```

Example 3
The following example shows the use of the parameter in PostgreSQL:

```
USERIDALIAS pgdsn
```

The following example shows using Kerberos authentication with Oracle GoldenGate Admin Client:

```
OGG (http://localhost:9005 demo)4> dblogin useridalias ggma
Successfully logged into database CDB1_PDB1
```

In this example, the credential store is previously set up using the `ALTER CREDENTIALSTORE` command.

VARWIDTHNCHAR | NOVARWIDTHNCHAR

Valid For

Extract, Replicat, DEFGEN for Oracle

Description

Use the `VARWIDTHNCHAR` and `NOVARWIDTHNCHAR` parameters to control how `NCHAR` data is written to the trail and interpreted by Replicat.

- VARWIDTHNCHAR causes an NCHAR, NVARCHAR2, or NCLOB character set to be treated as a variable-length character set (UTF-8).
- NOVARWIDTHNCHAR causes an NCHAR, NVARCHAR2, or NCLOB character set to be treated as UTF-16.
- If neither option is specified, the NLS_NCHAR_CHARACTERSET property value from the database is used to determine how an NCHAR, NVARCHAR2, or NCLOB character set is treated.

Default

Use NLS_NCHAR_CHARACTERSET property from database

Syntax

```
VARWIDTHNCHAR | NOVARWIDTHNCHAR
```

WARNLONGTRANS

Valid For

Extract

Description

Use the WARNLONGTRANS parameter to specify a length of time that a transaction can be open before Extract generates a warning message that the transaction is long-running. Also use WARNLONGTRANS to control the frequency with which Oracle GoldenGate checks for long-running transactions.

This parameter is valid for Oracle and SQL Server.

When WARNLONGTRANS is specified, Oracle GoldenGate checks for transactions that satisfy the specified threshold, and it reports the first one that it finds to the Oracle GoldenGate error log, the Extract report file, and the system log. By default, Oracle GoldenGate repeats this check every five minutes.

To view a list of open transactions on demand, to output transaction details to a file, or to either cancel those transactions or force them to the trail, use the options of the SEND EXTRACT command.

Default

One hour (and check every five minutes using a separate processing thread)

Syntax

```
WARNLONGTRANS duration
[, CHECKINTERVAL interval]
[, NOUSETHREADS]
[, USELASTREADTIME]
```

duration
Sets a length of time after which an open transaction is considered to be long-running. The duration is specified as a whole number, followed by the unit of time in any of the following formats to indicate seconds, minutes, or hours. Do not put a space between the numeric value and the unit of time. The unit is not case-sensitive. The default is one hour.

```
S|SEC|SECS|SECOND|SECONDS
M|MIN|MINS|MINUTE|MINUTES
H|HOUR|HOURS
D|DAY|DAYS
```

The following are examples of valid durations:

```
WARNLONGTRANS 1DAY
WARNLONGTRANS 600sec
WARNLONGTRANS 40s
```

CHECKINTERVAL *interval*
Sets the frequency at which Oracle GoldenGate checks for transactions that satisfy WARNLONGTRANS and reports the longest running one. The interval is specified as a whole number, followed by the unit of time in any of the following formats to indicate seconds, minutes, or hours. Do not put a space between the numeric value and the unit of time. The unit is not case-sensitive. The default is five minutes, which is also the minimum valid value. The minimum value is 300 and the maximum is 20000000.

```
S|SEC|SECS|SECOND|SECONDS
M|MIN|MINS|MINUTE|MINUTES
H|HOUR|HOURS
D|DAY|DAYS

CHECKINTERVAL 1day
CHECKINTERVAL 600SEC
CHECKINTERVAL 2m
```

NOUSETHREADS
Valid for Oracle.
Specifies that the monitoring will be done by the main process thread. By default, it is done with a separate thread for performance reasons. NOUSETHREADS should only be used if the system does not support multi-threading.

USELASTREADTIME
Valid for Oracle.
Forces Extract to always use the time that it last read the Oracle 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

```
NOUSETHREADS
```

WARNRATE

Valid For

Replicat

Description

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.

When setting WARNRATE for a coordinated Replicat, take into account that the specified WARNRATE threshold is applied to each thread in the configuration, not as an aggregate threshold for Replicat as a whole. For example, if WARNRATE 100 is specified, it is possible for each thread to return 99 errors without a warning from Replicat.

For Replicat running in an Oracle environment, this parameter is valid for nonintegrated mode only.

Default

100 errors

Syntax

```
WARNRATE number_of_errors
```

number_of_errors
The number of SQL errors after which a warning is issued.

Example

```
WARNRATE 1000
```

WILDCARDRESOLVE

Valid For

Extract and Replicat

Description

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}
```

DYNAMIC
Source objects that satisfy the wildcard definition are resolved each time the wildcard rule is satisfied. The newly resolved object is included in the Oracle GoldenGate configuration upon resolution. This is the default. This is the required setting for Teradata.
Do not use this option when SOURCEISTABLE or GENLOADFILES is specified. WILDCARDRESOLVE will always be implicitly set to IMMEDIATE for these parameters.
DYNAMIC must be used when using wildcards to replicate Oracle sequences with the SEQUENCE parameter.
To keep the default of DYNAMIC, an explicit WILDCARDRESOLVE parameter is optional, but its presence helps make it clear to someone who is reviewing the parameter file which method is being used.

IMMEDIATE
Source objects that satisfy the wildcard definition are processed at startup. This option is not supported for Teradata. This is the forced default for `SOURCEISTABLE`.
This option does not support the Oracle interval partitioning feature. Dynamic resolution is required so that new partitions are found by Oracle GoldenGate.

Example

The following example resolves wildcards at startup.

```
WILDCARDRESOLVE IMMEDIATE
TABLE hq.acct_*;
```

Y2KCENTURYADJUSTMENT | NOY2KCENTURYADJUSTMENT

Valid For

Extract and Replicat

Description

Use the `Y2KCENTURYADJUSTMENT` and `NOY2KCENTURYADJUSTMENT` parameters to control the conversion of year values when the century portion consists of zeroes (such as 0055) or is not specified (such as in a two-digit, year-only specification).

With `Y2KCENTURYADJUSTMENT` enabled (the default), a two-digit year value that is greater than or equal to 50 is converted to a four-digit year in the 20th century (19xx). If a two-digit year value is less than 50, it is converted to a four-digit year in the 21st century (20xx).If the century portion of the year is non-zero, or if `NOY2KCENTURYADJUSTMENT` is specified, no conversion is performed.

Default

```
Y2KCENTURYADJUSTMENT
```

Syntax

```
Y2KCENTURYADJUSTMENT | NOY2KCENTURYADJUSTMENT
```

3
Table and Column Mapping Functions

The Table and Column mapping functions of Oracle GoldenGate enable you to manipulate source values into the appropriate format for target tables and columns.

You can manipulate numbers and characters, perform tests, extract parameter values, return environment information, and more using these functions. See Mapping and Manipulating Data to know more.

Topics:

Summary of Column-Conversion Functions

This summary is organized according to the types of processing that can be performed with the Oracle GoldenGate functions.

These functions are used to perform tests.

Function	Description
@CASE	Selects a value depending on a series of value tests.
@EVAL	Selects a value based on a series of independent tests.
@IF	Selects one of two values depending on whether a conditional statement returns `TRUE` or `FALSE`.

These functions handle 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`.

These functions work 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.

These functions are used to perform arithmetic calculations.

Function	Description
@COMPUTE	Returns the result of an arithmetic expression.

These functions work with strings.

Function	Description
@NUMBIN	Converts a binary string into a number.
@NUMSTR	Converts a string into a number.
@STRCAT	Concatenates one or more strings.
@STRCMP	Compares two strings.
@STREXT	Extracts a portion of a string.
@STREQ	Determines whether or not two strings are equal.
@STRFIND	Finds the occurrence of a string within a string.
@STRLEN	Returns the length of a string.
@STRLTRIM	Trims leading spaces.
@STRNCAT	Concatenates one or more strings to a maximum length.
@STRNCMP	Compares two strings based on a specified number of characters.
@STRNUM	Converts a number into a string.
@STRRTRIM	Trims trailing spaces.
@STRSUB	Substitutes one string for another.
@STRTRIM	Trims leading and trailing spaces.
@STRUP	Changes a string to uppercase.
@VALONEOF	Compares a string or string column to a list of values.

These are miscellaneous functions.

Function	Description
@AFTER	Returns the after image of the specified column.
@BEFORE	Returns the before image of the specified column.
@BEFOREAFTER	Returns the before image of the specified column, if available, otherwise returns the after image.
@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.
@OGG_SHA1	Hashes some fields while replicating them to Operational Data Store.

@RANGE

Use the `@RANGE` function to divide the rows of any table across two or more Oracle GoldenGate processes. It can be used to increase the throughput of large and heavily accessed tables and also can be used to divide data into sets for distribution to different destinations. Specify each range in a `FILTER` clause in a `TABLE` or `MAP` statement.

`@RANGE` is safe and scalable. It preserves data integrity by guaranteeing that the same row will always be processed by the same process group. To ensure that rows do not shift partitions to another process group and that the DML is performed in the correct order, the column on which you base the `@RANGE` partitioning must not ever change during a process run. Updates to the partition column may result in "row not found" errors or unique-constraint errors.

`@RANGE` computes a hash value of the columns specified in the input. If no columns are specified, the `KEYCOLS` clause of the `TABLE` or `MAP` statement is used to determine the columns to hash, if a `KEYCOLS` clause exists. Otherwise, the primary key columns are used.

Oracle GoldenGate adjusts the total number of ranges to optimize the even distribution across the number of ranges specified.

Because any columns can be specified for this function, rows in tables with relational constraints to one another must be grouped together into the same process or trail to preserve referential integrity.

> **Note:**
>
> Using Extract to calculate the ranges is more efficient than using Replicat. Calculating ranges on the target side requires Replicat to read through the entire trail to find the data that meets each range specification.

Syntax

```
@RANGE (range, total_ranges [, column] [, column] [, ...])
```

range
The range assigned to the specified process or trail. Valid values are `1`, `2`, `3`, and so forth, with the maximum value being the value defined by *`total_ranges`*.

total_ranges
The total number of ranges allocated. For example, to divide data into three groups, use the value `3`.

column
The name of a column on which to base the range allocation. This argument is optional. If not used, Oracle GoldenGate allocates ranges based on the table's primary key.
Your data cannot contain missing or NULL columns; this will cause the `@RANGE` function to abend.
The columns specified in a list of columns must exist in the trail file. You can control this using `KEYCOLS` in the Extract to include this column, or by using `FETCHCOLS` in the Extract for the column, or by ensuring that the column is part of the supplemental log group and then using `LOGALLSUPCOLS`.

Examples

Example 1
In the following example, the replication workload is split into three ranges (between three Replicat processes) based on the `ID` column of the source `acct` table.
(Replicat group 1 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (1, 3, ID));
```

(Replicat group 2 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (2, 3, ID));
```

(Replicat group 3 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (3, 3, ID));
```

Example 2
In the following example, one Extract process splits the processing load into two trails. Since no columns were defined on which to base the range calculation, Oracle GoldenGate will use the primary key columns.

```
RMTTRAIL /ggs/dirdat/aa
TABLE fin.account, FILTER (@RANGE (1, 2));
RMTTRAIL /ggs/dirdat/bb
TABLE fin.account, FILTER (@RANGE (2, 2));
```

Example 3
In the following example, two tables have relative operations based on an `order_ID` column. The `order_master` table has a key of `order_ID`, and the `order_detail` table has a key of `order_ID` and `item_number`. Because the key `order_ID` establishes relativity, it is used in `@RANGE` filters for both tables to preserve referential integrity. The load is split into two ranges.
(Parameter file #1)

```
MAP sales.order_master, TARGET sales.order_master,
FILTER (@RANGE (1, 2, order_ID));
MAP sales.order_detail, TARGET sales.order_detail,
FILTER (@RANGE (1, 2, order_ID));
```

(Parameter file #2)

```
MAP sales.order_master, TARGET sales.order_master,
FILTER (@RANGE (2, 2, order_ID));
MAP sales.order_detail, TARGET sales.order_detail,
FILTER (@RANGE (2, 2, order_ID));
```

@AFTER

Use the `@AFTER` function to return the after image of the specified source column. This is the default behavior.

Syntax

```
@AFTER (column)
```

column
The name of the source column for which to return the after image.

Example

```
@AFTER (quantity)
```

@BEFORE

Use the `@BEFORE` function to return the before image of the specified source column.

When using this parameter, use the `GETUPDATEBEFORES` parameter in the Extract parameter file to capture before images from the transaction record, or use it in the Replicat parameter file to use the before image in a column mapping or filter. If using the Conflict Resolution and Detection (CDR) feature, you can use the `GETBEFORECOLS` option of `TABLE`. To use these parameters, the specified column must be present in the transaction log.

If the database only logs values for changed columns, make certain the required column values are available by enabling supplemental logging for those columns. Alternatively, you can use the `FETCHCOLS` or `FETCHCOLSEXCEPT` option of the `TABLE` parameter. The fetch option involves additional processing overhead.

Syntax

```
@BEFORE (column)
```

column
The name of the source column for which to return the before image.

Example

```
@BEFORE (quantity)
```

@BEFOREAFTER

Use the `@BEFOREAFTER` function to return the before image if available, or otherwise the after image.

When using this parameter, use the `GETUPDATEBEFORES` parameter in the Extract parameter file to capture before images from the transaction record, or use it in the Replicat parameter file to use the before image in a column mapping or filter. If using the Conflict Resolution and Detection (CDR) feature, you can use the `GETBEFORECOLS` option of `TABLE`. To use these parameters, all columns must be present in the transaction log.

If the database only logs values for changed columns, make certain the required column values are available by enabling supplemental logging for those columns. Alternatively, you can use the `FETCHCOLS` or `FETCHCOLSEXCEPT` option of the `TABLE` parameter. The fetch option involves additional processing overhead.

Syntax

```
@BEFOREAFTER (column)
```

column
The name of the source column for which to return the before image, if available, or otherwise the after image.

Example

```
@BEFOREAFTER (quantity)
```

@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)
```

column
The name of the target column to which the data will be copied.

Example

The following shows how `@BINARY` can be used to copy the data from the source column `ACCT_CREATE_DATE` to the target column `ACCT_COMPLAINT`.

```
ACCT_COMPLAINT =
@IF ( @NUMBIN (ACCT_CREATE_DATE ) < 48633, 'xxxxxx',
@BINARY (ACCT_COMPLAINT))
```

@BINTOBASE64

Use the `@BINTOBASE64` function to convert supplied binary data into BASE64 text.

Syntax

```
@BINTOBASE64 (data)
```

data
Can be one of the following:

- The name of the source column that contains the data
- An expression
- A literal string that is enclosed within single quote marks

Example

```
@BINTOBASE64('12345') converts to 'MTIzNDU='
```

@BINTOHEX

Use the `@BINTOHEX` function to convert supplied binary data into its hexadecimal equivalent.

Syntax

```
@BINTOHEX (data)
```

data
Can be one of the following:

- The name of the source column that contains the data
- An expression
- A literal string that is enclosed within single quote marks

Example

`@BINTOHEX ('12345')` converts to `3132333435`.

@CASE

Use the `@CASE` function to select a value depending on a series of value tests. There is no limit to the number of cases you can test with `@CASE`. If the number of cases is large, list the most frequently encountered conditions first for the best performance.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@CASE (value, test_value1, test_result1
[, test_value2, test_result2] [, ...]
[, default_result]
```

value
A value to test, for example, a column name. Enclose literals within single quote marks.

test_value
A valid result for *value*. Enclose literals within single quote marks.

test_result
A value to return based on the value of *test_value*. Enclose literals within single quote marks.

default_result
A default value to return if *value* results in none of the *test_value* values. Enclose literals within single quote marks.

Examples

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})
```

Examples

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

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

Example 2
The following `@IF` calculation uses `@COLSTAT` to return `NULL` to the target column if `PRICE` and `QUANTITY` are less than zero.

```
ORDER_TOTAL = PRICE * QUANTITY, @IF (PRICE < 0 AND QUANTITY < 0, @COLSTAT(NULL))
```

@COLTEST

Use the `@COLTEST` function to enable conditional calculations by testing for one or more column conditions. If a condition is satisfied, `@COLTEST` returns `TRUE`. To perform the conditional calculation, use the `@IF` function.

Syntax

```
@COLTEST (source_column, test_condition [, test_condition] [, ...])
```

source_column
The name of a source column.

test_condition
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.

Examples

Example 1
The following example uses `@IF` to map a value to the `HIGH_SALARY` column only if the `BASE_SALARY` column in the source record was both present (and not `NULL`) and greater than 250000. Otherwise, `NULL` is returned.

```
HIGH_SALARY =
@IF (@COLTEST (BASE_SALARY, PRESENT) AND
BASE_SALARY > 250000,
BASE_SALARY, @COLSTAT (NULL))
```

Example 2
In the following example, 0 is returned when the `AMT` column is missing or invalid; otherwise a value for `AMT` is returned.

```
AMOUNT = @IF (@COLTEST (AMT, MISSING, INVALID), 0, AMT)
```

@COMPUTE

Use the `@COMPUTE` function to return the value of an arithmetic expression to a target column. The value returned from the function is in the form of a string.

You can omit the `@COMPUTE` phrase when returning the value of an arithmetic expression to another Oracle GoldenGate function, as in:

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

The preceding returns the same result as:

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

Arithmetic expressions can be combinations of the following elements.

- Numbers
- The names of columns that contain numbers
- Functions that return numbers
- Arithmetic operators:

  + (plus)

  - (minus)

  * (multiply)

  / (divide)

  \ (remainder)
- Comparison operators:

  > (greater than)

  >= (greater than or equal)

  < (less than)

  <= (less than or equal)

  = (equal)

<> (not equal)

Results that are derived from comparisons can be zero (indicating `FALSE`) or non-zero (indicating `TRUE`).

- Parentheses (for grouping results in the expression)
- The conjunction operators `AND`, `OR`. Oracle GoldenGate only evaluates the necessary part of a conjunction expression. Once a statement is `FALSE`, the rest of the expression is ignored. This can be valuable when evaluating fields that may be missing or null. For example, if the value of `COL1` is `25` and the value of `COL2` is `10`, then the following are possible:

  `@COMPUTE (COL1 > 0 AND COL2 < 3)` returns `0`.

  `@COMPUTE (COL1 < 0 AND COL2 < 3)` returns `0`. `COL2 < 3` is never evaluated.

  `@COMPUTE ((COL1 + COL2)/5)` returns `7`.

Syntax

```
@COMPUTE (expression)
```

expression
A valid arithmetic expression. The numeric value plus the precision cannot be greater than 17 digits. If this limit is exceeded, `@COMPUTE` returns an error similar to the following.

```
2013-08-01 01:54:22  ERROR   OGG-01334  Error mapping data from column to column in
function COMPUTE.
```

Examples

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)
```

```
mod(id,10) = 1 MAP otest.tab,TARGET mtest.tab1,FILTER (@compute (id \ 10) = 1)
```

This example illustrates how to achieve the remainder SQL. Ensure that there is a space between `id \ 10` in the three characters, otherwise it will be reported as a filter syntax error.

@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_column
[, 'input_descriptor', source_column] [, ...])
```

'output_descriptor'
The output of the function. The valid value is a string that is composed of date descriptors and optional literal values, such as spaces or colons, that are required by the target column. Date descriptors can be strung together as needed. See Table 3-1 for descriptions of date descriptors. The format descriptor must match the `date/time/timestamp` format for the target. Oracle GoldenGate overrides the specified format to make it correct, if necessary.

'input_descriptor'
The source input. The valid value is a string that is composed of date descriptors and optional literal values, such as spaces or colons. Date descriptors can be strung together as needed. The following are examples:

- Descriptor string `'YYYYMMDD'` indicates that the source column specified with *source_column* contains (in order) a four-digit year (`YYYY`), month (`MM`), and day (`DD`).
- Descriptor string `'DD/MM/YY'` indicates that the source column specified with *source_column* contains the day, a slash, the month, a slash, and the two digit year.

See Table 3-1 for date descriptions.

source_column
The name of the numeric or character source column that supplies the input specified with *input_descriptor*.

Table 3-1 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 3-1 (Cont.) Date Descriptors

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. The unit is microseconds. On a Windows machine, the value will be padded with zeros (0) because the granularity of the Windows timestamp is milliseconds.	Input/Output
JUL	Julian day. JUL and JTS produce numbers you can use in numeric expressions.	Input/Output
TTS	NonStop 48-bit timestamp	Input
PHAMIS	PHAMIS application date format	Input
FFFFFF	Fraction (up to microseconds)	Input/Output
STRATUS	STRATUS application timestamp	Input/Output
CDATE	C timestamp in seconds since the Epoch	Input/Output

Examples

Example 1
In an instance where a two-digit year is supplied, but a four-digit year is required in the output, several options exist to obtain the correct century.

- The century can be hard coded, as in:

  ```
  'CC', 19 or 'CC', 20
  ```

- The @IF function can be used to set a condition, as in:

  ```
  'CC', @IF (YY > 70, 19, 20)
  ```

  This causes the century to be set to 19 when the year is greater than 70; otherwise the century is set to 20.

- The system can calculate the century automatically. If the year is less than 50, the system calculates a century of 20; otherwise, a century of 19 is calculated.

Example 2
The following converts year, month and day columns into a date.

```
date_col = @DATE ('YYYY-MM-DD', 'YY', date1_yy, 'MM', date1_mm, 'DD', date1_dd)
```

Example 3
The following converts a date and time, defaulting seconds to zero.

```
date_col = @DATE ('YYYY-MM-DD HH:MI:00', 'YYMMDD', date1, 'HHMI', time1)
```

Example 4
The following converts a numeric column stored as `YYYYMMDDHHMISS` to a SQL date.

```
datetime_col = @DATE ('YYYY-MM-DD HH:MI:SS', 'YYYYMMDDHHMISS', numeric_date)
```

Example 5
The following converts a numeric column stored as `YYYYMMDDHHMISS` to a Julian timestamp.

```
julian_ts_col = @DATE ('JTS', 'YYYYMMDDHHMISS', numeric_date)
```

Example 6
The following converts a Julian timestamp column to two separate columns: a datetime column in the format `YYYY-MM-DD HH:MI:SS` and a fraction column that holds the microseconds portion of the timestamp.

```
datetime_col = @DATE ('YYYY-MM-DD HH:MI:SS', 'JTS', jts_field), fraction_col = @DATE
('FFFFFF', 'JTS', jts_field)
```

Example 7
The following produces the time at which an order is filled. The inner `@DATE` expression changes the `order_taken` column into a Julian timestamp, then adds the `order_minutes` column converted into microseconds to this timestamp. The expression is passed back as a new Julian timestamp to the outer `@DATE` expression, which converts it back to a more readable date and time.

```
order_filled = @DATE ('YYYY-MM-DD HH:MI:SS', 'JTS', @DATE ('JTS',
'YYMMDDHHMISS', order_taken) + order_minutes * 60 * 1000000)
```

Example 8
The following does a full calculation of times. It goes from a source date column named `dt` to a target column named `dt5` that is to be converted to the date + 5 hours. The calculation also goes from a source timestamp column named `ts` to a target column named `ts5` that is to be converted to the timestamp + 5 hours.

```
MAP scratch.t4, TARGET scratch.t4_copy,
COLMAP ( USEDEFAULTS,
dt5 = @DATE ('YYYY-MM-DD HH:MI:SS', 'JTS',
@COMPUTE (@DATE ('JTS', 'YYYY-MM-DD HH:MI:SS', dt) + 18000000000 ) ),
ts5 = @DATE ('YYYY-MM-DD HH:MI:SS.FFFFFF', 'JTS',
@COMPUTE ( @DATE ('JTS', 'YYYY-MM-DD HH:MI:SS.FFFFFF', ts) + 18000000000 ) )
) ;
```

@DBFUNCTION

`@DBFUNCTION` is a column-conversion function introduced in Oracle GoldenGate 23ai. It provides a column mapping to a database function that executes within the database when exectuing a DML operation. This is helpful for applications that are tracking the database timestamp, time-sensitive operations, or ETL loads where the Apply Time of the DML operation within the database is needed. The database function must exist within the database and the Replicat user must have privileges to execute it. For Oracle database, this function is available for all Replicats. For non-Oracle databases, this function is available for Replicat in classic mode, parallel Replicat, and coordinated Replicat.

Limitations

- The database function used by `@DBFUNCTION` does not allow column as arguments. However, static arguments/constants are supported.

For example, the following argument is supported:

```
TO_CHAR(SYSTIMESTAMP, 'SSSSS.FF')
```

If you want to map this argument (`to_char`) with `@dbfunction`, it can be done as follows:

```
col1 = @dbfunction('TO_CHAR(SYSTIMESTAMP, ''SSSSS.FF'')')
```

If the string mapped to `@dbfunction` contains single quote `'`, then it needs to be written as `''` to be parsed correctly.

- `@DBFUNCTION` cannot be mapped to key columns.
- `@DBFUNCTION` cannot be used as an argument including inside:
  - `FILTER` or `WHERE` clause
  - `SQLEXEC`
  - Other column-conversion functions

Example

The following example shows the use of `@DBFUNCTION` to determine the system timestamp for the `ORDERS` table.

```
MAP OE.ORDERS, TARGET OE.ORDERS, COLMAP (USEDEFAULTS, TS =
@DBFUNCTION('SYSTIMESTAMP'))
```

@DATEDIFF

Use the `@DATEDIFF` function to calculate the difference between two dates or datetimes, in days or seconds.

Syntax

```
@DATEDIFF ('difference', 'date', 'date')
```

difference
The difference between the specified dates. Valid values can be:

- `DD`, which computes the difference in days.
- `SS`, which computes the difference in seconds.

date
A string within single quote marks, 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.

Examples

Example 1
The following calculates the number of days since the beginning of the year 2011.

```
YTD = @DATEDIFF ('DD', '2011-01-01', @DATENOW ())
```

Example 2
The following calculates the numerical day of the year. (`@DATEDIFF` returns 0 for `2011-01-01`):

```
todays_day = @COMPUTE (@DATEDIFF ('DD', '2011-01-01', @DATENOW ()) +1)
```

@DATENOW

Use the `@DATENOW` function to return the current date and time in the format `YYYY-MM-DD HH:MI:SS`. The date and time are returned in local time, including adjustments for Daylight Saving Time. `@DATENOW` takes no arguments.

Syntax

```
@DATENOW ()
```

@DDL

Use the `@DDL` function to return information about a DDL operation.

Syntax

```
@DDL ({TEXT | OPTYPE | OBJNAME | OBJTYPE | OBJOWNER})
```

OBJNAME
Returns the name of the object that is affected by the DDL.

OBJOWNER
Returns the name of the owner of the object that is affected by the DDL.

OBJTYPE
Returns the type of object that is affected by the DDL, such as `TABLE` or `INDEX`)

OPTYPE
Returns the operation type of the DDL, such as `CREATE` or `ALTER`.

TEXT
Returns the first 200 characters of the text of the DDL statement.

Example

The following example uses the output from `@DDL` in an `EVENTACTIONS` shell command.

```
DDL INCLUDE OBJNAME src.t* &
EVENTACTIONS (SHELL ('echo The DDL text is var1> out.txt ', &
VAR var1 = @DDL (TEXT)));
```

The redirected output file might contain a string like this:

```
The DDL text is CREATE TABLE src.test_tab (col1 int);
```

@EVAL

Use the `@EVAL` function to select a value based on a series of independent tests. There is no limit to the number of conditions you can test. If the number of cases is large, list the most frequently encountered conditions first for best performance.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@EVAL (condition, result
[condition, result] [, ....]
[, default_result])
```

condition
A conditional test using standard conditional operators. More than one condition can be specified.

result
A value or string to return based on the results of the conditional test. Enclose literals within single quote marks. Specify a result for each condition that is used.

default_result
A default result to return if none of the conditions is satisfied. A default result is optional.

NOT_SUPPORTED
In the following example, if the `AMOUNT` column is greater than 10000, a result of `high amount` is returned. If `AMOUNT` is greater than 5000 (and less than or equal to 10000), a result of `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')
```

NOT_SUPPORTED
The following is a modification of the preceding example. It returns the same results, except that a default value is specified, and a result of `lower` is returned if `AMOUNT` is less than or equal to 5000.

```
@EVAL (AMOUNT > 10000, 'high amount', AMOUNT > 5000, 'somewhat high', 'lower')
```

@GETENV

Use the `@GETENV` function to return information about the Oracle GoldenGate environment. You can use the information as input into the following:

- Stored procedures or queries (with `SQLEXEC`)
- Column maps (with the `COLMAP` option of `TABLE` or `MAP`)
- User tokens (defined with the `TOKENS` option of `TABLE` and mapped to target columns by means of the `@TOKEN` function)
- The `GET_ENV_VALUE` user exit function (see "GET_ENV_VALUE")

> **Note:**
>
> All syntax options must be enclosed within quotes as shown in the syntax descriptions.

- Retrieve the value of the DB_UNIQUE_NAME parameter of the source or the target databases, depending on which processes (Extract or Replicat) executes the function.

Syntax

```
@GETENV (
'LAG' , 'unit' |
'LASTERR' , 'error_info' |
'JULIANTIMESTAMP' |
'JULIANTIMESTAMP_PRECISE' |
'RECSOUTPUT' |
{'STATS'|'DELTASTATS'}, ['TABLE', 'table'], 'statistic' |
'GGENVIRONMENT', 'environment_info' |
'GGFILEHEADER', 'header_info'|
'GGHEADER', 'header_info' |
'RECORD', 'location_info' |
'DBENVIRONMENT', 'database_info,',
'TRANSACTION', 'transaction_info' |
'OSVARIABLE', 'variable' |
'TLFKEY', SYSKEY, unique_key
'USERNAME',
'OSUSERNAME',
'MACHINENAME',
'PROGRAMNAME',
'CLIENTIDENTIFIER',
'SOURCEDATABASEINFO'
)
```

'LAG' , 'unit'

Valid for Extract and Replicat.

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.

Syntax

```
@GETENV ('LAG', {'SEC'|'MSEC'|'MIN'})
```

'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.

'LASTERR' , 'error_info'

Valid for Replicat.

Use the LASTERR option of @GETENV to return information about the last failed operation processed by Replicat.

Syntax

```
@GETENV ('LASTERR', {'DBERRNUM'|'DBERRMSG'|'OPTYPE'|'ERRTYPE'})
```

'DBERRNUM'
Returns the database error number associated with the failed operation.

'DBERRMSG'
Returns the database error message associated with the failed operation.

'OPTYPE'
Returns the operation type that was attempted.

'ERRTYPE'
Returns the type of error. Possible results are:

- `DB` (for database errors)
- `MAP` (for errors in mapping)

'JULIANTIMESTAMP' | 'JULIANTIMESTAMP_PRECISE'

Valid for Extract and Replicat.

Use the `JULIANTIMESTAMP` option of `@GETENV` to return the current time in Julian format. The unit is microseconds (one millionth of a second). On a Windows machine, the value is padded with zeros (0) because the granularity of the Windows timestamp is milliseconds (one thousandth of a second). For example, the following is a typical column mapping:

```
MAP dbo.tab8451, Target targ.tabjts, COLMAP (USEDEFAULTS, &
JTSS = @GETENV ('JULIANTIMESTAMP')
JTSFFFFFF = @date ('yyyy-mm-dd hh:mi:ss.ffffff', 'JTS', &
@getenv ('JULIANTIMESTAMP') ) )
;
```

Possible values that the `JTSS` and `JTSFFFFFF` columns can have are:

```
212096320960773000 2010-12-17:16:42:40.773000
212096321536540000 2010-12-17:16:52:16.540000
212096322856385000 2010-12-17:17:14:16.385000
212096323062919000 2010-12-17:17:17:42.919000
212096380852787000 2010-12-18:09:20:52.787000
```

The last three digits (the microseconds) of the number all contain the padding of 0s .

Optionally, you can use the `'JULIANTIMESTAMP_PRECISE'` option to obtain a timestamp with high precision though this may effect performance.

> **Note:**
>
> Do not use these values for ordering operations. Instead use this value:
> `@COMPUTE(@COMPUTE(@NUMSTR(@GETENV ("RECORD", "FILESEQNO")*100000000000)+@NUMSTR(@GETENV ("RECORD", "FILERBA")))"`

Syntax

```
@GETENV ('JULIANTIMESTAMP')
@GETENV ('JULIANTIMESTAMP_PRECISE')
```

'RECSOUTPUT'

Valid for Extract.

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')
```

{'STATS'|'DELTASTATS'}, ['TABLE', '*table*'], '*statistic*'

Valid for Extract and Replicat.

Use the STATS and DELTASTATS options of @GETENV to return the number of operations that were processed per table for any or all of the following:

- INSERT operations
- UPDATE operations
- DELETE operations
- TRUNCATE operations
- Total DML operations
- Total DDL operations
- Number of conflicts that occurred, if the Conflict Detection and Resolution (CDR) feature is used.
- Number of CDR resolutions that succeeded
- Number of CDR resolutions that failed

Any errors in the processing of this function, such as an unresolved table entry or incorrect syntax, returns a zero (0) for the requested statistics value.

Understanding How Recurring Table Specifications Affect Operation Counts

An Extract that is processing the same source table to multiple output trails returns statistics based on each localized output trail to which the table linked to @GETENV is written. For example, if Extract captures 100 inserts for table ABC and writes table ABC to three trails, the result for the @GETENV is 300

```
EXTRACT ABC
...
EXTTRAIL c:\north\aa;
TABLE TEST.ABC;
EXTTRAIL c:\north\bb;
TABLE TEST.ABC;
TABLE EMI, TOKENS (TOKEN-CNT = @GETENV ('STATS', 'TABLE', 'ABC', 'DML'));
EXTTRAIL c:\north\cc;
TABLE TEST.ABC;
```

In the case of an Extract that writes a source table multiple times to a single output trail, or in the case of a Replicat that has multiple MAP statements for the same TARGET table, the statistics results are based on all matching TARGET entries. For example, if Replicat filters 20 rows for

REGION 'WEST,' 10 rows for REGION 'EAST,' 5 rows for REGION 'NORTH,' and 2 rows for REGION 'SOUTH' (all for table ABC) the result of the @GETENV is 37.

```
REPLICAT ABC
...
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'WEST'));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'EAST'));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'NORTH'));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'SOUTH'));
MAP TEST.EMI, TARGET TEST.EMI, &
    COLMAP (CNT = @GETENV ('STATS', 'TABLE', 'ABC', 'DML'));
```

Capturing Multiple Statistics

You can execute multiple instances of @GETENV to get counts for different operation types.

This example returns statistics only for INSERT and UPDATE operations:

```
REPLICAT TEST
..
..
MAP TEST.ABC, TARGET TEST.ABC, COLMAP (USEDEFAULTS, IU = @COMPUTE (@GETENV &
    ('STATS', 'TABLE', 'ABC', 'DML') - (@GETENV ('STATS', 'TABLE', &
    'ABC', 'DELETE'));
```

This example returns statistics for DDL and TRUNCATE operations:

```
REPLICAT TEST2
..
..
MAP TEST.ABC, TARGET TEST.ABC, COLMAP (USEDEFAULTS, DDL = @COMPUTE &
(@GETENV ('STATS', 'DDL') + (@GETENV ('STATS', 'TRUNCATE'));
```

Example Use Case

In the following use case, if all DML from the source is applied successfully to the target, Replicat suspends by means of EVENTACTIONS with SUSPEND, until resumed from GGSCI with SEND REPLICAT with RESUME.

GETENV used in Extract parameter file:

```
TABLE HR1.HR*;
TABLE HR1.STAT, TOKENS ('env_stats' = @GETENV ('STATS', 'TABLE', &
    'HR1.HR*', 'DML'));
```

GETENV used in Replicat parameter file:

```
MAP HR1.HR*, TARGET HR2.*;
MAP HR1.STAT, TARGET HR2.STAT, filter (
    @if (
    @token ('stats') =
    @getenv ('STATS', 'TABLE', 'TSSCAT.TCUSTORD', 'DML'), 1, 0 )
    ),
    eventactions (suspend);
```

Using Statistics in FILTER Clauses

Statistics returned by STATS and DELTASTATS are dynamic values and are incremented after mapping is performed. Therefore, when using CDR statistics in a FILTER clause in each of multiple MAP statements, you need to order the MAP statements in descending order of the statistics values. If the order is not correct, Oracle GoldenGate returns error OGG-01921. For detailed information about this requirement, see Document 1556241.1 in the Knowledge base of My Oracle Support at http://support.oracle.com.

Example 3-1 MAP statements containing statistics in FILTER clauses

In the following example, the MAP statements containing the filter for the CDR_CONFLICTS statistic are ordered in descending order of the statistic: >3, then =3, then <3.

```
MAP TEST.GG_HEARTBEAT_TABLE, TARGET TEST.GG_HEARTBEAT_TABLE COMPARECOLS (ON
UPDATE ALL),RESOLVECONFLICT(UPDATEROWEXISTS,(DEFAULT, OVERWRITE)),FILTER
(@GETENV ("STATS", "CDR_CONFLICTS") > 3),EVENTACTIONS (LOG INFO);MAP
TEST.GG_HEARTBEAT_TABLE, TARGET TEST.GG_HEARTBEAT_TABLE COMPARECOLS (ON
UPDATE ALL),RESOLVECONFLICT(UPDATEROWEXISTS,(DEFAULT, OVERWRITE)),FILTER
(@GETENV ("STATS", "CDR_CONFLICTS") = 3),EVENTACTIONS (LOG WARNING);MAP
TEST.GG_HEARTBEAT_TABLE, TARGET TEST.GG_HEARTBEAT_TABLE COMPARECOLS (ON
UPDATE ALL),RESOLVECONFLICT(UPDATEROWEXISTS,(DEFAULT, OVERWRITE)),FILTER
(@GETENV ("STATS", "CDR_CONFLICTS") < 3),EVENTACTIONS (LOG WARNING);
```

Syntax

```
@GETENV ({'STATS' | 'DELTASTATS'}, ['TABLE', 'table'], 'statistic')
```

{'STATS' | 'DELTASTATS'}
STATS returns counts since process startup, whereas DELTASTATS returns counts since the last execution of a DELTASTATS.
The execution logic is as follows:

- When Extract processes a transaction record that satisfies @GETENV with STATS or DELTASTATS, the table name is matched against resolved source tables in the TABLE statement.
- When Replicat processes a trail record that satisfies @GETENV with STATS or DELTASTATS, the table name is matched against resolved target tables in the TARGET clause of the MAP statement.

'TABLE', 'table'
Executes the STATS or DELTASTATS only for the specified table or tables. Without this option, counts are returned for all tables that are specified in TABLE (Extract) or MAP (Replicat) parameters in the parameter file.
Valid *table_name* values are:

- '*schema.table*' specifies a table.
- *'table'* specifies a table of the default schema.
- '*schema.**' specifies all tables of a schema.
- '*' specifies all tables of the default schema.

For example, the following counts DML operations only for tables in the `hr` schema:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = @GETENV ('STATS',
'TABLE', 'hr.*', 'DML'));
```

Likewise, the following counts DML operations only for the `emp` table in the `hr` schema:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = @GETENV ('STATS',
'TABLE', 'hr.emp', 'DML'));
```

By contrast, because there are no specific tables specified for `STATS` in the following example, the function counts all `INSERT`, `UPDATE`, and `DELETE` operations for all tables in all schemas that are represented in the `TARGET` clauses of `MAP` statements:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = &
@GETENV ('STATS', 'DML'));
```

'statistic'
The type of statistic to return. See Using Statistics in FILTER Clauses for important information when using statistics in `FILTER` clauses in multiple `TABLE` or `MAP` statements.

'INSERT'
Returns the number of `INSERT` operations that were processed.

'UPDATE'
Returns the number of `UPDATE` operations that were processed.

'DELETE'
Returns the number of `DELETE` operations that were processed.

'DML'
Returns the total of `INSERT`, `UPDATE`, and `DELETE` operations that were processed.

'TRUNCATE'
Returns the number of `TRUNCATE` operations that were processed. This variable returns a count only if Oracle GoldenGate DDL replication is not being used. If DDL replication is being used, this variable returns a zero.

'DDL'
Returns the number of DDL operations that were processed, including `TRUNCATE`s and DDL specified in `INCLUDE` and `EXCLUDE` clauses of the `DDL` parameter, all scopes (`MAPPED`, `UNMAPPED`, `OTHER`). This variable returns a count only if Oracle GoldenGate DDL replication is being used. This variable is not valid for `'DELTASTATS'`.

'CDR_CONFLICTS'
Returns the number of conflicts that Replicat detected when executing the Conflict Detection and Resolution (CDR) feature.
Example for a specific table:

```
@GETENV ('STATS','TABLE','HR.EMP','CDR_CONFLICTS')
```

Example for all tables processed by Replicat:

```
@GETENV ('STATS','CDR_CONFLICTS')
```

'CDR_RESOLUTIONS_SUCCEEDED'
Returns the number of conflicts that Replicat resolved when executing the Conflict Detection and Resolution (CDR) feature.
Example for a specific table:

```
@GETENV ('STATS','TABLE','HR.EMP', 'CDR_RESOLUTIONS_SUCCEEDED')
```

Example for all tables processed by Replicat:

```
@GETENV ('STATS','CDR_RESOLUTIONS_SUCCEEDED')
```

'CDR_RESOLUTIONS_FAILED'
Returns the number of conflicts that Replicat could not resolve when executing the Conflict Detection and Resolution (CDR) feature.
Example for a specific table:

```
@GETENV ('STATS','TABLE','HR.EMP', 'CDR_RESOLUTIONS_FAILED')
```

Example for all tables processed by Replicat:

```
@GETENV ('STATS','CDR_RESOLUTIONS_FAILED')
```

'GGENVIRONMENT' , '*environment_info*'

Valid for Extract and Replicat.

Use the `GGENVIRONMENT` option of `@GETENV` to return information about the Oracle GoldenGate environment.

Syntax

```
@GETENV ('GGENVIRONMENT', {'DOMAINNAME'|'GROUPDESCRIPTION'|'GROUPNAME'|
   'GROUPTYPE'|'HOSTNAME'|'OSUSERNAME'|'PROCESSID')
```

'DOMAINNAME'
(Windows only) Returns the domain name associated with the user that started the process.

'GROUPDESCRIPTION'
Returns the description of the group, taken from the checkpoint file. Requires that a description was provided with the `DESCRIPTION` parameter when the group was created with the `ADD` command.

'GROUPNAME'
Returns the name of the process group.

'GROUPTYPE'
Returns the type of process, either `EXTRACT` or `REPLICAT`.

'HOSTNAME'
Returns the name of the system running the Extract or Replicat process.

'OSUSERNAME'
Returns the operating system user name that started the process.

'PROCESSID'
Returns the process ID that is assigned to the process by the operating system.

'GGHEADER' , '*header_info*'

Valid for Extract and Replicat.

Use the `GGHEADER` option of `@GETENV` to return information from the header portion of an Oracle GoldenGate trail record. The header describes the transaction environment of the record. For more information on record headers and record types, see Trail Record Format.

Syntax

```
@GETENV ('GGHEADER', {'BEFOREAFTERINDICATOR'|'COMMITTIMESTAMP'|'LOGPOSITION'|
   'LOGRBA'|'OBJECTNAME'|'TABLENAME'|'OPTYPE'|'RECORDLENGTH'|
   'TRANSACTIONINDICATOR'})
```

> **Note:**
>
> Do not use `TIMESTAMP_PRECISE` for ordering operations. Instead use this value:
> `@COMPUTE(@COMPUTE(@NUMSTR(@GETENV ("RECORD", "FILESEQNO"))*100000000000)+@NUMSTR(@GETENV ("RECORD", "FILERBA")))`

'BEFOREAFTERINDICATOR'
Returns the before or after indicator showing whether the record is a before image or an after image. Possible results are:

- `BEFORE` (before image)
- `AFTER` (after image)

'COMMITTIMESTAMP'
Returns the transaction timestamp (the time when the transaction committed) expressed in the format of `YYYY-MM-DD HH:MI:SS.FFFFFF`, for example:

```
2011-01-24 17:08:59.000000
```

'LOGPOSITION'
Returns the position of the Extract process in the data source. (See the `LOGRBA` option.)

'LOGRBA'
`LOGRBA` and `LOGPOSITION` store details of the position in the data source of the record. For transactional log-based products, `LOGRBA` is the sequence number and `LOGPOSITION` is the relative byte address. However, these values will vary depending on the capture method and database type.

'OBJECTNAME' | 'TABLENAME'
Returns the table name or object name (if a non-table object).

'OPTYPE'
Returns the type of operation. Possible results are:

```
INSERT
UPDATE
DELETE
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.

'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.
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.)

'GGFILEHEADER' , '*header_info*'

Valid for Replicat only.

Use the `GGFILEHEADER` option of `@GETENV` to retrieve attributes of an Oracle GoldenGate Extract file or trail file. These attributes are stored as tokens in the file header.

> **Note:**
>
> If a given database, operating system, or Oracle GoldenGate version does not provide information that relates to a given token, a `NULL` value will be returned.

Syntax

```
@GETENV ('GGFILEHEADER', {'COMPATIBILITY'|'CHARSET'|'CREATETIMESTAMP'|
   'FILENAME'|'FILETYPE'|'FILESEQNO'|'FILESIZE'|'FIRSTRECCSN'|
   'LASTRECCSN'|'FIRSTRECIOTIME'|'LASTRECIOTIME'|'URI'|'URIHISTORY'|
   'GROUPNAME'|'DATASOURCE'|'GGMAJORVERSION'|'GGMINORVERSION'|
   'GGVERSIONSTRING'|'GGMAINTENANCELEVEL'|'GGBUGFIXLEVEL'|'GGBUILDNUMBER'|
   'HOSTNAME'|'OSVERSION'|'OSRELEASE'|'OSTYPE'|'HARDWARETYPE'|
   'DBNAME'|
'DBUNIQUENAME'|'DBINSTANCE'|'DBTYPE'|'DBCHARSET'|'DBMAJORVERSION'|
```

```
'DBMINORVERSION'|'DBVERSIONSTRING'|'DBCLIENTCHARSET'|'DBCLIENTVERSIONSTRING'|
   'LASTCOMPLETECSN'|'LASTCOMPLETEXIDS'|'LASTCSN'|'LASTXID'|
   'LASTCSNTS'|'RECOVERYMODE'})
```

'COMPATIBILITY'
Returns the compatibility level of the trail file. The compatibility level of the current Oracle GoldenGate version must be greater than, or equal to, the compatibility level of the trail file to be able to read the data records in that file. Current valid values are from 0 or 6.

- 1 means that the trail file is of Oracle GoldenGate version 10.0 or later, which supports file headers that contain file versioning information.
- 0 means that the trail file is of an Oracle GoldenGate version that is older than 10.0. File headers are not supported in those releases. The 0 value is used for backward compatibility to those Oracle GoldenGate versions.
- 5 means that the trail file is of Oracle GoldenGate version 12.2 or later.
- 6 means that the trail file is of Oracle GoldenGate version 12.3.0.1.

  This value keeps increasing as per the Oracle GoldenGate version depending on the trail file version.

'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 file system.

'FILETYPE'
Returns a numerical value 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. The valid values are:

- `0 - EXTFILE`
- `1 - EXTTRAIL`
- `2 - UNIFIED and EXTFILE`
- `3 - UNIFIED and EXTTRAIL`

'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.

'LASTRECCSN'
Returns the commit sequence number (CSN) of the last record in the trail file. Value is `NULL` until the trail file is completed.

'FIRSTRECIOTIME'
Returns the time that the first record was written to the trail file. Value is `NULL` until the trail file is completed.

'LASTRECIOTIME'
Returns the time that the last record was written to the trail file. Value is `NULL` until the trail file is completed.

'RECOVERYMODE'
Returns recovery information for internal Oracle GoldenGate use. It is usually set to `APPENDMODE`.

'URI'
Returns the universal resource identifier of the process that created the trail file, in the following format:

host_name:*dir*:[:*dir*][:*dir_n*]*group_name*

Where:

- `host_name` is the name of the server that hosts the process
- `dir` is a subdirectory of the Oracle GoldenGate installation path.
- `group_name` is the name of the process group that is linked with the process.

The following example shows where the trail was processed and by which process. This includes a history of previous runs.

```
sys1:home:oracle:v9.5:extora
```

'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.

'GROUPNAME'
Returns the name of the group that is associated with the Extract process that created the trail. The group name is the one that was supplied when the `ADD EXTRACT` command was issued.

'DATASOURCE'
Returns the data source that was read by the process as a number. The return value can be one of the following:

- `DS_EXTRACT_TRAILS`: The source was an Oracle GoldenGate extract file, populated with change data. The return value is `0`.
- `DS_DATABASE`: The source was a direct select from database table written to a trail, used for `SOURCEISTABLE`-driven initial load. The return value is `2`.
- `DS_TRAN_LOGS`: The source was the database transaction log. The return value is `3`.

- DS_INITIAL_DATA_LOAD: The source was a direct select from database tables for an initial load. The return value is 4.
- DS_VAM_EXTRACT: The source was a vendor access module (VAM). The return value is 5.
- DS_VAM_TWO_PHASE_COMMIT: The source was a VAM trail. The return value is 6.

'GGMAJORVERSION'
Returns the major version of the Extract process that created the trail, expressed as an integer. For example, if a version is 1.2.3, it returns 1.

'GGMINORVERSION'
Returns the minor version of the Extract process that created the trail, expressed as an integer. For example, if a version is 1.2.3, it returns 2.

'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.

'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:

- Version s10_69
- #1 SMP Fri Feb 24 16:56:28 EST 2006
- 5.00.2195 Service Pack 4

'OSRELEASE'
Returns the release version of the operating system of the machine where the Extract that wrote the trail is running. For example, release versions of the examples given for OSVERSION could be:

- 5.10
- 2.6.9-34.ELsmp

'OSTYPE'
Returns the type of operating system of the machine where the Extract that wrote the trail is running. For example:

- SunOS
- Linux
- Microsoft Windows

'HARDWARETYPE'
Returns the type of hardware of the machine where the Extract that wrote the trail is running. For example:

- sun4u
- x86_64
- x86

'DBNAME'
Returns the name of the database, for example findb.

'DBUNIQUENAME'
Returns the value of the DB_UNIQUE_NAME token as read from the header of the source trail file. Its value matches the DB_UNIQUE_NAME parameter of the source database.

'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
MSSQL
MYSQL
ORACLE
TERADATA
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.)

'LASTCOMPLETECSN'
Returns recovery information for internal Oracle GoldenGate use.

'LASTCOMPLETEXIDS'
Returns recovery information for internal Oracle GoldenGate use.

'LASTCSN'
Returns recovery information for internal Oracle GoldenGate use.

'LASTXID'
Returns recovery information for internal Oracle GoldenGate use.

'LASTCSNTS'
Returns recovery information for internal Oracle GoldenGate use.

'RECORD' , '*location_info*'

Valid for a data pump Extract or Replicat.

Use the `RECORD` option of `@GETENV` to return the location or Oracle rowid of a record in an Oracle GoldenGate trail file.

Syntax

```
@GETENV ('RECORD',
{'TIMESTAMP_PRECISE'|'FILESEQNO'|'FILERBA'|'ROWID'|'RSN'|'TIMESTAMP'})
```

'TIMESTAMP_PRECISE'
Valid for Extract or Replicat.
The `TIMESTAMP_PRECISE` option returns the timestamp from year to microseconds. However, depending on the database, the value can be in milliseconds with 0 microseconds.

'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.

'ROWID'
(Valid for Oracle) Returns the row id of the record.

'RSN'
Returns the record sequence number within the transaction. This value does not always generate uniquely increasing values and should not be used to order operations. For ordering transactions or DML operations within a transaction, use the information outlined in MOS DOC ID 1340823.1.

'TIMESTAMP'
Returns the timestamp of the record.

Example:

```
REC-TIMESTAMP: 2017-10-31 06:21:07 REC-TIMESTAMP-PRECISE: 2017-10-31
06:21:07.478064
```

'DBENVIRONMENT' , '*database_info*'

Valid for Extract and Replicat.

Use the DBENVIRONMENT option of @GETENV to return global environment information for a database.

Syntax

```
@GETENV ('DBENVIRONMENT',
{'DBNAME'|'DBUNIQUENAME'|'DBVERSION'|'DBUSER'|'SERVERNAME'})
```

'DBNAME'
Returns the database name.

'DBUNIQUENAME'
Returns the value of the DB_UNIQUE_NAME parameter of the database to which the process is connected. The source database in the case of Extract and the target database for Replicat. This value will be set only for Oracle databases.

'DBVERSION'
Returns the database version.

'DBUSER'
Returns the database login user. Note that SQL Server does not log the user ID.

'SERVERNAME'
Returns the name of the server.

'TRANSACTION' , '*transaction_info*'

Valid for Extract.

Use the TRANSACTION option of @GETENV to return information about a source transaction. This option is valid for the Extract process but not for pump Extract and Replicat.

Syntax

```
@GETENV ('TRANSACTION',
{'TIMESTAMP_PRECISE'|'TRANSACTIONID'|'XID'|'CSN'|'TIMESTAMP'|'NAME'|
   'USERNAME'|'PLANNAME' | 'LOGBSN' | 'REDOTHREAD' | 'PROGRAMNAME' |
'CLIENTIDENTIFIER' | 'MACHINENAME' | 'USERNAME')
```

> **Note:**
>
> Do not use TIMETSAMP_PRECISE or TIMESTAMP for ordering operations. Instead use this value: @COMPUTE(@COMPUTE(@NUMSTR(@GETENV ("RECORD", "FILESEQNO"))*100000000000)+@NUMSTR(@GETENV ("RECORD", "FILERBA")))

'TIMESTAMP_PRECISE'
This option is valid for Extract. Use the `TIMESTAMP_PRECISE` returns the timestamp from year to microseconds. However, depending on the database, the value can be in milliseconds with 0 microseconds

'TRANSACTIONID' | 'XID'
Returns the transaction ID number. Either `TRANSACTIONID` or `XID` can be used. 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.
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, see Commit Sequence Number (CSN).

'TIMESTAMP'
Returns the commit timestamp of the transaction.

'NAME'
Returns the transaction name, if available.

'USERNAME'
(Oracle) Returns the Oracle user name of the database user that committed the last transaction. This is not valid for pump Extract and/or Replicat.

'PLANNAME'
(DB2 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.

'LOGBSN'
Returns the begin sequence number (BSN) in the transaction log. The BSN is the native sequence number that identifies the beginning of the oldest uncommitted transaction that is held in Extract memory. For example, given an Oracle database, the BSN would be expressed as a system change number (SCN). The BSN corresponds to the current I/O checkpoint value of Extract. This value can be obtained from the trail by Replicat when `@GETENV ('TRANSACTION', 'LOGBSN')` is used. This value also can be obtained by using the `INFO REPLICAT` command with the `DETAIL` option. The purpose of obtaining the BSN from Replicat is to get a recovery point for Extract in the event that a system failure or file system corruption makes the Extract checkpoint file unusable.

'REDOTHREAD'
Returns the thread number of a RAC node extract; on non-RAC node extracts the value is always 1. For data pump and Replicat, the thread id used by Extract capture of a RAC node is returned; on non-RAC, `@GETENV()` returns an error. Logdump shows the token, `ORATHREADID`, in the token section if the transaction is captured by Extract on a RAC node.

'PROGRAMNAME'
Name of the program or application that started the transaction or session.

'CLIENTIDENTIFIER'
Value set by using `DBMS_SESSION_.set_identifier()`.

'MACHINENAME'
Name of the host, machine, or server where database is running

'USERNAME'
Database login user name.

Example:

```
DB2 zOS:
TRANS-TIMESTAMP:         2017-10-31 06:21:07
TRANS-TIMESTAMP-PRECISE: 2017-10-31 06:21:07.485792
```

'OSVARIABLE' , '*variable*'

Valid for Extract and Replicat.

Use the `OSVARIABLE` option of `@GETENV` to return the string value of a specified operating-system environment variable.

Syntax

```
@GETENV ('OSVARIABLE', 'variable')
```

'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.

'TLFKEY' , SYSKEY, '*unique_key*'

Valid for Extract and Replicat.

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).

Syntax

```
@GETENV ('TLFKEY', SYSKEY, unique_key)
```

SYSKEY, *unique_key*
The NonStop node number of the source TLF/PTLF file. Do not enclose this syntax element in quotes.
Example:

```
GETENV ('TLFKEY', SYSKEY, 27)
```

'SOURCEDATABASEINFO'

This option has the DBUNIQUENAME and DBNAME fields. The fields from SOURCEDATABASEINFO are different from the GGFILEHEADER fields. Firstly, their performance is better as compared to the fields from GGFILEHEADER, so using SOURCEDATABASEINFO is a better alternative for scenarios where performance is critical. Secondly, when the DBUNIQUENAME token is not available in the trail header, either because the trail file was generated by an older version of Oracle GoldenGate, or because the database is not Oracle, @GETENV will treat DBUNIQUENAME as a synonym of DBNAME. In this case, a warning message will be written to the report file, the first time a header without the token is read.

@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.

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)
```

name
The name of the stored procedure or query. When using SQLEXEC to execute the procedure or query, valid values are as follows:
For queries, use the logical name specified with the ID option of the SQLEXEC clause. ID is a required SQLEXEC argument for queries.
For 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.

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:

```
{procedure_name | logical_name}.parameter
```

Examples, Standard Syntax

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));
```

Examples, Alternate Syntax

Example 1
In the following example, `@GETVAL` is called implicitly for the phrase `proc1.p2` without the `@GETVAL` keyword.

```
MAP test.tab1, TARGET test.tab2,
SQLEXEC (SPNAME proc1, ID myproc, PARAMS (p1 = sourcecol1)),
COLMAP (targcol1 = proc1.p2);
```

Example 2
In the following example, the `@GETVAL` function is called implicitly for the phrase `lookup.desc_param` without the `@GETVAL` keyword.

```
MAP sales.account, TARGET sales.newacct,
SQLEXEC (ID lookup,
QUERY ' select desc_col into desc_param from lookup_table '
' where code_col = :code_param ',
PARAMS (code_param = account_code)),
COLMAP (newacct_id = account_id, newacct_val = lookup.desc_param);
```

@HEXTOBIN

Use the `@HEXTOBIN` function to convert a supplied string of hexadecimal data into raw format.

Syntax

```
@HEXTOBIN (data)
```

data
The name of the source column, an expression, or a literal string that is enclosed within double quote marks.

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])
```

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}`

Function statement	Result
group_level = @LOWVAL (3)	{0x00, 0x00, 0x00}

@IF

Use the @IF function to return one of two values, based on a condition. You can use the @IF function with other Oracle GoldenGate functions to begin a conditional argument that tests for one or more exception conditions. You can direct processing based on the results of the test. You can nest @IF statements, if needed.

Syntax

```
@IF (condition, value_if_non-zero, value_if-zero)
```

condition
A valid conditional expression or Oracle GoldenGate function. Use numeric operators (such as =, > or <) only for numeric comparisons. For character comparisons, use one of the character-comparison functions.

value_if_non-zero
Non-zero is considered true.

value_if_zero
Zero (0) is considered false.

Examples

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))
```

Example 4
The following example demonstrates a nested @IF statement. In the example, if the QUANTITY is more than 10, then the item price is 90% of thePRICE.

```
 ORDER_TOTAL = @IF (PRICE > 0 AND QUANTITY > 0, @IF (QUANTITY > 10, (PRICE * 0.9) *
QUANTITY, PRICE * QUANTITY), @COLSTAT(NULL))
```

> **Note:**
>
> When enclosed in parenthesis (), Oracle GoldenGate column mapping function expects numeric results. The column value must be specified using single quotes.

@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)
```

source_column
The name of the source column that contains the string to be converted.

Example

The following combines `@NUMBIN` and `@DATE` to transform a 48-bit 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)
```

input
Can be either of the following:

- The name of a character column.
- A literal string that is enclosed within single quote marks.

Example

```
PAGE_NUM = @NUMSTR (ALPHA_PAGE_NO)
```

@OGG_SHA1

Use the `OGG_SHA1` function to return the SHA-1 160 bit / 20 bytes hash value.

Syntax

```
OGG_SHA1(expression)
```

expression
The name of a column, literal string, other column mapping function.

Example

```
OGG_SHA1(col_name)
```

@STRCAT

Use the `@STRCAT` function to concatenate one or more strings or string (character) columns. Enclose literal strings within single quote marks.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRCAT (string1, string2 [, ...])
```

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 single quote marks.

`@STRCMP` returns the following:

- `-1` if the first string is less than the second.
- `0` if the strings are equal.
- `1` if the first string is greater than the second.

Trailing spaces are truncated before comparing the strings.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STRCMP (string1, string2)
```

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.

```
@STRCMP ('JOHNSON', 'JONES')
```

@STRCMPNULL

Use the `@STRCMPNULL` in the same way as `@STRCMP` function to compare two character columns or literal strings, but if the arguments are `NULL`, the result value is `0` instead of `NULL`.

Syntax

```
@STRCMPNULL (string1, string2)
```

string1
The first column or literal string to be compared.

string2
The second column or literal string to be compared.

@STREQ

Use the `@STREQ` function to determine whether or not two string (character) columns or literal strings are equal. Enclose literals within single quote marks. `@STREQ` returns the following:

- `1` (true) if the strings are equal.
- `0` (false) if the strings are not equal.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems.

Trailing spaces are truncated before comparing the strings.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STREQ (string1, string2)
```

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')
```

@STREQNULL

Use the `@STREQNULL` function in the same way as `@STREQ` to determine whether or not two string (character) columns or literal strings are equal. However, if the two arguments passed to the function are `NULL`, then the return value is 1.

Syntax

```
@STREQNULL (string1, string2)
```

string1
The first column or literal string to be compared.

string2
The second column or literal string to be compared.

@STREXT

Use the `@STREXT` function to extract a portion of a string.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STREXT (string, begin_position, end_position)
```

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 single quote marks.

begin_position
The character position at which to begin extraction.

end_position
The character position at which to end extraction. The end position is included in the extraction.

Example

The following example uses three `@STREXT` functions to extract a phone number into three different columns.

```
AREA_CODE = @STREXT (PHONE, 1, 3),
PREFIX = @STREXT (PHONE, 4, 6),
PHONE_NO = @STREXT (PHONE, 7, 10)
```

@STRFIND

Use the `@STRFIND` function to determine the position of a string within a string column or else return zero if the string is not found. Optionally, `@STRFIND` can accept a starting position within the string.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STRFIND (string, 'search_string' [,  begin_position])
```

string
The string in which to search. This can be either the name of a character column or a literal string that is within single quote marks.

'search_string'
The string for which to search. Enclose the search string within single quote marks.

begin_position
The byte 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 byte)

@STRLEN

Use the `@STRLEN` function to return the length of a string, expressed as the number of characters.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STRLEN (string)
```

string
The name of a string (character) column or a literal string. Enclose literals within single quote marks.

Examples

```
@STRLEN (ID_NO)

@STRLEN ('abcd')
```

@STRLTRIM

Use the `@STRLTRIM` function to trim leading spaces.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRLTRIM (string)
```

string
The name of a character column or a literal string that is enclosed within single quote marks.

Example

```
birth_state = @strltrim (state)
```

@STRNCAT

Use the `@STRNCAT` function to concatenate one or more strings to a maximum length.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STRNCAT (string, max_length [, string, max_length] [, ...] )
```

string
The name of a string (character) column or a literal string that is enclosed within single quote marks.

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 bytes. The string can be either the name of a string (character) column or a literal string that is enclosed within single quote marks. The comparison starts at the first byte in the string.

`@STRNCMP` returns the following:

- `-1` if the first string is less than the second.
- `0` if the strings are equal.
- `1` if the first string is greater than the second.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STRNCMP (string1, string2, max_length)
```

string1
The first string to be compared.

string2
The second string to be compared.

max_length
The maximum number of bytes in the string to compare.

Example

The following example compares the first two bytes 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] )
```

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, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft

Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRRTRIM (string)
```

string
The name of a character column or a literal string that is enclosed within single quote marks.

Example

```
street_address = @strrtrim (address)
```

@STRSUB

Use the `@STRSUB` function to substitute strings within a string (character) column or constant. Enclose literal strings within single quote marks.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Any single byte code value 1 to 255 can be used in hexadecimal or octal format for the string arguments. Hex values A to F are case insensitive and the leading '`x`' must be lower case. Value zero (0) (`\x00` and `\000`) is not allowed because it is a string terminator.

No UNICODE values are supported.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STRSUB
(source_string, search_string, substitute_string
[, search_string, substitute_string] [, ...])
```

For multibyte characters, `STRSUB()` must be used as follows:

```
FIND_CATEGORY = @STRSUB (FIND_CATEGORY, '\u00A0', ''),
```

source_string
A source string, within single quotes, or the name of a source column that contains the characters for which substitution is to occur.

search_string
The string, within single quotes, for which substitution is to occur.

substitute_string
The string, within single quotes, that will be substituted for the search string.

Examples

The following returns `xxABCxx`.

```
@STRSUB ('123ABC123', '123', 'xx')
```

The following returns `023zBC023`.

```
@STRSUB ('123ABC123', 'A', 'z', '1', '0')
```

The following is an example of replacing `^Z`, using a hexadecimal string argument, with a space.

```
@strsub (col1,'\x1A',' '));
```

@STRTRIM

Use the `@STRTRIM` function to trim leading and trailing spaces.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

Syntax

```
@STRTRIM (string)
```

string
The name of a character column or a literal string that is enclosed within single quote marks.

Example

```
pin_no = @strtrim (custpin)
```

@STRUP

Use the `@STRUP` function to change an alphanumeric string or string (character) column to upper case.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@STRUP (string)
```

string
The name of a character column or a literal string that is enclosed within single quote marks.

Example

The following returns `SALESPERSON`.

```
@STRUP ('salesperson')
```

@TOKEN

Use the `@TOKEN` function to retrieve token data that is stored in the user token area of the Oracle GoldenGate record header. You can map token data to a target column by using `@TOKEN` in the source expression of a `COLMAP` clause. As an alternative, you can use `@TOKEN` within a `SQLEXEC` statement, an Oracle GoldenGate macro, or a user exit.

To define token data, use the `TOKENS` clause of the `TABLE` parameter in the Extract parameter file.

Syntax

```
@TOKEN ('token')
```

'token'
The name, enclosed within single quote marks, 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. This function trims trailing spaces before the comparison.

For this function, Oracle GoldenGate supports the use of an escape sequence to represent characters in a string column in Unicode or in the native character encoding of the Microsoft Windows, UNIX, and Linux operating systems. The target column must be a SQL Unicode data type if any argument is supplied as Unicode.

This function does not support `NCHAR` or `NVARCHAR` data types.

Syntax

```
@VALONEOF (expression, value [, value] [, ...])
```

expression
The name of a character column or a literal enclosed within single quote marks.

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')
```

4

User Exit Functions

This chapter describes the Oracle GoldenGate user exit functions and their syntax. For more information about using Oracle GoldenGate user exits, see Using User Exits to Extend Oracle GoldenGate Capabilities.

Topics:

Summary of User Exit Functions

If you use `CUSEREXITS`, the `LD_LIBRARY_PATH` environment variable needs to be extended for Microservices Architecture. With `CUSEREXITS`, you create shared objects (`*.so`, `*.ddl`), which are picked up only if the files are in the path.

It is recommended that you do not use (the default) `$OGG_HOME/lib` directory for the shared objects as the software location should be managed as Read-Only.

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 Oracle GoldenGate context information, and they modify the contents of data records.

Calling a User Exit

Write the user exit routine in C programming code. Use the `CUSEREXIT` parameter to call the user exit from a Windows DLL or UNIX shared object at a defined exit point within Oracle GoldenGate processing. Your user exit routine must be able to accept different events and information from the Extract and Replicat processes, process the information as desired, and return a response and information to the caller (the Oracle GoldenGate process that called it). For more information and syntax for the `CUSEREXIT` parameter, see "CUSEREXIT".

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 4-1 User Exit Calls

Call type	Processing point
EXIT_CALL_ABORT_TRANS	Valid when the RECOVERYOPTIONS mode is APPEND (the default). Called when a data pump or Replicat reads a RESTART ABEND record from the trail, placed there by a writer process that abended. (The writer process can be the primary Extract writing to a local trail read by a data pump, or a data pump writing to a remote trail read by Replicat.) This call type enables the user exit to abort or discard the transaction that was left incomplete when the writer process stopped, and then to recover and resume processing at the start of the previous completed transaction.
EXIT_CALL_BEGIN_TRANS	Called just before either of the following: • a BEGIN record of a transaction that is read by a data pump • the start of a Replicat transaction
EXIT_CALL_CHECKPOINT	Called just before an Extract or Replicat checkpoint is written.
EXIT_CALL_DISCARD_ASCII_RECORD	Called during Extract processing before an ASCII input record is written to the discard file. The associated ASCII buffer can be retrieved and manipulated by the user exit using callback routines. This call type is not applicable for use with the Replicat process.
EXIT_CALL_DISCARD_RECORD	Called during Replicat processing before a record is written to the discard file. Records can be discarded for several reasons, such as when a value in the Oracle GoldenGate change record is different from the current version in the target table.The associated discard buffer can be retrieved and manipulated by the user exit using callback routines. This call type is not applicable for use with the Extract process.
EXIT_CALL_END_TRANS	Called just after either of the following: • an END record of a transaction that is read by a data pump • the last record in a Replicat transaction
EXIT_CALL_FATAL_ERROR	Called during Extract or Replicat processing just before Oracle GoldenGate terminates after a fatal error.
EXIT_CALL_PROCESS_MARKER	Called during Replicat processing when a marker from a NonStop server is read from the trail, and before writing to the marker history file.
EXIT_CALL_PROCESS_RECORD	• For Extract, called before a record buffer is output to the trail. • For Replicat, called just before a replicated operation is performed. This call is the basis of most user exit processing. When EXIT_CALL_PROCESS_RECORD is called, the record buffer and other record information are available to the user exit through callback routines. If source-target mapping is specified in the parameter file, the mapping is performed before the EXIT_CALL_PROCESS_RECORD event takes place. The user exit can map, transform, clean, or perform virtually any other operation with the record. The user exit can return a status indicating whether the caller should process or ignore the record.
EXIT_CALL_START	Called at the start of processing. The user exit can perform initialization work, such as opening files and initializing variables.
EXIT_CALL_STOP	Called before the process stops gracefully or ends abnormally. The user exit can perform completion work, such as closing files or outputting totals.
EXIT_CALL_RESULT	Set by the user exit routines to instruct the caller how to respond when each exit call completes.

Using EXIT_CALL_RESULT

Use EXIT_CALL_RESULT to provide a response to the routine.

Table 4-2 User Exit Responses

Call result	Description
EXIT_ABEND_VAL	Instructs the caller to terminate immediately.
EXIT_IGNORE_VAL	Rejects records for further processing. EXIT_IGNORE_VAL is appropriate when the user exit performs all the required processing for a record and there is no need to output or replicate the data record.
EXIT_OK_VAL	If the routine does nothing to respond to an event, EXIT_OK_VAL is assumed. If the exit call type is any of the following... • EXIT_CALL_PROCESS_RECORD • EXIT_CALL_DISCARD_RECORD • EXIT_CALL_DISCARD_ASCII_RECORD ... and EXIT_OK_VAL is returned, then Oracle GoldenGate processes the record buffer that was returned by the user exit.
EXIT_PROCESSED_REC_VAL	Instructs Extract or Replicat to skip the record, but update the statistics that are printed to the report file for that table and for that operation type.
EXIT_STOP_VAL	Instructs the caller to stop processing gracefully. EXIT_STOP_VAL or EXIT_ABEND_VAL may be appropriate when an error condition occurs in the user exit.

Using EXIT_PARAMS

Use EXIT_PARAMS to supply information to the user exit routine, such as the program name and user-defined parameters. You can process a single data record multiple times.

Table 4-3 User Exit Input

Exit parameter	Description
PROGRAM_NAME	Specifies the full path and name of the calling process, for example \ggs\extract or \ggs\replicat. Use this parameter when loading an Oracle GoldenGate callback routine using the Windows API or to identify the calling program when user exits are used with both Extract and Replicat processing.
FUNCTION_PARAM	• Allows you to pass a parameter that is a literal string to the user exit. Specify the parameter with the EXITPARAM option of the TABLE or MAP statement from which the parameter will be passed. See "EXITPARAM *'parameter'*". This is only valid during the exit call to process a specific record. • 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 "CUSEREXIT".) 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. To request the same record again, set MORE_RECS_IND to CHAR_NO_VAL or CHAR_YES_VAL.

Using ERCALLBACK

ERCALLBACK is the basic user exit function for Oracle GoldenGate. It is used to pull the record context into user exit. It's like a package that contains multiple individual functions inside it. You can call these functions and get return values. For example, functions such as GET_BEFORE_AFTER_IND, or GET_COLUMN_VALUE_FROM_NAME can be called. These functions are called function_code.

Syntax

```
ERCALLBACK (function_code, buffer, result_code );
```

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 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 4-4.

Table 4-4 Result Codes

Code	Description
EXIT_FN_RET_BAD_COLUMN_DATA	Invalid data was encountered when retrieving or setting column data.
EXIT_FN_RET_BAD_DATE_TIME	A date, timestamp, or interval type of column contains an invalid date or time value.
EXIT_FN_RET_BAD_NUMERIC_VALUE	A numeric type of column contains an invalid numeric value.
EXIT_FN_RET_COLUMN_NOT_FOUND	The column was not found in a compressed update record (update by a database that only logs the values that were changed).
EXIT_FN_RET_ENV_NOT_FOUND	The specified environment value could not be found in the record.
EXIT_FN_RET_EXCEEDED_MAX_LENGTH	The metadata could not be retrieved because the name of the table or column did not fit in the allocated buffer.
EXIT_FN_RET_FETCH_ERROR	The record could not be fetched. View the error message to see the reason.
EXIT_FN_RET_INCOMPLETE_DDL_REC	An internal error occurred when processing the DDL record. The record is probably incomplete.
EXIT_FN_RET_INVALID_CALLBACK_FNC_CD	An invalid callback function code was passed to the callback routine.
EXIT_FN_RET_INVALID_COLUMN	A non-existent column was referred to in the function call.
EXIT_FN_RET_INVALID_COLUMN_TYPE	The routine is trying to manipulate a data type that is not supported by Oracle GoldenGate for that purpose.
EXIT_FN_RET_INVALID_CONTEXT	The callback function was called at an improper time.
EXIT_FN_RET_INVALID_PARAM	An invalid parameter was passed to the callback function.

Table 4-4 (Cont.) Result Codes

Code	Description
EXIT_FN_RET_NO_SRCDB_INSTANCE	The source database instance could not be found.
EXIT_FN_RET_NO_TGTDB_INSTANCE	The target database instance could not be found.
EXIT_FN_RET_NOT_SUPPORTED	This function is not supported for this process.
EXIT_FN_RET_OK	The callback function succeeded.
EXIT_FN_RET_SESSION_CS_CNV_ERR	A ULIB_ERR_INVALID_CHAR_FOUND error was returned to the character-set conversion routine. The conversion failed.
EXIT_FN_RET_TABLE_NOT_FOUND	An invalid table name was specified.
EXIT_FN_RET_TOKEN_NOT_FOUND	The specified user token could not be found in the record.

You can use ERCALLBACK to perform many different function calls. For example, if you want to get the name of a table, you can use the following command:

```
ERCALLBACK (GET_TABLE_NAME, &var, &result_code)
```

These functions are used inside the c code for the user exit to perform any of the calls for functions provided in the section Function Codes. With the combination of the different function_code calls, you can perform many tasks using ERCALLBACK, such as:

- Recreate DML statements
- Perform transformations
- Pull specific columns out of a record
- Write information to a report file

For example, if you need a message written to the report file each time the lag in the heartbeat table exceeds a certain threshold, you could use the CUSEREXIT function. The CUSEREXIT function would then make numerous calls to ERCALLBACK to get the lag column data, perform calculations and the comparison, and if the lag is over the specified threshold then write a message to the report file.

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 4-5 Summary of Oracle GoldenGate Function Codes

Function code	Description
COMPRESS_RECORD	Use the COMPRESS_RECORD function when some, but not all, of a target table's columns are present after mapping and the entire record must be manipulated, rather than individual column values.
DECOMPRESS_RECORD	Use the DECOMPRESS_RECORD function when some, but not all, of a target table's columns are present after mapping and the entire record must be manipulated, rather than individual column values.

Table 4-5 (Cont.) Summary of Oracle GoldenGate Function Codes

Function code	Description
GET_BASE_OBJECT_NAME	Use the GET_BASE_OBJECT_NAME function to retrieve the fully qualified name of the base object of an object in a record.
GET_BASE_OBJECT_NAME_ONLY	Use the GET_BASE_OBJECT_NAME_ONLY function to retrieve only the name of the base object of an object in a record.
GET_BASE_SCHEMA_NAME_ONLY	Use the GET_BASE_SCHEMA_NAME_ONLY function to retrieve only the name of the schema of the base object of an object in a record.
GET_BEFORE_AFTER_IND	Use the GET_BEFORE_AFTER_IND function to determine whether a record is a before image or an after image of the database operation.
GET_CATALOG_NAME_ONLY	Use the GET_CATALOG_NAME_ONLY function to return the name of the database catalog.
GET_COL_METADATA_FROM_INDEX	Use the GET_COL_METADATA_FROM_INDEX function to determine the column metadata that is associated with a specified column index.
GET_COL_METADATA_FROM_NAME	Use the GET_COL_METADATA_FROM_NAME function to determine the column metadata that is associated with a specified column name.
GET_COLUMN_INDEX_FROM_NAME	Use the GET_COLUMN_INDEX_FROM_NAME function to determine the column index associated with a specified column name.
GET_COLUMN_NAME_FROM_INDEX	Use the GET_COLUMN_NAME_FROM_INDEX function to determine the column name associated with a specified column index.
GET_COLUMN_VALUE_FROM_INDEX	Use the GET_COLUMN_VALUE_FROM_INDEX function to return the column value from the data record using the specified column index.
GET_COLUMN_VALUE_FROM_NAME	Use the GET_COLUMN_VALUE_FROM_NAME function to return the column value from the data record by using the specified column name.
GET_DATABASE_METADATA	Use the GET_DATABASE_METADATA function to return database metadata.
GET_DDL_RECORD_PROPERTIES	Use the GET_DDL_RECORD_PROPERTIES function to return information about a DDL operation.
GET_ENV_VALUE	Use the GET_ENV_VALUE function to return information about the Oracle GoldenGate environment.
GET_ERROR_INFO	Use the GET_ERROR_INFO function to return error information associated with a discard record.
GET_GMT_TIMESTAMP	Use the GET_GMT_TIMESTAMP function to return the operation commit timestamp in GMT format.
GET_MARKER_INFO	Use the GET_MARKER_INFO function to return marker information when posting data. Use markers to trigger custom processing within a user exit.
GET_OBJECT_NAME	Returns the fully qualified two- or three-part name of a table or other object that is associated with the record that is being processed.
GET_OBJECT_NAME_ONLY	Returns the unqualified name of a table or other object that is associated with the record that is being processed.
GET_OPERATION_TYPE	Use the GET_OPERATION_TYPE function to determine the operation type associated with a record.
GET_POSITION	Use the GET_POSITION function is obtain a read position of an Extract data pump or Replicat in the Oracle GoldenGate trail.
GET_RECORD_BUFFER	Use the GET_RECORD_BUFFER function to obtain information for custom column conversions.

Table 4-5 (Cont.) Summary of Oracle GoldenGate Function Codes

Function code	Description
GET_RECORD_LENGTH	Use the GET_RECORD_LENGTH function to return the length of the data record.
GET_RECORD_TYPE	Use the GET_RECORD_TYPE function to return the type of record being processed
GET_SCHEMA_NAME_ONLY	Use the GET_SCHEMA_NAME_ONLY function to return only the schema name of a table.
GET_SESSION_CHARSET	Use the GET_SESSION_CHARSET function to return the character set of the user exit session.
GET_STATISTICS	Use the GET_STATISTICS function to return the current processing statistics for the Extract or Replicat process.
GET_TABLE_COLUMN_COUNT	Use the GET_TABLE_COLUMN_COUNT function to return the total number of columns in a table.
GET_TABLE_METADATA	Use the GET_TABLE_METADATA function to return metadata for the table that associated with the record that is being processed.
GET_TABLE_NAME	Use the GET_TABLE_NAME function to return the fully qualified two- or three-part name of the source or target table that is associated with the record that is being processed.
GET_TABLE_NAME_ONLY	Use the GET_TABLE_NAME_ONLY function to return only the unqualified name of the table that is associated with the record that is being processed.
GET_TIMESTAMP	Use the GET_TIMESTAMP function to return the I/O timestamp associated with a source data record.
GET_TRANSACTION_IND	Use the GET_TRANSACTION_IND function to determine whether a data record is the first, last or middle operation in a transaction,
GET_USER_TOKEN_VALUE	Use the GET_USER_TOKEN_VALUE function to obtain the value of a user token from a trail record.
OUTPUT_MESSAGE_TO_REPORT	Use the OUTPUT_MESSAGE_TO_REPORT function to output a message to the report file.
RESET_USEREXIT_STATS	Use the RESET_USEREXIT_STATS function to reset the statistics for the Oracle GoldenGate process.
SET_COLUMN_VALUE_BY_INDEX	Use the SET_COLUMN_VALUE_BY_INDEX function to modify a single column value without manipulating the entire data record.
@STRNCMP	Use the SET_COLUMN_VALUE_BY_NAME function to modify a single column value without manipulating the entire data record.
SET_OPERATION_TYPE	Use the SET_OPERATION_TYPE function to change the operation type associated with a data record.
SET_RECORD_BUFFER	Use the SET_RECORD_BUFFER function for compatibility with HP NonStop user exits, and for complex data record manipulation.
SET_SESSION_CHARSET	Use the SET_SESSION_CHARSET function to set the character set of the user exit session.
SET_TABLE_NAME	Use the SET_TABLE_NAME function to change the table name associated with a data record.

COMPRESS_RECORD

Valid For

Extract and Replicat

Description

Use the `COMPRESS_RECORD` function to re-compress records that have been decompressed with the `DECOMPRESS_RECORD` function. Call `COMPRESS_RECORD` only *after* using `DECOMPRESS_RECORD`.

The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.

Syntax

```
#include "usrdecs.h"
short result_code;
compressed_rec_def compressed_rec;
ERCALLBACK (COMPRESS_RECORD, &compressed_rec, &result_code);
```

Buffer

```
typedef struct
{
char *compressed_rec;
long compressed_len;
char *decompressed_rec;
long decompressed_len;
short *columns_present;
short source_or_target;
char requesting_before_after_ind;
} compressed_rec_def;
```

Input

`decompressed_rec`
A pointer to the buffer containing the record before compression. The record is assumed to be in the default Oracle GoldenGate canonical format.

`decompressed_len`
The length of the decompressed record.

`source_or_target`
One of the following to indicate whether the source or target record is being compressed.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

`requesting_before_after_ind`
Used as internal input. Does not need to be set. If set, it will be ignored.

`columns_present`
An array of values that indicates the columns present in the compressed record. For example, if the first, third and sixth columns exist in the compressed record, and the total number of columns in the table is seven, the array should contain:

```
1, 0, 1, 0, 0, 1, 0
```

Use the GET_TABLE_COLUMN_COUNT function to get the number of columns in the table (see "GET_TABLE_COLUMN_COUNT").

Output

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

Description

Use the DECOMPRESS_RECORD function when you want to retrieve or manipulate an entire update record with the GET_RECORD_BUFFER (see "GET_RECORD_BUFFER") or SET_RECORD_BUFFER function (see "SET_RECORD_BUFFER") 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 "COMPRESS_RECORD") to re-compress the record before returning it to the Oracle GoldenGate process.

This function is valid for processing UPDATE operations only. Deletes, inserts and updates appear in the buffer as full record images.

The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.

Compressed Record Format

Compressed SQL updates have the following format:

```
index length value [index length value ][...]
```

where:

- *index* is a two-byte index into the list of columns of the table (first column is zero).
- *length* is the two-byte length of the table.

- *value* is the actual column value, including one of the following two-byte null indicators when applicable. 0 is not null. -1 is null.

Syntax

```
#include "usrdecs.h"
short result_code;
compressed_rec_def compressed_rec;
ERCALLBACK (DECOMPRESS_RECORD, &compressed_rec, &result_code);
```

Buffer

```
typedef struct
{
char *compressed_rec;
long compressed_len;
char *decompressed_rec;
long decompressed_len;
short *columns_present;
short source_or_target;
char requesting_before_after_ind;
} compressed_rec_def;
```

Input

compressed_rec
A pointer to the record in compressed format. Use the GET_RECORD_BUFFER function to obtain this value (see "GET_RECORD_BUFFER").

compressed_len
The length of the compressed record. Use the GET_RECORD_BUFFER (see "GET_RECORD_BUFFER") or GET_RECORD_LENGTH (see "GET_RECORD_LENGTH") 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

decompressed_rec
A pointer to the record returned in decompressed format. The record is assumed to be in the Oracle GoldenGate internal canonical format. The caller must ensure that the appropriate amount of memory is allocated to decompressed_rec.

decompressed_len
The returned length of the decompressed record.

columns_present
An array of values that indicate the columns present in the compressed record. For example, if the first, third and sixth columns exist in the compressed record, and the total number of columns in the table is seven, the array should contain:

```
1, 0, 1, 0, 0, 1, 0
```

This array helps mapping functions determine when and whether a compressed column should be mapped.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
EXIT_FN_RET_INVALID_PARAM
```

GET_BASE_OBJECT_NAME

Valid For

Extract and Replicat

Description

Use the `GET_BASE_OBJECT_NAME` function to retrieve the fully qualified name of the base object of a source or target object that is associated with the record being processed. This function is valid tables and other objects in a DDL operation.

To return only part of the base object name, see the following:

GET_BASE_OBJECT_NAME_ONLY GET_BASE_SCHEMA_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_BASE_OBJECT_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

`buffer`
A pointer to a buffer to accept the returned object name. The name is null-terminated.

`max_length`
The maximum length of your allocated buffer to accept the object name. This is returned as a `NULL` terminated string.

`source_or_target`
One of the following indicating whether to return the source or target object name.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

buffer
The fully qualified, null-terminated object name, for example `schema.object` or `catalog.schema.object`, depending on the database platform.
If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the object name is interpreted in the session character set.

actual length
The string length of the returned object name. The actual length does not include the null terminator. The actual length is 0 if the object is a table.

value_truncated
A flag (`0` or `1`) indicating whether or not the value was truncated. Truncation occurs if the length of the object 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_BASE_OBJECT_NAME_ONLY

Valid For

Extract and Replicat

Description

Use the `GET_BASE_OBJECT_NAME_ONLY` function to retrieve the unqualified name (without the catalog, container, or schema) of the base object of a source or target object that is associated with the record that is being processed. This function is valid for tables and other objects in a DDL operation.

To return the fully qualified name of a base object, see the following:

GET_OBJECT_NAME

To return only the schema of the base object, see the following:

GET_BASE_SCHEMA_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_BASE_OBJECT_NAME_ONLY, &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

buffer
A pointer to a buffer to accept the returned object name. The name is null-terminated.

max_length
The maximum length of your allocated buffer to accept the object name. This is returned as a `NULL` terminated string.

source_or_target
One of the following indicating whether to return the source or target object name.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

buffer
The fully qualified, null-terminated object name, for example `schema.object` or `catalog.schema.object`, depending on the database platform.
If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the object name is interpreted in the session character set.

actual length
The string length of the returned object name. The actual length does not include the null terminator. The actual length is 0 if the object is a table.

value_truncated
A flag (`0` or `1`) indicating whether or not the value was truncated. Truncation occurs if the length of the object 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_BASE_SCHEMA_NAME_ONLY

Description

Use the `GET_BASE_SCHEMA_NAME_ONLY` function to retrieve the name of the owner (such as schema), but not the name, of the base object of the source or target object associated with the record being processed. This function is valid for DDL operations.

To return the fully qualified name of a base object, see the following:

GET_BASE_OBJECT_NAME

To return only the unqualified base object name, see the following:

GET_BASE_OBJECT_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_BASE_SCHEMA_NAME_ONLY, &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

`buffer`
A pointer to a buffer to accept the returned schema name. The name is null-terminated.

`max_length`
The maximum length of your allocated buffer to accept the schema name. This is returned as a `NULL` terminated string.

`source_or_target`
One of the following indicating whether to return the source or target schema name.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

`buffer`
The fully qualified, null-terminated schema name.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the schema name is interpreted in the session character set.

actual_length
The string length of the returned name. The actual length does not include the null terminator.

value_truncated
A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the schema name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_BEFORE_AFTER_IND

Valid For

Extract and Replicat

Description

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 the before images of UPDATE operations 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

before_after_ind
One of the following to indicate whether the record is a before or after image.

```
BEFORE_IMAGE_VAL
AFTER_IMAGE_VAL
```

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_CATALOG_NAME_ONLY

Valid For

Extract and Replicat

Description

Use the `GET_CATALOG_NAME_ONLY` function to retrieve the name of theOracle CDB container, but not the name of the owner (such as schema) or object, of the source or target object associated with the record being processed. This function is valid for DML and DDL operations.

To return the fully qualified name of a table, see the following:

GET_TABLE_NAME

To return the fully qualified name of a non-table object, such as a user, view or index, see the following:

GET_OBJECT_NAME

To return only the unqualified table or object name, see the following:

GET_TABLE_NAME_ONLY

GET_OBJECT_NAME_ONLY

To return other parts of the table or object name, see the following:

GET_SCHEMA_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_CATALOG_NAME_ONLY, &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

buffer
A pointer to a buffer to accept the returned catalog name. The name is null-terminated. If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the catalog name is interpreted in the session character set.

max_length
The maximum length of your allocated buffer to accept the name. This is returned as a `NULL` terminated string.

source_or_target
One of the following indicating whether to return the source or target table catalog.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

buffer
The fully qualified, null-terminated catalog name.

actual_length
The string length of the returned name. The actual length does not include the null terminator.

value_truncated
A flag (`0` or `1`) indicating whether or not the value was truncated. Truncation occurs if the length of the catalog name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_COL_METADATA_FROM_INDEX

Valid For

Extract and Replicat

Description

Use the `GET_COL_METADATA_FROM_INDEX` function to retrieve column metadata by specifying the index of the desired column.

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
col_metadata_def column_meta_rec;
ERCALLBACK (GET_COL_METADATA_FROM_INDEX, &column_meta_rec, &result_code);
```

Buffer

```
typedef struct
{
  short column_index;
  char *column_name;
  long max_name_length;
  short native_data_type;
  short gg_data_type;
  short gg_sub_data_type;
  short is_nullable;
  short is_part_of_key;
  short key_column_index;
  short length;
  short precision;
  short scale;
  short source_or_target;
} col_metadata_def;
```

Input

`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

`column_name`
The column name of the column value to be returned.

`actual_name_length`
The actual length of the returned name.

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.

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).
- 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 Oracle GoldenGate data type of the column.

gg_sub_data_type
The Oracle GoldenGate sub-type of the column.

is_nullable
Flag indicating whether the column permits a null value (TRUE or FALSE).

is_part_of_key
Flag (TRUE or FALSE) indicating whether the column is part of the key that is being used by Oracle GoldenGate.

key_column_index
Indicates the order of the columns in the index. For example, the following table has two key columns that 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:

- cust_code returns 1
- name returns -1

- `city` returns `0`
- `state` returns `-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.

`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

Description

Use the `GET_COL_METADATA_FROM_NAME` function to retrieve column metadata by specifying the name of the desired column. If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include "usrdecs.h"
short result_code;
col_metadata_def column_meta_rec;
ERCALLBACK (GET_COL_METADATA_FROM_NAME, &column_meta_rec, &result_code);
```

Buffer

```
typedef struct
{
  short column_index;
  char *column_name;
  long max_name_length;
  short native_data_type;
  short gg_data_type;
  short gg_sub_data_type;
  short is_nullable;
```

```
  short is_part_of_key;
  short key_column_index;
  short length;
  short precision;
  short scale;
  short source_or_target;
} col_metadata_def;
```

Input

column_name
The column name 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

column_index
The column index of the column value to be returned.

actual_name_length
The actual length of the returned 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
```

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.

native_data_type
The native (to the database) data type of the column.

gg_data_type
The Oracle GoldenGate data type of the column.

gg_sub_data_type
The Oracle GoldenGate sub-type of the column.

is_nullable
Flag indicating whether the column permits a null value (`TRUE` or `FALSE`).

is_part_of_key
Flag (`TRUE` or `FALSE`) indicating whether the column is part of the key that is being used by Oracle GoldenGate.

key_column_index
Indicates the order of the columns in the index. For example, the following table has two key columns that are defined in one order in the table and another in the index definition.

```
CREATE TABLE tcustmer
(
cust_code        VARCHAR2(4),
name             VARCHAR2(30),
city             VARCHAR2(20),
state            CHAR(2),
PRIMARY KEY (city, cust_code)
USING INDEX
);
```

The return is as follows:

- `cust_code` returns `1`
- `name` returns `-1`
- `city` returns `0`
- `state` returns `-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

Description

Use the `GET_COLUMN_INDEX_FROM_NAME` function to determine the column index associated with a specified column name. If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_COLUMN_INDEX_FROM_NAME, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input

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

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

Description

Use the `GET_COLUMN_NAME_FROM_INDEX` function to determine the column name associated with a specified column index. If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_COLUMN_NAME_FROM_INDEX, &env_value, &result_code);
```

Buffer

```
typedef struct
{
char *buffer;
long max_length;
long actual_length;
short value_truncated;
short index;
short source_or_target;
} env_value_def;
```

Input

`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

`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

Description

Use the `GET_COLUMN_VALUE_FROM_INDEX` function to retrieve the column value from the data record using the specified column index. Column values are the basis for most logic within the user exit. You can base complex logic on the values of individual columns within the data record. You can specify the character format of the returned value.

If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (`CHAR/VARCHAR2/CLOB`, `NCHAR/NVARCHAR2/NCLOB`), a SQL date/timestamp/interval/number type)
- The `column_value_mode` indicator is set to `EXIT_FN_CNVTED_SESS_CHAR_FORMAT`.

Syntax

```
#include "usrdecs.h"
short result_code;
column_def column;
ERCALLBACK (GET_COLUMN_VALUE_FROM_INDEX, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input

column_value
A pointer to a buffer to accept the returned column value.

max_value_length
The maximum length of the returned column value. Typically, the maximum length is the length of the column value buffer. If ASCII format is specified with `column_value_mode`, the column value is null-terminated and the maximum length should equal the maximum length of the column value.

column_index
The column index of the column value to be returned.

column_value_mode
Indicates the format of the column value.

EXIT_FN_CHAR_FORMAT
ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type `UTF16_BE`, which is converted to UTF8.)

> **Note:**
>
> A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of `0` becomes the string terminator.

- Dates are in the format `CCYY-MM-DD HH:MI:SS.FFFFFF`, in which the fractional time is database-dependent.
- Numeric values are in their string format. For example, `123.45` is represented as `"123.45"`.
- Non-printable characters or binary values are converted to hexadecimal notation.
- Floating point types are output as null-terminated strings, to the first 14 significant digits.

EXIT_FN_RAW_FORMAT
Internal Oracle GoldenGate canonical format: This format includes a two-byte `NULL` indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type.

EXIT_FN_CNVTED_SESS_CHAR_FORMAT
User exit character set: This only applies if the column data type is:

- a character-based type, single or multi-byte
- a numeric type with a string representation

This format is not null-terminated.

source_or_target
One of the following to indicate whether to use the source or the target data record to retrieve the column value.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

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.

requesting_before_after_ind
To get the before image of the column, set the `char requesting_before_after_ind` flag to `BEFORE_IMAGE_VAL`. To get the after image, set it to `AFTER_IMAGE_VAL`. The default is to always work with the after image unless the before is specified.
To make the before images available, you can use the `GETUPDATEBEFORES` parameter for the `TABLE` statement that contains the table, or you can use the `INCLUDEUPDATEBEFORES` option within the `CUSEREXIT` parameter statement. Both will cause the same callout to the user exit for `process_record`.

Output

column_value
A pointer to the returned column value. If `column_value_mode` is specified as `EXIT_FN_CHAR_FORMAT`, the column value is returned as a null-terminated ASCII string; otherwise, the column value is returned in the Oracle GoldenGate internal canonical format. In ASCII format, dates are returned in the following format:

```
YYYY-MM-DD HH:MI:SS.FFFFFF
```

The inclusion of fractional time is database-dependent.

actual_value_length
The string length of the returned column name, in bytes. The actual length does not include a null terminator when `column_value_mode` is specified as `EXIT_FN_CHAR_FORMAT`.

null_value
A flag (`0` or `1`) indicating whether or not the column value is null. If the `null_value` flag is `1`, then the column value buffer is filled with null bytes.

value_truncated
A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the column value exceeds the maximum buffer length. If column_value_mode was specified as EXIT_FN_CHAR_FORMAT, the null terminator is included in the length of the column.

char more_lob_data
A flag that indicates if more LOB data is present beyond the initial 4K that can be stored in the base record. When a LOB is larger than the 4K limit, it is stored in LOB fragments.
You must allocate the appropriate amount of memory to contain the returned values. Oracle GoldenGate will access LOB columns up to 8K of data at all times, filling up the buffer to the amount that the user exit has allocated. If the LOB is larger than that which was allocated, subsequent callbacks are required to obtain the total column data, until all data has been sent to the user exit.
To determine the end of the data, evaluate more_lob_data. The user exit sets this flag to either CHAR_NO_VAL or CHAR_YES_VAL before accessing a new column. If this flag is still initialized after first callback and is not set to either CHAR_YES_VAL or CAR_NO_VAL, then one of the following is true:

- Enough memory was allocated to handle the LOB.
- It is not a LOB.
- It was not over the 4K limit of the base trail record size.

It is recommended that you obtain the source table metadata to determine if a column might be a LOB.

Return Values

```
EXIT_FN_RET_BAD_COLUMN_DATA
EXIT_FN_RET_COLUMN_NOT_FOUND
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_COLUMN_VALUE_FROM_NAME

Valid For

Extract and Replicat

Description

Use the GET_COLUMN_VALUE_FROM_NAME function to retrieve the column value from the data record by using the specified column name. Column values are the basis for most logic within the user exit. You can base complex logic on the values of individual columns within the data record.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (CHAR/VARCHAR2/CLOB, NCHAR/NVARCHAR2/NCLOB), a SQL date/timestamp/interval/number type)

- The `column_value_mode` indicator is set to `EXIT_FN_CNVTED_SESS_CHAR_FORMAT`.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include "usrdecs.h"
short result_code;
column_def column;
ERCALLBACK (GET_COLUMN_VALUE_FROM_NAME, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input

`column_value`
A pointer to a buffer to accept the returned column value.

`max_value_length`
The maximum length of the returned column value. Typically, the maximum length is the length of the column value buffer. If ASCII format is specified (see `column_value_mode`) the column value is null-terminated, and the maximum length should equal the maximum length of the column value.

`column_name`
The name of the column for the column value to be returned.

`column_value_mode`
Indicates the character set of the column value.

> **`EXIT_FN_CHAR_FORMAT`**
> ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type `UTF16_BE`, which is converted to UTF8.)

> **Note:**
>
> A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of `0` becomes the string terminator.

- Dates are in the format `CCYY-MM-DD HH:MI:SS.FFFFFF`, in which the fractional time is database-dependent.
- Numeric values are in their string format. For example, `123.45` is represented as `"123.45"`.
- Non-printable characters or binary values are converted to hexadecimal notation.
- Floating point types are output as null-terminated strings, to the first 14 significant digits.

`EXIT_FN_RAW_FORMAT`
Internal Oracle GoldenGate canonical format: This format includes a two-byte null indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type.

`EXIT_FN_CNVTED_SESS_CHAR_FORMAT`
User exit character set: This only applies if the column data type is:

- a character-based type, single or multi-byte
- a numeric type with a string representation

This format is not null-terminated.

`source_or_target`
One of the following indicating whether to use the source or target data record to retrieve the column value.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

`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

column_value
A pointer to the returned column value. If column_value_mode is specified as EXIT_FN_CHAR_FORMAT, the column value is returned as a null-terminated ASCII string; otherwise, the column value is returned in the Oracle GoldenGate internal canonical format. In ASCII format, dates are returned in the following format:

```
CCYY-MM-DD HH:MI:SS.FFFFFF
```

The inclusion of fractional time is database-dependent.

actual length
The string length of the returned column name. The actual length does not include a null terminator when column_value_mode is specified as EXIT_FN_CHAR_FORMAT.

null_value
A flag (0 or 1) indicating whether or not the column value is null. If the null_value flag is 1, then the column value buffer is filled with null bytes.

value_truncated
A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the column value exceeds the maximum buffer length. If column_value_mode was specified as EXIT_FN_CHAR_FORMAT, the null terminator is included in the length of the column.

char more_lob_data
A flag that indicates if more LOB data is present beyond the initial 4K that can be stored in the base record. When a LOB is larger than the 4K limit, it is stored in LOB fragments.
You must allocate the appropriate amount of memory to contain the returned values. Oracle GoldenGate will access LOB columns up to 8K of data at all times, filling up the buffer to the amount that the user exit has allocated. If the LOB is larger than that which was allocated, subsequent callbacks are required to obtain the total column data, until all data has been sent to the user exit.
To determine the end of the data, evaluate more_lob_data. The user exit sets this flag to either CAR_NO_VAL or CHAR_YES_VAL before accessing a new column. If this flag is still initialized after first callback and is not set to either CHAR_YES_VAL or CAR_NO_VAL, then one of the following is true:

- Enough memory was allocated to handle the LOB.
- It is not a LOB.
- It was not over the 4K limit of the base trail record size.

It is recommended that you obtain the source table metadata to determine if a column might be a LOB.

Return Values

```
EXIT_FN_RET_BAD_COLUMN_DATA
EXIT_FN_RET_COLUMN_NOT_FOUND
EXIT_FN_RET_INVALID_COLUMN
```

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

Example

```
memset (&col_meta, 0, sizeof(col_meta));
if (record.mapped)
col_meta.source_or_target = EXIT_FN_TARGET_VAL;
else
col_meta.source_or_target = EXIT_FN_SOURCE_VAL;
col_meta.source_or_target = EXIT_FN_SOURCE_VAL;
col_meta.column_name = (char *)malloc(100);
col_meta.max_name_length = 100;
col_meta.column_index = 1;

call_callback (GET_COL_METADATA_FROM_NAME, &col_meta, &result_code);
```

GET_DATABASE_METADATA

Valid For

Extract and Replicat

Description

Use the `GET_DATABASE_METADATA` function to return the metadata of the database that is associated with a record.

Buffer

```
typedef struct
{
char*    dbName;
long      dbName_max_length;
long      dbName_actual_length;
unsigned char    dbNameMetadata[MAXDBOBJTYPE];
char*   locale;
long     locale_max_length;
long     locale_actual_length;
} database_def;
typedef struct
{
    database_def  source_db_def;
    database_def  target_db_def;
} database_defs;
```

Input

`dbname`
A pointer to a buffer to accept the database name.

`dbname_max_length`
The maximum length of the buffer to hold the name.

`dbname_actual_length`
The actual length of the database name.

dbNameMetadata
The name metadata for case-sensitivity, which is the same value that is written by Extract and the data pump to a trail. See for a list of macros that can be used by the user exit to checkUsing Macros database object name metadata, given an object name type.

locale
A null-terminated character string specifying the locale of the database. This is returned as a conjunction of:

- ISO-639 two-letter language code
- ISO-3166 two-letter country code
- Variant code using '_' U+005F as separator.

Example: `"en_US", "ja_Japen"`

locale_max_length
The maximum length of the buffer to accept the locale.

locale_actual_length
The actual length of the locale.

database_def source_db_def
Directs the process to return metadata for the source database.

database_def target_db_def
Directs the process to return metadata for the target database.

GET_DDL_RECORD_PROPERTIES

Valid For

Extract and Replicat, for databases for which DDL replication is supported

Description

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

If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the character data that is exchanged between the user exit and the process is interpreted in the session character set. This includes the DDL type, the object type, the two- or three-part object name, the owner name and the DDL text itself.

```
#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; /* Fully qualified name of the object
  (3-part for CDB, 2-part for non-CDB) */
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

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.

object_length
Contains the length of the owner of the object (schema or database).

ddl_text_length
Contains the length of the actual DDL statement text.

ddl_type_max_length
The maximum length of the DDL operation type that is returned by `*ddl_type`. The DDL type is any DDL command that is valid for the database, such as `ALTER`.

object_type_max_length
The maximum length of the object type that is returned by `*object_type`. The object type is any object that is valid for the database, such as `TABLE`, `INDEX`, and `TRIGGER`.

object_max_length
The maximum length of the name of the object that is returned by `*object_name`.

owner_max_length
The maximum length of the name of the owner that is returned by `*owner_name`.

ddl_text_max_length
The maximum length of the text of the DDL statement that is returned by `*ddl_text`.

source_or_target
One of the following indicating whether to return the operation type for the source or the target data record.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

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
```

@GETENV

Use the `@GETENV` function to return information about the Oracle GoldenGate environment. You can use the information as input into the following:

- Stored procedures or queries (with `SQLEXEC`)
- Column maps (with the `COLMAP` option of `TABLE` or `MAP`)
- User tokens (defined with the `TOKENS` option of `TABLE` and mapped to target columns by means of the `@TOKEN` function)
- The `GET_ENV_VALUE` user exit function (see "GET_ENV_VALUE")

> **Note:**
>
> All syntax options must be enclosed within quotes as shown in the syntax descriptions.

- Retrieve the value of the `DB_UNIQUE_NAME` parameter of the source or the target databases, depending on which processes (Extract or Replicat) executes the function.

Syntax

```
@GETENV (
'LAG' , 'unit' |
'LASTERR' , 'error_info' |
'JULIANTIMESTAMP' |
'JULIANTIMESTAMP_PRECISE' |
'RECSOUTPUT' |
{'STATS'|'DELTASTATS'}, ['TABLE', 'table'], 'statistic' |
'GGENVIRONMENT', 'environment_info' |
'GGFILEHEADER', 'header_info'|
'GGHEADER', 'header_info' |
'RECORD', 'location_info' |
'DBENVIRONMENT', 'database_info,',
'TRANSACTION', 'transaction_info' |
'OSVARIABLE', 'variable' |
'TLFKEY', SYSKEY, unique_key
'USERNAME',
'OSUSERNAME',
'MACHINENAME',
'PROGRAMNAME',
'CLIENTIDENTIFIER',
'SOURCEDATABASEINFO'
)
```

'LAG' , 'unit'

Valid for Extract and Replicat.

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.

Syntax

```
@GETENV ('LAG', {'SEC'|'MSEC'|'MIN'})
```

'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.

'LASTERR' , 'error_info'

Valid for Replicat.

Use the `LASTERR` option of `@GETENV` to return information about the last failed operation processed by Replicat.

Syntax

```
@GETENV ('LASTERR', {'DBERRNUM'|'DBERRMSG'|'OPTYPE'|'ERRTYPE'})
```

'DBERRNUM'
Returns the database error number associated with the failed operation.

'DBERRMSG'
Returns the database error message associated with the failed operation.

'OPTYPE'
Returns the operation type that was attempted.

'ERRTYPE'
Returns the type of error. Possible results are:

- `DB` (for database errors)
- `MAP` (for errors in mapping)

'JULIANTIMESTAMP' | 'JULIANTIMESTAMP_PRECISE'

Valid for Extract and Replicat.

Use the `JULIANTIMESTAMP` option of `@GETENV` to return the current time in Julian format. The unit is microseconds (one millionth of a second). On a Windows machine, the value is padded with zeros (0) because the granularity of the Windows timestamp is milliseconds (one thousandth of a second). For example, the following is a typical column mapping:

```
MAP dbo.tab8451, Target targ.tabjts, COLMAP (USEDEFAULTS, &
JTSS = @GETENV ('JULIANTIMESTAMP')
JTSFFFFFF = @date ('yyyy-mm-dd hh:mi:ss.ffffff', 'JTS', &
@getenv ('JULIANTIMESTAMP') ) )
;
```

Possible values that the `JTSS` and `JTSFFFFFF` columns can have are:

```
212096320960773000 2010-12-17:16:42:40.773000
212096321536540000 2010-12-17:16:52:16.540000
212096322856385000 2010-12-17:17:14:16.385000
212096323062919000 2010-12-17:17:17:42.919000
212096380852787000 2010-12-18:09:20:52.787000
```

The last three digits (the microseconds) of the number all contain the padding of 0s .

Optionally, you can use the `'JULIANTIMESTAMP_PRECISE'` option to obtain a timestamp with high precision though this may effect performance.

> **Note:**
>
> Do not use these values for ordering operations. Instead use this value:
> `@COMPUTE(@COMPUTE(@NUMSTR(@GETENV ("RECORD",`
> `"FILESEQNO")*100000000000)+@NUMSTR(@GETENV ("RECORD", "FILERBA")))"`

Syntax

```
@GETENV ('JULIANTIMESTAMP')
@GETENV ('JULIANTIMESTAMP_PRECISE')
```

'RECSOUTPUT'

Valid for Extract.

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')

{'STATS'|'DELTASTATS'}, ['TABLE', 'table'], 'statistic'
```

Valid for Extract and Replicat.

Use the STATS and DELTASTATS options of @GETENV to return the number of operations that were processed per table for any or all of the following:

- INSERT operations
- UPDATE operations
- DELETE operations
- TRUNCATE operations
- Total DML operations
- Total DDL operations
- Number of conflicts that occurred, if the Conflict Detection and Resolution (CDR) feature is used.
- Number of CDR resolutions that succeeded
- Number of CDR resolutions that failed

Any errors in the processing of this function, such as an unresolved table entry or incorrect syntax, returns a zero (0) for the requested statistics value.

Understanding How Recurring Table Specifications Affect Operation Counts

An Extract that is processing the same source table to multiple output trails returns statistics based on each localized output trail to which the table linked to @GETENV is written. For example, if Extract captures 100 inserts for table ABC and writes table ABC to three trails, the result for the @GETENV is 300

```
EXTRACT ABC
...
EXTTRAIL c:\north\aa;
TABLE TEST.ABC;
EXTTRAIL c:\north\bb;
TABLE TEST.ABC;
TABLE EMI, TOKENS (TOKEN-CNT = @GETENV ('STATS', 'TABLE', 'ABC', 'DML'));
EXTTRAIL c:\north\cc;
TABLE TEST.ABC;
```

In the case of an Extract that writes a source table multiple times to a single output trail, or in the case of a Replicat that has multiple MAP statements for the same TARGET table, the statistics results are based on all matching TARGET entries. For example, if Replicat filters 20 rows for REGION 'WEST,' 10 rows for REGION 'EAST,' 5 rows for REGION 'NORTH,' and 2 rows for REGION 'SOUTH' (all for table ABC) the result of the @GETENV is 37.

```
REPLICAT ABC
...
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'WEST'));
```

```
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'EAST'));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'NORTH'));
MAP TEST.ABC, TARGET TEST.ABC, FILTER (@STREQ (REGION, 'SOUTH'));
MAP TEST.EMI, TARGET TEST.EMI, &
    COLMAP (CNT = @GETENV ('STATS', 'TABLE', 'ABC', 'DML'));
```

Capturing Multiple Statistics

You can execute multiple instances of `@GETENV` to get counts for different operation types.

This example returns statistics only for `INSERT` and `UPDATE` operations:

```
REPLICAT TEST
..
..
MAP TEST.ABC, TARGET TEST.ABC, COLMAP (USEDEFAULTS, IU = @COMPUTE (@GETENV &
    ('STATS', 'TABLE', 'ABC', 'DML') - (@GETENV ('STATS', 'TABLE', &
    'ABC', 'DELETE'));
```

This example returns statistics for DDL and `TRUNCATE` operations:

```
REPLICAT TEST2
..
..
MAP TEST.ABC, TARGET TEST.ABC, COLMAP (USEDEFAULTS, DDL = @COMPUTE &
(@GETENV ('STATS', 'DDL') + (@GETENV ('STATS', 'TRUNCATE'));
```

Example Use Case

In the following use case, if all DML from the source is applied successfully to the target, Replicat suspends by means of `EVENTACTIONS` with `SUSPEND`, until resumed from GGSCI with `SEND REPLICAT` with `RESUME`.

`GETENV` used in Extract parameter file:

```
TABLE HR1.HR*;
TABLE HR1.STAT, TOKENS ('env_stats' = @GETENV ('STATS', 'TABLE', &
    'HR1.HR*', 'DML'));
```

`GETENV` used in Replicat parameter file:

```
MAP HR1.HR*, TARGET HR2.*;
MAP HR1.STAT, TARGET HR2.STAT, filter (
    @if (
    @token ('stats') =
    @getenv ('STATS', 'TABLE', 'TSSCAT.TCUSTORD', 'DML'), 1, 0 )
    ),
    eventactions (suspend);
```

Using Statistics in FILTER Clauses

Statistics returned by `STATS` and `DELTASTATS` are dynamic values and are incremented after mapping is performed. Therefore, when using CDR statistics in a `FILTER` clause in each of multiple `MAP` statements, you need to order the `MAP` statements in descending order of the statistics values. If the order is not correct, Oracle GoldenGate returns error OGG-01921. For

detailed information about this requirement, see Document 1556241.1 in the Knowledge base of My Oracle Support at http://support.oracle.com.

Example 4-1 MAP statements containing statistics in FILTER clauses

In the following example, the MAP statements containing the filter for the CDR_CONFLICTS statistic are ordered in descending order of the statistic: >3, then =3, then <3.

```
MAP TEST.GG_HEARTBEAT_TABLE, TARGET TEST.GG_HEARTBEAT_TABLE COMPARECOLS (ON
UPDATE ALL),RESOLVECONFLICT(UPDATEROWEXISTS,(DEFAULT, OVERWRITE)),FILTER
(@GETENV ("STATS", "CDR_CONFLICTS") > 3),EVENTACTIONS (LOG INFO);MAP
TEST.GG_HEARTBEAT_TABLE, TARGET TEST.GG_HEARTBEAT_TABLE COMPARECOLS (ON
UPDATE ALL),RESOLVECONFLICT(UPDATEROWEXISTS,(DEFAULT, OVERWRITE)),FILTER
(@GETENV ("STATS", "CDR_CONFLICTS") = 3),EVENTACTIONS (LOG WARNING);MAP
TEST.GG_HEARTBEAT_TABLE, TARGET TEST.GG_HEARTBEAT_TABLE COMPARECOLS (ON
UPDATE ALL),RESOLVECONFLICT(UPDATEROWEXISTS,(DEFAULT, OVERWRITE)),FILTER
(@GETENV ("STATS", "CDR_CONFLICTS") < 3),EVENTACTIONS (LOG WARNING);
```

Syntax

```
@GETENV ({'STATS' | 'DELTASTATS'}, ['TABLE', 'table'], 'statistic')
```

{'STATS' | 'DELTASTATS'}
STATS returns counts since process startup, whereas DELTASTATS returns counts since the last execution of a DELTASTATS.
The execution logic is as follows:

- When Extract processes a transaction record that satisfies @GETENV with STATS or DELTASTATS, the table name is matched against resolved source tables in the TABLE statement.
- When Replicat processes a trail record that satisfies @GETENV with STATS or DELTASTATS, the table name is matched against resolved target tables in the TARGET clause of the MAP statement.

'TABLE', '*table*'
Executes the STATS or DELTASTATS only for the specified table or tables. Without this option, counts are returned for all tables that are specified in TABLE (Extract) or MAP (Replicat) parameters in the parameter file.
Valid *table_name* values are:

- '*schema.table*' specifies a table.
- *'table'* specifies a table of the default schema.
- '*schema.**' specifies all tables of a schema.
- '*' specifies all tables of the default schema.

For example, the following counts DML operations only for tables in the hr schema:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = @GETENV ('STATS',
'TABLE', 'hr.*', 'DML'));
```

Likewise, the following counts DML operations only for the `emp` table in the `hr` schema:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = @GETENV ('STATS',
'TABLE', 'hr.emp', 'DML'));
```

By contrast, because there are no specific tables specified for `STATS` in the following example, the function counts all `INSERT`, `UPDATE`, and `DELETE` operations for all tables in all schemas that are represented in the `TARGET` clauses of `MAP` statements:

```
MAP fin.*, TARGET fin.*;
MAP hr.*, TARGET hr.*;
MAP hq.rpt, TARGET hq.rpt, COLMAP (USEDEFAULTS, CNT = &
@GETENV ('STATS', 'DML'));
```

'statistic'
The type of statistic to return. See Using Statistics in FILTER Clauses for important information when using statistics in `FILTER` clauses in multiple `TABLE` or `MAP` statements.

'INSERT'
Returns the number of `INSERT` operations that were processed.

'UPDATE'
Returns the number of `UPDATE` operations that were processed.

'DELETE'
Returns the number of `DELETE` operations that were processed.

'DML'
Returns the total of `INSERT`, `UPDATE`, and `DELETE` operations that were processed.

'TRUNCATE'
Returns the number of `TRUNCATE` operations that were processed. This variable returns a count only if Oracle GoldenGate DDL replication is not being used. If DDL replication is being used, this variable returns a zero.

'DDL'
Returns the number of DDL operations that were processed, including `TRUNCATE`s and DDL specified in `INCLUDE` and `EXCLUDE` clauses of the `DDL` parameter, all scopes (`MAPPED`, `UNMAPPED`, `OTHER`). This variable returns a count only if Oracle GoldenGate DDL replication is being used. This variable is not valid for `'DELTASTATS'`.

'CDR_CONFLICTS'
Returns the number of conflicts that Replicat detected when executing the Conflict Detection and Resolution (CDR) feature.
Example for a specific table:

```
@GETENV ('STATS','TABLE','HR.EMP','CDR_CONFLICTS')
```

Example for all tables processed by Replicat:

```
@GETENV ('STATS','CDR_CONFLICTS')
```

'CDR_RESOLUTIONS_SUCCEEDED'
Returns the number of conflicts that Replicat resolved when executing the Conflict Detection and Resolution (CDR) feature.
Example for a specific table:

```
@GETENV ('STATS','TABLE','HR.EMP', 'CDR_RESOLUTIONS_SUCCEEDED')
```

Example for all tables processed by Replicat:

```
@GETENV ('STATS','CDR_RESOLUTIONS_SUCCEEDED')
```

'CDR_RESOLUTIONS_FAILED'
Returns the number of conflicts that Replicat could not resolve when executing the Conflict Detection and Resolution (CDR) feature.
Example for a specific table:

```
@GETENV ('STATS','TABLE','HR.EMP', 'CDR_RESOLUTIONS_FAILED')
```

Example for all tables processed by Replicat:

```
@GETENV ('STATS','CDR_RESOLUTIONS_FAILED')
```

'GGENVIRONMENT' , '*environment_info*'

Valid for Extract and Replicat.

Use the `GGENVIRONMENT` option of `@GETENV` to return information about the Oracle GoldenGate environment.

Syntax

```
@GETENV ('GGENVIRONMENT', {'DOMAINNAME'|'GROUPDESCRIPTION'|'GROUPNAME'|
   'GROUPTYPE'|'HOSTNAME'|'OSUSERNAME'|'PROCESSID')
```

'DOMAINNAME'
(Windows only) Returns the domain name associated with the user that started the process.

'GROUPDESCRIPTION'
Returns the description of the group, taken from the checkpoint file. Requires that a description was provided with the `DESCRIPTION` parameter when the group was created with the `ADD` command.

'GROUPNAME'
Returns the name of the process group.

'GROUPTYPE'
Returns the type of process, either `EXTRACT` or `REPLICAT`.

'HOSTNAME'
Returns the name of the system running the Extract or Replicat process.

'OSUSERNAME'
Returns the operating system user name that started the process.

'PROCESSID'
Returns the process ID that is assigned to the process by the operating system.

'GGHEADER' , '*header_info*'

Valid for Extract and Replicat.

Use the `GGHEADER` option of `@GETENV` to return information from the header portion of an Oracle GoldenGate trail record. The header describes the transaction environment of the record. For more information on record headers and record types, see Trail Record Format.

Syntax

```
@GETENV ('GGHEADER', {'BEFOREAFTERINDICATOR'|'COMMITTIMESTAMP'|'LOGPOSITION'|
   'LOGRBA'|'OBJECTNAME'|'TABLENAME'|'OPTYPE'|'RECORDLENGTH'|
   'TRANSACTIONINDICATOR'})
```

> **Note:**
>
> Do not use `TIMESTAMP_PRECISE` for ordering operations. Instead use this value:
> `@COMPUTE(@COMPUTE(@NUMSTR(@GETENV ("RECORD", "FILESEQNO"))*100000000000)+@NUMSTR(@GETENV ("RECORD", "FILERBA")))`

'BEFOREAFTERINDICATOR'
Returns the before or after indicator showing whether the record is a before image or an after image. Possible results are:

- `BEFORE` (before image)
- `AFTER` (after image)

'COMMITTIMESTAMP'
Returns the transaction timestamp (the time when the transaction committed) expressed in the format of `YYYY-MM-DD HH:MI:SS.FFFFFF`, for example:

```
2011-01-24 17:08:59.000000
```

'LOGPOSITION'
Returns the position of the Extract process in the data source. (See the `LOGRBA` option.)

'LOGRBA'
`LOGRBA` and `LOGPOSITION` store details of the position in the data source of the record. For transactional log-based products, `LOGRBA` is the sequence number and `LOGPOSITION` is the relative byte address. However, these values will vary depending on the capture method and database type.

'OBJECTNAME' | 'TABLENAME'
Returns the table name or object name (if a non-table object).

'OPTYPE'
Returns the type of operation. Possible results are:

```
INSERT
UPDATE
DELETE
```

```
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.

`'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.
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.)

`'GGFILEHEADER' , 'header_info'`

Valid for Replicat only.

Use the `GGFILEHEADER` option of `@GETENV` to retrieve attributes of an Oracle GoldenGate Extract file or trail file. These attributes are stored as tokens in the file header.

> **Note:**
>
> If a given database, operating system, or Oracle GoldenGate version does not provide information that relates to a given token, a `NULL` value will be returned.

Syntax

```
@GETENV ('GGFILEHEADER', {'COMPATIBILITY'|'CHARSET'|'CREATETIMESTAMP'|
   'FILENAME'|'FILETYPE'|'FILESEQNO'|'FILESIZE'|'FIRSTRECCSN'|
   'LASTRECCSN'|'FIRSTRECIOTIME'|'LASTRECIOTIME'|'URI'|'URIHISTORY'|
   'GROUPNAME'|'DATASOURCE'|'GGMAJORVERSION'|'GGMINORVERSION'|
   'GGVERSIONSTRING'|'GGMAINTENANCELEVEL'|'GGBUGFIXLEVEL'|'GGBUILDNUMBER'|
   'HOSTNAME'|'OSVERSION'|'OSRELEASE'|'OSTYPE'|'HARDWARETYPE'|
   'DBNAME'|
'DBUNIQUENAME'|'DBINSTANCE'|'DBTYPE'|'DBCHARSET'|'DBMAJORVERSION'|

'DBMINORVERSION'|'DBVERSIONSTRING'|'DBCLIENTCHARSET'|'DBCLIENTVERSIONSTRING'|
   'LASTCOMPLETECSN'|'LASTCOMPLETEXIDS'|'LASTCSN'|'LASTXID'|
   'LASTCSNTS'|'RECOVERYMODE'})
```

`'COMPATIBILITY'`
Returns the compatibility level of the trail file. The compatibility level of the current Oracle GoldenGate version must be greater than, or equal to, the compatibility level of the trail file to be able to read the data records in that file. Current valid values are from 0 or 6.

- 1 means that the trail file is of Oracle GoldenGate version 10.0 or later, which supports file headers that contain file versioning information.
- 0 means that the trail file is of an Oracle GoldenGate version that is older than 10.0. File headers are not supported in those releases. The 0 value is used for backward compatibility to those Oracle GoldenGate versions.
- 5 means that the trail file is of Oracle GoldenGate version 12.2 or later.
- 6 means that the trail file is of Oracle GoldenGate version 12.3.0.1.

  This value keeps increasing as per the Oracle GoldenGate version depending on the trail file version.

'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 file system.

'FILETYPE'
Returns a numerical value 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. The valid values are:

- `0 - EXTFILE`
- `1 - EXTTRAIL`
- `2 - UNIFIED and EXTFILE`
- `3 - UNIFIED and EXTTRAIL`

'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.

'LASTRECCSN'
Returns the commit sequence number (CSN) of the last record in the trail file. Value is `NULL` until the trail file is completed.

'FIRSTRECIOTIME'
Returns the time that the first record was written to the trail file. Value is `NULL` until the trail file is completed.

'LASTRECIOTIME'
Returns the time that the last record was written to the trail file. Value is `NULL` until the trail file is completed.

'RECOVERYMODE'
Returns recovery information for internal Oracle GoldenGate use. It is usually set to `APPENDMODE`.

'URI'
Returns the universal resource identifier of the process that created the trail file, in the following format:

host_name:*dir*:[:*dir*][:*dir_n*]*group_name*

Where:

- `host_name` is the name of the server that hosts the process
- `dir` is a subdirectory of the Oracle GoldenGate installation path.
- `group_name` is the name of the process group that is linked with the process.

The following example shows where the trail was processed and by which process. This includes a history of previous runs.

```
sys1:home:oracle:v9.5:extora
```

'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.

'GROUPNAME'
Returns the name of the group that is associated with the Extract process that created the trail. The group name is the one that was supplied when the `ADD EXTRACT` command was issued.

'DATASOURCE'
Returns the data source that was read by the process as a number. The return value can be one of the following:

- `DS_EXTRACT_TRAILS`: The source was an Oracle GoldenGate extract file, populated with change data. The return value is `0`.
- `DS_DATABASE`: The source was a direct select from database table written to a trail, used for `SOURCEISTABLE`-driven initial load. The return value is `2`.
- `DS_TRAN_LOGS`: The source was the database transaction log. The return value is `3`.
- `DS_INITIAL_DATA_LOAD`: The source was a direct select from database tables for an initial load. The return value is `4`.
- `DS_VAM_EXTRACT`: The source was a vendor access module (VAM). The return value is `5`.
- `DS_VAM_TWO_PHASE_COMMIT`: The source was a VAM trail. The return value is `6`.

'GGMAJORVERSION'
Returns the major version of the Extract process that created the trail, expressed as an integer. For example, if a version is 1.2.3, it returns 1.

'GGMINORVERSION'
Returns the minor version of the Extract process that created the trail, expressed as an integer. For example, if a version is 1.2.3, it returns 2.

'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.

'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:

- `Version s10_69`
- `#1 SMP Fri Feb 24 16:56:28 EST 2006`
- `5.00.2195 Service Pack 4`

'OSRELEASE'
Returns the release version of the operating system of the machine where the Extract that wrote the trail is running. For example, release versions of the examples given for `OSVERSION` could be:

- `5.10`
- `2.6.9-34.ELsmp`

'OSTYPE'
Returns the type of operating system of the machine where the Extract that wrote the trail is running. For example:

- `SunOS`
- `Linux`
- `Microsoft Windows`

'HARDWARETYPE'
Returns the type of hardware of the machine where the Extract that wrote the trail is running. For example:

- sun4u
- x86_64
- x86

'DBNAME'
Returns the name of the database, for example findb.

'DBUNIQUENAME'
Returns the value of the DB_UNIQUE_NAME token as read from the header of the source trail file. Its value matches the DB_UNIQUE_NAME parameter of the source database.

'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
MSSQL
MYSQL
ORACLE
TERADATA
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.)

'LASTCOMPLETECSN'
Returns recovery information for internal Oracle GoldenGate use.

'LASTCOMPLETEXIDS'
Returns recovery information for internal Oracle GoldenGate use.

'LASTCSN'
Returns recovery information for internal Oracle GoldenGate use.

'LASTXID'
Returns recovery information for internal Oracle GoldenGate use.

'LASTCSNTS'
Returns recovery information for internal Oracle GoldenGate use.

'RECORD' , '*location_info*'

Valid for a data pump Extract or Replicat.

Use the `RECORD` option of `@GETENV` to return the location or Oracle rowid of a record in an Oracle GoldenGate trail file.

Syntax

```
@GETENV ('RECORD',
{'TIMESTAMP_PRECISE'|'FILESEQNO'|'FILERBA'|'ROWID'|'RSN'|'TIMESTAMP'})
```

'TIMESTAMP_PRECISE'
Valid for Extract or Replicat.
The `TIMESTAMP_PRECISE` option returns the timestamp from year to microseconds. However, depending on the database, the value can be in milliseconds with 0 microseconds.

'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.

'ROWID'
(Valid for Oracle) Returns the row id of the record.

'RSN'
Returns the record sequence number within the transaction. This value does not always generate uniquely increasing values and should not be used to order operations. For ordering transactions or DML operations within a transaction, use the information outlined in MOS DOC ID 1340823.1.

'TIMESTAMP'
Returns the timestamp of the record.

Example:

```
REC-TIMESTAMP: 2017-10-31 06:21:07 REC-TIMESTAMP-PRECISE: 2017-10-31
06:21:07.478064
```

'DBENVIRONMENT' , '*database_info*'

Valid for Extract and Replicat.

Use the `DBENVIRONMENT` option of `@GETENV` to return global environment information for a database.

Syntax

```
@GETENV ('DBENVIRONMENT',
{'DBNAME'|'DBUNIQUENAME'|'DBVERSION'|'DBUSER'|'SERVERNAME'})
```

'DBNAME'
Returns the database name.

'DBUNIQUENAME'
Returns the value of the `DB_UNIQUE_NAME` parameter of the database to which the process is connected. The source database in the case of Extract and the target database for Replicat. This value will be set only for Oracle databases.

'DBVERSION'
Returns the database version.

'DBUSER'
Returns the database login user. Note that SQL Server does not log the user ID.

'SERVERNAME'
Returns the name of the server.

'TRANSACTION' , '*transaction_info*

Valid for Extract.

Use the `TRANSACTION` option of `@GETENV` to return information about a source transaction. This option is valid for the Extract process but not for pump Extract and Replicat.

Syntax

```
@GETENV ('TRANSACTION',
{'TIMESTAMP_PRECISE'|'TRANSACTIONID'|'XID'|'CSN'|'TIMESTAMP'|'NAME'|
   'USERNAME'|'PLANNAME' | 'LOGBSN' | 'REDOTHREAD' | 'PROGRAMNAME' |
'CLIENTIDENTIFIER' | 'MACHINENAME' | 'USERNAME')
```

> **Note:**
>
> Do not use `TIMETSAMP_PRECISE` or `TIMESTAMP` for ordering operations. Instead use this value: `@COMPUTE(@COMPUTE(@NUMSTR(@GETENV ("RECORD", "FILESEQNO"))*100000000000)+@NUMSTR(@GETENV ("RECORD", "FILERBA")))`

'TIMESTAMP_PRECISE'
This option is valid for Extract. Use the `TIMESTAMP_PRECISE` returns the timestamp from year to microseconds. However, depending on the database, the value can be in milliseconds with 0 microseconds

'TRANSACTIONID' | 'XID'
Returns the transaction ID number. Either `TRANSACTIONID` or `XID` can be used. 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.
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, see Commit Sequence Number (CSN).

'TIMESTAMP'
Returns the commit timestamp of the transaction.

'NAME'
Returns the transaction name, if available.

'USERNAME'
(Oracle) Returns the Oracle user name of the database user that committed the last transaction. This is not valid for pump Extract and/or Replicat.

'PLANNAME'
(DB2 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.

'LOGBSN'
Returns the begin sequence number (BSN) in the transaction log. The BSN is the native sequence number that identifies the beginning of the oldest uncommitted transaction that is held in Extract memory. For example, given an Oracle database, the BSN would be expressed as a system change number (SCN). The BSN corresponds to the current I/O checkpoint value of Extract. This value can be obtained from the trail by Replicat when `@GETENV ('TRANSACTION', 'LOGBSN')` is used. This value also can be obtained by using the `INFO REPLICAT` command with the `DETAIL` option. The purpose of obtaining the BSN from Replicat is to get a recovery point for Extract in the event that a system failure or file system corruption makes the Extract checkpoint file unusable.

'REDOTHREAD'
Returns the thread number of a RAC node extract; on non-RAC node extracts the value is always 1. For data pump and Replicat, the thread id used by Extract capture of a RAC node is returned; on non-RAC, `@GETENV()` returns an error. Logdump shows the token, `ORATHREADID`, in the token section if the transaction is captured by Extract on a RAC node.

'PROGRAMNAME'
Name of the program or application that started the transaction or session.

'CLIENTIDENTIFIER'
Value set by using `DBMS_SESSION_.set_identifier()`.

'MACHINENAME'
Name of the host, machine, or server where database is running

'USERNAME'
Database login user name.

Example:

```
DB2 zOS:
TRANS-TIMESTAMP:         2017-10-31 06:21:07
TRANS-TIMESTAMP-PRECISE: 2017-10-31 06:21:07.485792
```

'OSVARIABLE' , '*variable*'

Valid for Extract and Replicat.

Use the `OSVARIABLE` option of `@GETENV` to return the string value of a specified operating-system environment variable.

Syntax

```
@GETENV ('OSVARIABLE', 'variable')
```

'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.

'TLFKEY' , SYSKEY, '*unique_key*'

Valid for Extract and Replicat.

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).

Syntax

```
@GETENV ('TLFKEY', SYSKEY, unique_key)
```

SYSKEY, *unique_key*
The NonStop node number of the source TLF/PTLF file. Do not enclose this syntax element in quotes.
Example:

```
GETENV ('TLFKEY', SYSKEY, 27)
```

'SOURCEDATABASEINFO'

This option has the DBUNIQUENAME and DBNAME fields. The fields from SOURCEDATABASEINFO are different from the GGFILEHEADER fields. Firstly, their performance is better as compared to the fields from GGFILEHEADER, so using SOURCEDATABASEINFO is a better alternative for scenarios where performance is critical. Secondly, when the DBUNIQUENAME token is not available in the trail header, either because the trail file was generated by an older version of Oracle GoldenGate, or because the database is not Oracle, @GETENV will treat DBUNIQUENAME as a synonym of DBNAME. In this case, a warning message will be written to the report file, the first time a header without the token is read.

GET_ENV_VALUE

Valid For

Extract and Replicat

Description

Use the GET_ENV_VALUE function to return information about the Oracle GoldenGate environment. The information that is supplied is the same as that of the @GETENV column-conversion function and is specified by using the same input values. For more information about the valid information types, environment variables, and return values, see "@GETENV".

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

Syntax

```
#include "usrdecs.h"
short result_code;
getenv_value_def env_ptr;
ERCALLBACK (GET_ENV_VALUE, &env_ptr, &result_code);
```

Buffer

```
typedef struct
{
char *information_type;
char *env_value_name;
char *return_value;
long max_return_length;
long actual_length;
short value_truncated;
} getenv_value_def;
```

Input

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 "@GETENV".

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 "@GETENV". 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

return_value
A valid return value for the supplied environment value.

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

Description

Use the GET_ERROR_INFO function to retrieve error information associated with a discard record. The user exit can use this information in custom error handling logic. For example, the user exit could send an e-mail message with detailed error information.

If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the message data that is exchanged between the user exit and the process is interpreted in the session character set.

Syntax

```
#include "usrdecs.h"
short result_code;
error_info_def error_info;
ERCALLBACK (GET_ERROR_INFO, &error_info, &result_code);
```

Buffer

```
typedef struct
{
long error_num;
char *error_msg;
long max_length;
long actual_length;
```

```
short msg_truncated;
} error_info_def;
```

Input

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

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

Description

Use the `GET_GMT_TIMESTAMP` function to retrieve the operation commit timestamp in GMT format. This function requires compiling with Version 2 `usrdecs.h` or later.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_GMT_TIMESTAMP, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
```

```
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input

None

Output

`timestamp`
The returned 64-bit I/O timestamp in GMT format.

`io_datetime`
A null-terminated string containing the local I/O date and time:
`YYYY-MM-DD HH:MI:SS.FFFFFF`
The format of the datetime string is in the session character set.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_OK
```

GET_MARKER_INFO

Valid For

Extract (data pump only) and Replicat

Description

Use the `GET_MARKER_INFO` function to retrieve marker information sent from a NonStop source system when Replicat is applying data. Use markers to trigger custom processing within a user exit.

If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, all of the returned marker data is interpreted in the session character set.

Syntax

```
#include "usrdecs.h"
short result_code;
marker_info_def marker_info;
ERCALLBACK (GET_MARKER_INFO, &marker_info, &result_code);
```

Buffer

```
typedef struct
{
char *processed;
char *added;
char *text;
char *group;
```

```
char *program;
char *node;
} marker_info_def;
```

Input

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

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.

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_OBJECT_NAME

Valid For

Extract and Replicat

Description

Use the GET_OBJECT_NAME function to retrieve the fully qualified name of a source or target object that is associated with the record being processed. This function is valid tables and other objects in a DML or DDL operation.

To return only part of the object name, see the following:

GET_OBJECT_NAME_ONLY GET_SCHEMA_NAME_ONLY GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_OBJECT_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

buffer
A pointer to a buffer to accept the returned object name. The name is null-terminated.

max_length
The maximum length of your allocated buffer to accept the object name. This is returned as a NULL terminated string.

source_or_target
One of the following indicating whether to return the source or target object name.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

buffer
The fully qualified, null-terminated object name, for example schema.object or catalog.schema.object, depending on the database platform.
If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the object name is interpreted in the session character set.

actual length
The string length of the returned object name. The actual length does not include the null terminator. The actual length is 0 if the object is a table.

value_truncated
A flag (`0` or `1`) indicating whether or not the value was truncated. Truncation occurs if the length of the object 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_OBJECT_NAME_ONLY

Valid For

Extract and Replicat

Description

Use the `GET_OBJECT_NAME_ONLY` function to retrieve the unqualified name (without the catalog, container, or schema) of a source or target object that is associated with the record that is being processed. This function is valid for tables and other objects in a DML or DDL operation.

To return the fully qualified name of an object, see the following:

GET_OBJECT_NAME

To return other parts of the object name, see the following:

GET_SCHEMA_NAME_ONLY GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_OBJECT_NAME_ONLY, &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

buffer
A pointer to a buffer to accept the returned object name. The name is null-terminated.

max_length
The maximum length of your allocated buffer to accept the object name. This is returned as a `NULL` terminated string.

source_or_target
One of the following indicating whether to return the source or target object name.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

buffer
The fully qualified, null-terminated object name, for example `schema.object` or `catalog.schema.object`, depending on the database platform.
If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the object name is interpreted in the session character set.

actual length
The string length of the returned object name. The actual length does not include the null terminator. The actual length is 0 if the object is a table.

value_truncated
A flag (`0` or `1`) indicating whether or not the value was truncated. Truncation occurs if the length of the object 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_OPERATION_TYPE

Valid For

Extract and Replicat

Description

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 "GET_RECORD_BUFFER").

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

`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

`io_type`
Returned as one of the following:

- **DDL type:**

  ```
  SQL_DDL_VAL
  ```

- **DML types:**

  ```
  DELETE_VAL
  INSERT_VAL
  UPDATE_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

Description

Use the `GET_POSITION` function is obtain a read position of an Extract data pump or Replicat in the Oracle GoldenGate trail.

Syntax

```
#include "usrdecs.h"
short result_code;
ERCALLBACK (GET_POSITION &position_def, &result_code);
```

Buffer

```
typedef struct
{
char *position;
long position_len;
short position_type;
short ascii_or_internal;
} position_def;
```

Input

`position_len`
Allocation length for the position length.

`position_type`
Can be one of the following:

`STARTUP_CHECKPOINT`
The start position in the trail.

`CURRENT_CHECKPOINT`
The position of the last read in the trail.

`column_value_mode`
An indicator for the format in which the column value was passed. Currently, only the default Oracle GoldenGate canonical format is supported, as represented by:
`EXIT_FN_RAW_FORMAT`

Output

`*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

Description

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.

You can use the `SET_RECORD_BUFFER` function (see "SET_RECORD_BUFFER") to modify the data retrieved with `GET_RECORD_BUFFER`. However, it requires an understanding of the data record as written in the internal Oracle GoldenGate canonical format. As an alternative, you can set column values in the data record with the `SET_COLUMN_VALUE_BY_INDEX` function (see "SET_COLUMN_VALUE_BY_INDEX") or the `SET_COLUMN_VALUE_BY_NAME` function (see "@STRNCMP").

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 ("DECOMPRESS_RECORD") to decompress the record for possible manipulation and then use the `COMPRESS_RECORD` function ("COMPRESS_RECORD") to compress it again, as expected by the process.

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

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

buffer
A pointer to the record buffer. Typically, buffer is a pointer to a buffer of type `exit_rec_buf_def`. The `exit_rec_buf_def` buffer contains the actual record about to be processed by Extract or Replicat. The buffer is supplied when the call type is `EXIT_CALL_DISCARD_RECORD`. Exit routines can change the contents of this buffer, for example, to perform custom mapping functions.
The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.

length
The returned length of the record buffer.

io_type
Returned as one of the following:

- **DDL type:**

  ```
  SQL_DDL_VAL
  ```

- **DML types:**

  ```
  DELETE_VAL
  INSERT_VAL
  UPDATE_VAL
  ```

- **Compressed 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.

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

Description

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

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

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

Description

Use the `GET_RECORD_TYPE` function to retrieve the type of record being processed. The record can be a SQL record. The record type is important when manipulating the record buffer, because each record type has a different format.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_RECORD_TYPE, &record, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input

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

record_type
The returned record type.

```
EXIT_REC_TYPE_SQL
```

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_SCHEMA_NAME_ONLY

Valid For

Extract and Replicat

Description

Use the `GET_SCHEMA_NAME_ONLY` function to retrieve the name of the owner (such as schema), but not the name of the catalog or container (if applicable) or the object, of the source or target object associated with the record being processed. This function is valid for DML and DDL operations.

To return the fully qualified name of a table, see the following:

GET_TABLE_NAME

To return the fully qualified name of a non-table object, such as a user, view or index, see the following:

GET_OBJECT_NAME

To return only the unqualified table or object name, see the following:

GET_TABLE_NAME_ONLY

GET_OBJECT_NAME_ONLY

To return other parts of the table or object name, see the following:

GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_SCHEMA_NAME_ONLY, &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

buffer
A pointer to a buffer to accept the returned schema name. The name is null-terminated.

max_length
The maximum length of your allocated buffer to accept the schema name. This is returned as a `NULL` terminated string.

source_or_target
One of the following indicating whether to return the source or target schema name.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

buffer
The fully qualified, null-terminated schema name.
If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the schema name is interpreted in the session character set.

actual_length
The string length of the returned name. The actual length does not include the null terminator.

value_truncated
A flag (`0` or `1`) indicating whether or not the value was truncated. Truncation occurs if the length of the schema name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_SESSION_CHARSET

Valid For

Extract and Replicat

Description

Use `GET_SESSION_CHARSET` to get the current user exit session character set. This character set can be set through callback function `SET_SESSION_CHARSET`. The character set of the user exit session indicates the encoding of any character-based callback structure members that are used between the user exit and the caller process (Extract, data pump, Replicat), including metadata such as (but not limited to):

- database names and locales
- table and column names
- DDL text
- error messages
- character-type columns such as `CHAR` and `NCHAR`
- date-time and numeric columns that are represented in string form

The valid values of the session character set are defined in the header file `ucharset.h`. This function can be called at any time that the user exit has control.

Syntax

```
#include usrdecs.h
short result_code;
session_def session_charset_def;
ERCALLBACK (GET_SESSION_CHARSET, &session_charset_def, &result_code);
```

Buffer

```
typedef struct
{
ULibCharSet  session_charset;
} session_def;
```

Input

None

Output

```
session_charset_def.session_charset
```

Return Values

```
EXIT_FN_RET_OK
```

GET_STATISTICS

Valid For

Extract and Replicat

Description

Use the GET_STATISTICS function to retrieve the current processing statistics for the Extract or Replicat process. For example, the user exit can output statistics to a custom report should a fatal error occur during Extract or Replicat processing.

Statistics are automatically handled based on which process type has requested the data:

- The Extract process will always treat the request as a source table, counting that table once regardless of the number of times output.
- The Replicat process will always treat the request as a set of target tables. The set includes all counts to the target regardless of the number of source tables.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include "usrdecs.h"
short result_code;
statistics_def statistics;
ERCALLBACK (GET_STATISTICS, &statistics, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
short group;
exit_timestamp_string start_datetime;
long num_inserts;
long num_updates;
long num_befores;
long num_deletes;
long num_discards;
long num_ignores;
long total_db_operations;
long total_operations;
/* Version 2 CALLBACK_STRUCT_VERSION */
long num_truncates;
} statistics_def;
```

Input

table_name
A null-terminated string specifying the fully qualified name of the source table. Statistics are always recorded against the source records. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the table name and the date are interpreted in the session character set.

group
Can be one of the following:

EXIT_STAT_GROUP_STARTUP
Retrieves statistics since the Oracle GoldenGate process was last started.

EXIT_STAT_GROUP_DAILY
Retrieves statistics since midnight of the current day.

EXIT_STAT_GROUP_HOURLY
Retrieves statistics since the start of the current hour.

EXIT_STAT_GROUP_RECENT
Retrieves statistics since the statistics were reset using GGSCI.

EXIT_STAT_GROUP_REPORT
Retrieves statistics since the last report was generated.

EXIT_STAT_GROUP_USEREXIT
Retrieves statistics since the last time the user exit reset the statistics with `RESET_USEREXIT_STATS`.

Output

start_datetime
A null-terminated string in the format of `YYYY-MM-DD HH:MI:SS` indicating the local date and time that statistics started to be recorded for the specified group.

num_inserts
The returned number of inserts processed by Extract or Replicat.

num_updates
The returned number of updates processed by Extract or Replicat.

num_befores
The returned number of update before images processed by Extract or Replicat.

num_deletes
The returned number of deletes processed by Extract or Replicat.

num_discards
The returned number of records discarded by Extract or Replicat.

num_ignores
The returned number of records ignored by Extract or Replicat.

total_db_operations
The returned number of total database operations processed by Extract or Replicat.

total_operations
The returned number of total operations processed by Extract or Replicat, including discards and ignores.

num_truncates
The returned number of truncates processed by Extract or Replicat.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_TABLE_NOT_FOUND
EXIT_FN_RET_OK
```

GET_TABLE_COLUMN_COUNT

Valid For

Extract and Replicat

Description

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

`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

`num_columns`
The returned total number of columns in the specified table.

`num_key_columns`
The returned total number of columns that are being used by Oracle GoldenGate as the key for the specified table.

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_TABLE_METADATA

Valid For

Extract and Replicat

Description

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

table_name
A pointer to a buffer to accept the table_name return value

key_columns
A pointer to an array of key_columns indexes.

max_name_length
The maximum length of the returned table name. Typically, the maximum length is the length of the table name buffer. Since the returned table name is null-terminated, the maximum length should equal the maximum length of the table name.

source_or_target
One of the following indicating whether to return the source or target table name.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

Output

table_name
The name of the table associated with the record that is being processed. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the table name is interpreted in the session character set.

value_truncated
A flag (0 or 1) indicating whether or not the value was truncated. Truncation occurs if the length of the table name plus the null terminator exceeds the maximum buffer length.

actual_name_length
The string length of the returned table name. The actual length does not include the null terminator.

num_columns
The number of columns in the table.

num_key_columns
The number of columns in the key that is being used by Oracle GoldenGate.

key_columns
The values for the key columns. You must know the expected number of keys multiplied by the length of the columns, and then allocate the appropriate amount of buffer.

num_keys_returned
The number of key columns that are requested.

using_pseudo_key
A flag that indicates whether or not `KEYCOLS`-specified columns are being used as a key. Returns `TRUE` or `FALSE`.

Return Values

```
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_EXCEEDED_MAX_LENGTH
EXIT_FN_RET_OK
```

GET_TABLE_NAME

Valid For

Extract and Replicat

Description

Use the `GET_TABLE_NAME` function to retrieve the fully qualified name of the source or target table associated with the record being processed. This function is valid only for tables in DML and DDL operations. To retrieve the fully qualified name of a non-table object, see the following:

GET_OBJECT_NAME

To return only part of the fully qualified name, see also the following:

GET_TABLE_NAME_ONLY GET_SCHEMA_NAME_ONLY GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

This function returns a value only if the object is a table. Otherwise, the `actual_length` of the `env_value_def` variable returns 0.

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

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

buffer
The fully qualified, null-terminated table name, for example `schema.table` or `catalog.schema.table`, depending on the database platform.
If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the table name is interpreted in the session character set.

actual length
The string length of the returned table name. The actual length does not include the null terminator. The actual length returned is 0 if the object is anything other than a table.

value_truncated
A flag (`0` or `1`) indicating whether or not the value was truncated. Truncation occurs if the length of the table name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

GET_TABLE_NAME_ONLY

Valid For

Extract and Replicat

Description

Use the GET_TABLE_NAME_ONLY function to retrieve the unqualified name (without the catalog, container, or schema) of the source or target table associated with the record being processed. This function is valid only for tables in DML and DDL operations. To retrieve the unqualified name of a non-table object, see the following:

GET_OBJECT_NAME_ONLY

To return the fully qualified name of a table, see the following:

GET_TABLE_NAME

To return other parts of the table name, see the following:

GET_SCHEMA_NAME_ONLY GET_CATALOG_NAME_ONLY

Database object names are returned exactly as they are defined in the hosting database, including the letter case.

This function returns a value only if the object is a table. Otherwise, the actual_length of the env_value_def variable returns 0.

Syntax

```
#include "usrdecs.h"
short result_code;
env_value_def env_value;
ERCALLBACK (GET_TABLE_NAME_ONLY, &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

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

buffer
The fully qualified, null-terminated table name, for example `schema.table` or `catalog.schema.table`, depending on the database platform.
If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the table name is interpreted in the session character set.

actual length
The string length of the returned table name. The actual length does not include the null terminator. The actual length returned is 0 if the object is anything other than a table.

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

Description

Use the `GET_TIMESTAMP` function to retrieve the I/O timestamp associated with a source data record in ASCII datetime format. The timestamp is then converted to local time and approximates the time of the original database operation.

> **Note:**
>
> The ASCII commit timestamp can vary with the varying regional use of Daylight Savings Time. The user exit callback should return the ASCII datetime as a GMT time to avoid this variance. The Oracle GoldenGate trail uses GMT format. See "GET_GMT_TIMESTAMP".

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (GET_TIMESTAMP, &record, &result_code);
```

Buffer

```
typedef struct
{
```

```
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input

None

Output

`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

Description

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

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

Description

Use the `GET_USER_TOKEN_VALUE` function to obtain the value of a user token from a trail record. No character-set conversion is performed on the token value.

Syntax

```
#include "usrdecs.h"
```

Buffer

```
typedef struct
{
char *token_name;
char *token_value;
long max_length;
long actual_length;
```

```
short value_truncated;
} token_value_def;
```

Input

token_name
A pointer to a buffer representing the name of a token. It is assumed that the token name is encoded in the default character set of the operating system that hosts the Extract `TABLE` statement where the token is configured. The user exit prepares the token name in the character set that is specified with `SET_SESSION_CHARSET`, but converts it back to the operating system character set before retrieving the matching token value.

max_length
The maximum length of your allocated `token_name` buffer to accept any resulting token value. This is returned as a `NULL` terminated string.

Output

token_value
A pointer to a buffer representing the return value (if any) of a token. The token value is passed back to the user exit as-is, without any character-set conversion.

actual_length
The actual length of the token value that is returned. A value of `0` is returned if the token is found and there is no value present.

value_truncated
A flag of either `0` or `1` that indicates whether or not the token value was truncated. Truncation occurs if the length of the table name plus the null terminator exceeds the maximum buffer length.

Return Values

```
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_TOKEN_NOT_FOUND
EXIT_FN_RET_OK
```

OUTPUT_MESSAGE_TO_REPORT

Valid For

Extract and Replicat

Description

Use the `OUTPUT_MESSAGE_TO_REPORT` function to output a message to the report file. If a character session for the user exit is set with `SET_SESSION_CHARSET`, the message is interpreted in the session character set but is converted to the default character set of the operating system before being written to the report file.

Syntax

```
#include "usrdecs.h"
short result_code;
char  message[500];
ERCALLBACK (OUTPUT_MESSAGE_TO_REPORT, message, &result_code);
```

Buffer

None

Input

message
A null-terminated string.

Output

None

Return Values

`EXIT_FN_RET_OK`

RESET_USEREXIT_STATS

Valid For

Extract and Replicat

Description

Use the `RESET_USEREXIT_STATS` function to reset the `EXIT_STAT_GROUP_USEREXIT` statistics for the Oracle GoldenGate process since the last call to `GET_STATISTICS` was processed. This function enables the user exit to control when to reset the group statistics that are returned by the `GET_STATISTICS` function, but does not permit any of the other statistics to be reset.

Syntax

```
#include "usrdecs.h"
short result_code;
call_callback (RESET_USEREXIT_STATS, NULL, &result_code);
```

Input

None

Output

None

Return Values

None

SET_COLUMN_VALUE_BY_INDEX

Valid For

Extract and Replicat

Description

Use the SET_COLUMN_VALUE_BY_INDEX or SET_COLUMN_VALUE_BY_NAME function to modify a single column value without manipulating the entire data record. If the character session of the user exit is set with SET_SESSION_CHARSET to a value other than the default character set of the operating system, as defined in ULIB_CS_DEFAULT in the ucharset.h file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (CHAR/VARCHAR2/CLOB, NCHAR/NVARCHAR2/NCLOB), a SQL date/timestamp/interval/number type)
- The column_value_mode indicator is set to EXIT_FN_CNVTED_SESS_CHAR_FORMAT.

Syntax

```
#include "usrdecs.h"
short result_code;
column_def column;
ERCALLBACK (SET_COLUMN_VALUE_BY_INDEX, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input

column_value
A pointer to a buffer representing the new column value.

actual_value_length
The length of the new column value, in bytes. The actual length should not include the null terminator if the new column value is in ASCII format.

null_value
A flag (0 or 1) indicating whether the new column value is null. If the null_value flag is set to 1, the column value in the data record is set to null.

remove_column
A flag (0 or 1) indicating whether to remove the column from a compressed update if it exists. A compressed update is one in which only the changed column values are logged, not all of the column values. This flag should only be set if the operation type for the record is UPDATE_COMP_SQL_VAL or PK_UPDATE_SQL_VAL.

column_index
The column index of the new column value to be copied into the data record buffer. Column indexes start at zero.

column_value_mode
Indicates the format of the column value.

EXIT_FN_CHAR_FORMAT
ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type UTF16_BE, which is converted to UTF8.)

> **Note:**
>
> A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of 0 becomes the string terminator.

- Dates are in the format CCYY-MM-DD HH:MI:SS.FFFFFF, in which the fractional time is database-dependent.
- Numeric values are in their string format. For example, 123.45 is represented as '123.45'.
- Non-printable characters or binary values are converted to hexadecimal notation.
- Floating point types are output as null-terminated strings, to the first 14 significant digits.

EXIT_FN_RAW_FORMAT
Internal Oracle GoldenGate canonical format: This format includes a two-byte null indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type.

EXIT_FN_CNVTED_SESS_CHAR_FORMAT
User exit character set: This only applies if the column data type is:

- a character-based type, single or multi-byte
- a numeric type with a string representation

This format is not null-terminated.

source_or_target
One of the following indicating whether the source or target record is being modified.

EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL

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

Description

Use the `SET_COLUMN_VALUE_BY_NAME` or `SET_COLUMN_VALUE_BY_INDEX` function to modify a single column value without manipulating the entire data record.

If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the character data that is exchanged between the user exit and the process is interpreted in the session character set.

A column value is set to the session character set only if the following is true:

- The column value is a SQL character type (`CHAR/VARCHAR2/CLOB`, `NCHAR/NVARCHAR2/NCLOB`), a SQL date/timestamp/interval/number type)
- The `column_value_mode` indicator is set to `EXIT_FN_CNVTED_SESS_CHAR_FORMAT`.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter.

Syntax

```
#include "usrdecs.h"
short result_code;
column_def column;
ERCALLBACK (SET_COLUMN_VALUE_BY_NAME, &column, &result_code);
```

Buffer

```
typedef struct
{
char *column_value;
```

```
unsigned short max_value_length;
unsigned short actual_value_length;
short null_value;
short remove_column;
short value_truncated;
short column_index;
char *column_name;
/* Version 3 CALLBACK_STRUCT_VERSION */
short column_value_mode;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION */
char requesting_before_after_ind;
char more_lob_data;
/* Version 3 CALLBACK_STRUCT_VERSION  */
ULibCharSet column_charset;
} column_def;
```

Input

`column_value`
A pointer to a buffer representing the new column value.

`actual_value_length`
The length of the new column value, in bytes. The actual length should not include the null terminator if the new column value is in ASCII format.

`null_value`
A flag (`0` or `1`) indicating whether the new column value is null. If the `null_value` flag is set to `1`, the column value in the data record is set to null.

`remove_column`
A flag (`0` or `1`) indicating whether to remove the column from a compressed update if it exists. A compressed update is one where only the changed column values are logged, not all of the column values. This flag should only be set if the operation type for the record is `UPDATE_COMP_SQL_VAL` or `PK_UPDATE_SQL_VAL`.

`column_name`
The name of the column that corresponds to the new column value to be copied into the data record buffer.

`column_value_mode`
Indicates the format of the column value.

`EXIT_FN_CHAR_FORMAT`
ASCII format: The value is a null-terminated ASCII (or EBCDIC) string (with a known exception for the sub-data type `UTF16_BE`, which is converted to UTF8.)

> **Note:**
>
> A column value might be truncated when presented to a user exit, because the value is interpreted as an ASCII string and is supposed to be null-terminated. The first value of `0` becomes the string terminator.

- Dates are in the format `CCYY-MM-DD HH:MI:SS.FFFFFF`, in which the fractional time is database-dependent.
- Numeric values are in their string format. For example, `123.45` is represented as `'123.45'`.
- Non-printable characters or binary values are converted to hexadecimal notation.
- Floating point types are output as null-terminated strings, to the first 14 significant digits.

`EXIT_FN_RAW_FORMAT`
Internal Oracle GoldenGate canonical format: This format includes a two-byte null indicator and a two-byte variable data length when applicable. No character-set conversion is performed by Oracle GoldenGate for this format for any character data type.

`EXIT_FN_CNVTED_SESS_CHAR_FORMAT`
User exit character set: This only applies if the column data type is:

- a character-based type, single or multi-byte
- a numeric type with a string representation

This format is not null-terminated.

`source_or_target`
One of the following indicating whether the source or the target data record is being modified.

```
EXIT_FN_SOURCE_VAL
EXIT_FN_TARGET_VAL
```

`requesting_before_after_ind`
Set when setting a column value on a record `io_type` of `UPDATE_COMP_PK_SQL_VAL` (primary key update). Use one of the following to indicate which portion of the primary key update is to be accessed. The default is `AFTER_IMAGE_VAL`.

- `BEFORE_IMAGE_VAL`
- `AFTER_IMAGE_VAL`

Output

None

Return Values

```
EXIT_FN_RET_BAD_COLUMN_DATA
EXIT_FN_RET_INVALID_COLUMN
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
EXIT_FN_RET_NOT_SUPPORTED
EXIT_FN_RET_INVALID_COLUMN_TYPE
```

SET_OPERATION_TYPE

Valid For

Extract and Replicat

Description

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

io_type
Returned as one of the following for deletes, inserts, and updates, respectively:

```
DELETE_VAL
INSERT_VAL
UPDATE_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)

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

Description

Use the `SET_RECORD_BUFFER` function for compatibility with user exits, and for complex data record manipulation. This function manipulates the entire record. It is best to modify individual column values, rather than the entire record, because the Oracle GoldenGate internal record formats must be known in order to accurately modify the data record buffer directly. To modify column values, use the `SET_COLUMN_VALUE_BY_INDEX` and `SET_COLUMN_VALUE_BY_NAME` functions. These functions are sufficient to handle most custom mapping within a user exit.

Syntax

```
#include "usrdecs.h"
short result_code;
record_def record;
ERCALLBACK (SET_RECORD_BUFFER, &record_def, &result_code);
```

Buffer

```
typedef struct
{
char *table_name;
char *buffer;
long length;
char before_after_ind;
short io_type;
short record_type;
short transaction_ind;
int64_t timestamp;
exit_ts_str io_datetime;
short mapped;
short source_or_target;
/* Version 2 CALLBACK_STRUCT_VERSION   */
char requesting_before_after_ind;
} record_def;
```

Input

`buffer`
A pointer to the new record buffer. Typically, `buffer` is a pointer to a buffer of type `exit_rec_buf_def`. The `exit_rec_buf_def` buffer contains the actual record about to be processed by Extract or Replicat. The buffer is supplied when the call type is `EXIT_CALL_DISCARD_RECORD`. Exit routines can change the contents of this buffer, for example to perform custom mapping functions.
The content of the record buffer is not converted to or from the character set of the user exit. It is passed as-is.

length
The new length of the record buffer.

Output

None

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
EXIT_FN_RET_NOT_SUPPORTED
```

SET_SESSION_CHARSET

Valid For

Extract and Replicat

Description

Use the `SET_SESSION_CHARSET` function to set the character set of the user exit. The character set of the user exit session indicates the encoding of any character-based callback structure members that are used between the user exit and the caller process (Extract, data pump, Replicat), including metadata such as (but not limited to):

- database names and locales
- table and column names
- DDL text
- error messages
- character-type columns such as `CHAR` and `NCHAR`
- date-time and numeric columns that are represented in string form

This function can be called at any time that the user exit has control. When the user exit sets the session character set, it takes effect immediately, and all character values start being converted to the specified set. The recommended place to call this function is with call type `EXIT_CALL_START`.

> **Note:**
>
> `SET_SESSION_CHARSET` is not thread-safe.

If `SET_SESSION_CHARSET` is not called, the session gets set to the default character set of the operating system, which is a predefined enumerated type value in `ULIB_CS_DEFAULT` in the `ucharset.h` file. When the session character set is a default from `ULIB_CS_DEFAULT`, no conversion is performed by Oracle GoldenGate for character-type values that are exchanged between the user exit and the caller process. In addition, the object-name metadata of the database are considered to be the default character set of the operating system. Keep in mind that the default may not be correct.

The character set of the user exit is printed to the report file when the user exit is loaded and when SET_SESSION_CHARSET is called. If the session character set is ULIB_CS_DEFAULT, there is a message stating that no column data character-set conversion is being performed.

Syntax

```
#include usrdecs.h
short result_code;
session_def session_charset_def;
ERCALLBACK (SET_SESSION_CHARSET, &session_charset_def, &result_code);
```

Buffer

```
typedef struct
{
ULibCharSet  session_charset;
} session_def;
```

Input

session_charset
The valid values of the session character set are defined in the header file ucharset.h.

Output

None

Return Values

```
EXIT_FN_RET_OK
```

SET_TABLE_NAME

Valid For

Extract and data pumps

Description

Use the SET_TABLE_NAME function to change the table name associated with a data record. For example, a delete on a specified table can be changed to an insert into a history table. You can change the table name only during Extract processing.

If the database is case-sensitive, object names must be specified in the same letter case as they are defined in the hosting database; otherwise, the case does not matter. Specify the full two-part or three-part table name.

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

`table_name`
A null-terminated string specifying the new table name to be associated with the data record. If the character session of the user exit is set with `SET_SESSION_CHARSET` to a value other than the default character set of the operating system, as defined in `ULIB_CS_DEFAULT` in the `ucharset.h` file, the table name is interpreted in the session character set.

Output

None

Return Values

```
EXIT_FN_RET_INVALID_CONTEXT
EXIT_FN_RET_INVALID_PARAM
EXIT_FN_RET_OK
```

5

Oracle GoldenGate Programs

This chapter describes the programs issued directly from the native command line of the Linux, UNIX, or Windows platforms.

> **Note:**
>
> Extract and Replicat typically, run from Admin Client. However, in some situations, such as initial load procedures, require running them from the command line of the operating system.

The following environment variables need to be set up from the bash prompt to be able to use command line utilities such as `defgen`:

- `OGG_HOME`: This value is usually present after Oracle GoldenGate installation. However, you can preset this value on the bash prompt. Example:

  ```
  export OGG_HOME=/u01/ogg
  ```

- `OGG_VAR_HOME`: This is the location in which the processing artifacts for logging and reporting, for each deployment are stored. Example:

  ```
  export OGG_VAR_HOME=/u02/Deployment/var
  ```

- `OGG_SERVICEMANAGER_ID`: This is the value of the Service Manager deployment ID which is used for encryption. Example:

  ```
  OGG_SERVICEMANAGER_ID=e5d29c3d-3d73-48be-95e6-b04eb9a1d407
  ```

  To obtain the Service Manager deployment ID, you can use either one of the following commands:

  ```
  strings /proc/$(pgrep adminsrvr)/environ | grep -E '(OGG_VAR_HOME|
  OGG_SERVICEMANAGER_ID)'
  ```

  or

  ```
  curl -su ggma:P@ssW0rd http://127.0.0.1:9011/services/v2/deployments/
  ServiceManager | jq .response.id,.response.oggVarHome
  ```

Topics:

checkprm

Use the `checkprm` program to assess the validity of the specified parameter file, with a configurable application and running environment. It can provide either a simple `PASS/FAIL` or with optional details about how the values of each parameter are stored and interpreted.

When you use `checkprm` and do not use any of these arguments, then `checkprm` attempts to automatically detect Extract or Replicat and the platform and database of the Oracle GoldenGate installation.

> **Note:**
>
> The options are not case-sensitive.

Syntax

```
checkprm
[None]
[-v]
[? | help]
[parameter_file]
[-COMPONENT | -C) component_name]
[-MODE | -M) mode_name]
[-PLATFORM | -P) platform_name]
[-DATABASE | -D) database _ame]
[-VERBOSE | -V)]
```

None
Displays usage information.

-v
Displays banner. Cannot be combined with other options. Does not produce verbose (`--VERBOSE | -V`) output.

? | help
Displays detailed usage information, include all possible values of each option. Cannot be combine with other options.

parameter_file
Specifies the name of the parameter file, has to be the first argument if a validation is requested. You can specify the relative path. For example, `CHECKPRM ./dirprm/myext.prm`.

--COMPONENT | -C *component_name*
Specifies the running component (application) that this parameter file is validated for. This option can be omitted for Extract or Replicat because automatic detection is attempted. Valid values include:

```
CACHEFILEDUMP COBGEN CONVCHK CONVPRM DDLCOB DEFGEN EMSCLNT EXTRACT GGCMD GGSCI KEYGEN
LOGDUMP
MGR OGGERR REPLICAT RETRACE
REVERSE SERVER GLOBALS
```

There is no default for this option.

--MODE | -M *mode_name*
Specifies the mode of the running application if applicable. This option is optional, only applicable to Extract or Replicat.
Valid input of this option includes:

- Integrated Extract
- Initial Load Extract
- Classic Replicat
- Coordinated Replicat
- Integrated Replicat
- Parallel Integrated Replicat
- Parallel Nonintegrated Replicat
- Special Run Replicat
- All

When key in the value for this option, the application name is optional, as long as it matches the value of component. For example, `"A Data Pump ExtractA"` is equivalent to `"A Data PumpA"` if the component is Extract. However, it is invalid if the component is Replicat.

--PLATFORM | -P *platform_name*
Specifies the platform the application is supposed to run on. The default value is the platform that this `checkprm` executable is running on.
The possible values are:

```
AIX HP-OSS HPUX-IT HPUX-PA
Linux OS400 ZOS Solaris SPARC
Solaris x86 Windows x64 All
```

--DATABASE | -D *database_name*
Specifies the database the application is built against. The default value is the database for your Oracle GoldenGate installation.
The database options are:

```
Generic Oracle 8 Oracle 9i
Oracle 10g Oracle 11g Oracle 12c
Oracle 18c
Oracle 19c
DB2LUW 9.5 DB2LUW 9.7
DB2LUW 10.5 DB2LUW 10.1 DB2 Remote
DB2LUW 11.1
Teradata
MySQL
DB2 for i
DB2 for i Remote
MS SQL
MS SQL CDC
DB2 z/OS
```

--VERBOSE | -V
Directs `checkprm` to display detailed parameter information, to demonstrate how the values are read and interpreted. It must be the last option specified in a validation.

defgen

> **Note:**
>
> Make sure to set the environment variables as described in Oracle GoldenGate Programs section, to be able to use the `defgen` utility.

Use `defgen` to run the `DEFGEN` utility from the command line of the Linux, UNIX, or Windows operating system. The `defgen` program is installed in the Oracle GoldenGate installation directory or library.

Syntax for Windows, UNIX, and Linux

```
defgen paramfile parameter_file
[CHARSET character_set]
[COLCHARSET character_set]
[noextattr]
[pauseatend | nopauseatend]
[reportfile report_file]
```

The following syntax can also be used without any other options:

```
defgen defs_file updatecs charset
```

defgen
Used without options, the command runs the program interactively.

paramfile *parameter_file*
Required. Specifies the relative or absolute path name of the parameter file for the `DEFGEN` program that is being run.

COLCHARSET *character_set*
Any supported character set. See COLCHARSET for more information.

noextattr
Can be used to support backward compatibility with Oracle GoldenGate versions that are older than Release 11.2.1 and do not support character sets other than ASCII, nor case-sensitivity or object names that are quoted with spaces. `NOEXTATTR` prevents `DEFGEN` from including the database locale and character set that support the globalization features that were introduced in Oracle GoldenGate Release 11.2.1. If the table or column name has multi-byte or special characters such as white spaces, `DEFGEN` does not include the table definition when `NOEXTATTR` is specified. If `APPEND` mode is used in the parameter file, `NOEXTATTR` is ignored, and the new table definition is appended in the existing file format, whether with the extra attributes or not.

pauseatend | nopauseatend
(Windows only) When the process stops, requires an Oracle GoldenGate user to look at the console output and then strike any key to clear it. Also indicates whether the process ended normally or abnormally.

reportfile *report_file*
Sends command output to the specified report file. Without the `reportfile` option, the command output is printed to the screen.

defs_file* updatecs *charset
Converts the character set of a definitions file to a different character set if the file is transferred to an operating system with an incompatible character set. This procedure takes the name of the definitions file and the targeted character set as input. For example: `defgen ./dirdef/source.def UPDATECS UTF-8`.
`updatecs` helps in situations such as when a Japanese table name on Japanese Windows is written in Windows CP932 to the data-definitions file, and then the definitions file is transferred to Japanese UNIX. The file cannot be used unless the UNIX is configured in PCK locale. Thus, you must use `updatecs` to convert the encoding of the definitions file to the correct format.

Example

Following is a sample `DEFGEN` parameter file, `defgenparam.prm`:

```
DEFSFILE /home/oracle/ogg/ora/bin/ora.defs,
PURGE useridalias oracle_source
table hr.employees;
```

Use the following command to call the `DEFGEN` program:

```
.$OGG_HOME/bin/defgen PARAMFILE defgenparam.prm
```

extract

Use `extract` to run the Extract program from the command line of the Linux, UNIX, or Windows operating system. The `extract` program is installed in the Oracle GoldenGate installation directory or library.

Syntax for Windows, UNIX, and Linux

```
extract paramfile parameter_file
[atcsn CSN | aftercsn CSN]
[initialdataload]
[pauseatend | nopauseatend]
[processid PID]
[reportfile report_file]
```

extract
Used without options, the command runs the program interactively.

paramfile *parameter_file*
Required. Specifies the relative or absolute path name of the parameter file for the Extract program that is being run. The default location is the `dirprm` subdirectory of the Oracle GoldenGate installation directory.

atcsn *CSN* | aftercsn *CSN*
Starts the process at or after the specified commit sequence number (CSN).

initialdataload
Runs Extract to extract all of the data records directly from the source database to support an initial load to the target.

pauseatend | nopauseatend
(Windows only) When the process stops, requires an Oracle GoldenGate user to look at the console output and then strike any key to clear it. Also indicates whether the process ended normally or abnormally.

processid *PID*
A name for the process. This name must match the name that is specified for the `EXTRACT` parameter in the parameter file. Use one alphanumeric word.

reportfile *report_file*
Sends command output to the specified report file. Without the `reportfile` option, the command output is printed to the screen. The default is the `dirrpt` subdirectory of the Oracle GoldenGate installation directory.

install

Use this program to:

- install event messages (into the registry) so they are displayed in the Windows Event Manager.
- install the Oracle GoldenGate Manager program as a Windows service.

The `install` program is installed in the Oracle GoldenGate installation directory or library.

By default, the Manager service is installed to start automatically at system boot time. This can be changed by specifying the `MANUALSTART` option.

In addition, install can be used to de-install event messages and the Manager service.

Installation of event messages and the Manager program as a service is recommended. For example:

```
install item [item...]
```

In this command *item* is one of `addevents | deleteevents addservice | deleteservice autostart | manualstart` and the user credentials:

ADDEVENTS
Adds Oracle GoldenGate events. By default, the errors logged are generic. To display specific content, including the user name and process, the parameter file name, and the error text, copy the `category.dll` and `ggsmsg.dll` files from the installation directory to the `SYSTEM32` directory.

DELETEEVENTS
Deletes Oracle GoldenGate events from the registry.

ADDSERVICE
Adds the Manager program as a Windows service named `GGSMGR` (default) or a name specified in a GLOBALS file. Create GLOBALS as a text file (uppercase name, no file

extension) in the installation directory, and specify the service name with the MGRSERVNAME parameter before running install.

```
MGRSERVNAME service name>
```

DELETESERVICE
Removes the Oracle GoldenGate Manager service.

AUTOSTART
If ADDSERVICE is used, specifies that the service starts at system boot time (the default).

MANUALSTART
If ADDSERVICE is used, specifies that the service starts only at user request (through GGSCI or the Services applet of the Control Panel). By default, the Manager service starts at system boot time. If ADDSERVICE is used, this adds the Manager program as an interactive Windows service.

USER *specification*
Specifies a user name for executing Manager. For *specification*, include the domain name, a backward slash, and the user name., For example, HEADQT\GGSMGR.

PASSWORD *password*
Specifies the user name password for the USER executing the Manager service. The password must be listed within double quotes.

WAITFORSERVICE *service name*
Specifies a service that the Manager service should wait on before starting. The server name must not contain spaces and can be obtained from the Windows Service Manager applet.

```
install.exe addservice addevents user hostname\oggmgr password "123abc"
waitforservice MSSQL$SQL2008R2
```

keygen

Use keygen to generate one or more encryption keys to use with Oracle GoldenGate security features that use an ENCKEYS file. The key values are returned to your screen. You can copy and paste them into the ENCKEYS file.

Syntax

```
KEYGEN key_length n
```

keygen
Used without options, the command runs the program interactively.

key_length
The length of the encryption key, up to 256 bits (32 bytes).

n
The number of keys to generate.

logdump

Use `logdump` to run the Logdump utility. This program takes no arguments and runs interactively. For more information about the Logdump utility, see *Logdump Reference for Oracle GoldenGate*.

Syntax for Windows, UNIX, and Linux

```
logdump
```

replicat

Use `replicat` to run the Replicat program from the command line of the Linux, UNIX, or Windows operating system. The `replicat` program is installed in the Oracle GoldenGate installation directory or library.

Syntax for Window, UNIX, and Linux

```
replicat paramfile parameter_file
[{atcsn CSN | aftercsn CSN} [threads(thread_list)]]
[filterduptransactions]
[initialdataload]
[pauseatend | nopauseatend]
[processid PID]
[reportfile report_file]
[skiptransaction [threads(thread_list)]]
```

replicat
Used without options, the command runs the program interactively.

paramfile *parameter_file*
Specifies the relative or absolute path name of the parameter file for the Replicat program that is being run. The default location is the `dirprm` subdirectory of the Oracle GoldenGate installation directory.

atcsn *CSN* | aftercsn *CSN* [threads(*thread_list*)]
Starts the process at or after the specified commit sequence number (CSN). F.

filterduptransactions
Causes Replicat to ignore transactions that it has already processed.

initialdataload
Runs Replicat to apply all of the data as an initial load to populate the target.

pauseatend | nopauseatend
(Windows only) When the process stops, requires an Oracle GoldenGate user to look at the console output and then strike any key to clear it. Also indicates whether the process ended normally or abnormally.

processid *PID*
A name for the process. This name must match the name that is specified for the `REPLICAT` parameter in the parameter file. Use one alphanumeric word.

reportfile *report_file*
Sends command output to the specified report file. Without the `reportfile` option, the command output is printed to the screen. The default is the `dirrpt` subdirectory of the Oracle GoldenGate installation directory.

skiptransaction [threads(*thread_list*)]
Causes the process to skip the first transaction after its expected startup position in the trail.