Oracle® GoldenGate for Flat File

Administrator's Guide

11*g* Release 1 (11.1.1)

E17342-01

August 2010

Oracle GoldenGate Oracle GoldenGate for Flat File 11*g* Release 1 (11.1.1)

E17342-01

Contents

Chapter 1 Introduction 4

Oracle GoldenGate 4

Adapter integration options 4

Capturing transactions to a trail 4

Applying transactions from a trail 5

Oracle GoldenGate for Flat File 5

Oracle GoldenGate documentation 6

Chapter 2 Installing and Configuring Oracle GoldenGate for Flat File 7

Installing 7

Configuring 7

User exit Extract parameters 8

User exit properties 9

Recommended data integration approach 9

Operation 9

Producing data files 9

Working with control files 10

Working with statistical summaries 11

Managing Oracle GoldenGate processes 11

Handling errors 11

Troubleshooting 12

Chapter 3 Properties 13

Using properties 13

Logging properties 13

log.logname 13

log.level 13

log.tostdout 14

log.tofile 14

log.modules, log.level.{module} 14

General properties 14

goldengate.flatfilewriter.writers 14

goldengate.userexit.buffertxs 15

goldengate.userexit.chkptprefix 15

goldengate.userexit.chkpt.ontxend 15

goldengate.userexit.datetime.removecolon 15
goldengate.userexit.timestamp 15
goldengate.userexit.datetime.maxlen 16
Output format properties 16
mode 16
groupcols 16
Output file properties 17
files.onepertable 17
files.oneperopcode 17
files.prefix 17
files.data.rootdir, files.data.ext, files.data.tmpext 17
files.control.use, files.control.rootdir, files.control.ext. 18
files.formatstring 18
files.data.bom.code 19
File rollover properties. 19
files.data.rollover.time 19
files.data.rollover.size 19
files.data.norecords.timeout. 19
files.rolloveronshutdown 19
files.data.rollover.timetype 20
files.data.rollover.multiple 20
files.data.rollover.attime 20
csvwriter.writebuffer.size 20
Data content properties 21
rawchars. 21
includebefores 21
afterfirst 21
includecolnames 21
omitvalues 21
diffsonly 22
omitplaceholders 22
Metadata columns 22
Valid metadata columns 22
Using metadata columns 23
metacols. 23
metacols.{metacol-name}.fixedlen 23
metacols.{metacol-name}.column. 24
metacols.{token-name}.novalue.chars/code 24
metacols.{metacol-name}.fixedjustify 24

metacols.{metacol-name}.fixedpadchar.chars/code ... 24
metacols.opcode.insert.chars/code, metacols.opcode.update.chars/code, metacols.opcode.delete.chars/code ... 24
metacols.txind.begin.chars/code, metacols.txind.middle.chars/code metacols.txind.end.chars/code, metacols.txind.whole.chars/code ... 24
metacols.position.format ... 25
metacols.{COLNAME}.omit ... 25
begintx.metacols, endtx.metacols ... 25
DSV specific properties ... 26
dsv.nullindicator.chars/code ... 26
dsv.fielddelim.chars/code ... 26
dsv.linedelim.chars/code ... 27
dsv.quote.chars/code ... 27
dsv.quotes.policy ... 27
dsv.quotes.policy.datatypes ... 27
dsv.nullindicator/fielddelim/linedelim/quotes.escaped.chars/code ... 28
dsv.onecolperline ... 28
dsv.quotealways ... 28
LDV specific properties ... 28
ldv.vals.missing.chars/code, ldv.vals.present.chars/code, ldv.vals.null.chars/code 29
ldv.lengths.record.mode, ldv.lengths.field.mode ... 29
ldv.lengths.record.length, ldv.lengths.field.length ... 29
Statistics and reporting ... 29
statistics.toreportfile ... 30
statistics.period ... 30
statistics.time ... 30
statistics.tosummaryfile ... 30
statistics.summary.fileformat ... 30
statistics.overall ... 31
statistics.summary.delimiter.chars/code, statistics.summary.eol.chars/code ... 31
statistics.summary.extension ... 31
Chapter 4 Property Templates ... 32
Using property templates ... 32
{writer}.template={template_name} ... 32
Default properties ... 32
Siebel Remote ... 33
Ab Initio ... 34
Netezza ... 34
Greenplum ... 35
Comma Delimited ... 35

CHAPTER 1

Introduction

This guide covers installing, configuring and running Oracle GoldenGate for Flat File.

Oracle GoldenGate

The core Oracle GoldenGate product:

- Captures transactional changes from a source database. For most databases this is accomplished by reading the database transaction log.
- Sends and queues these changes as a set of database-independent files called the Oracle GoldenGate trail.
- Optionally alters the source data using mapping parameters and functions.
- Applies the transactions in the trail to a target system database

Oracle GoldenGate performs this capture and apply in near real-time across heterogeneous databases and operating systems.

Adapter integration options

The Oracle GoldenGate adapters integrate with installations of the Oracle GoldenGate core product to either 1) read JMS messages and deliver them as an Oracle GoldenGate trail, 2) read an Oracle GoldenGate trail and deliver transactions to a JMS provider or other messaging system or custom application or 3) read an Oracle GoldenGate trail and write transactions to a flat file that can be used by other applications.

Capturing transactions to a trail

The Oracle GoldenGate message capture adapter can be used to read messages from a queue and communicate with an Oracle GoldenGate Extract process to generate a trail containing the processed data.

The message capture adapter is implemented as a Vendor Access Module (VAM) plug-in to a generic Extract process. A set of properties, rules and external files provide messaging connectivity information and define how messages are parsed and mapped to records in the target GoldenGate trail.

Currently this adapter supports capture from JMS text messages.

Applying transactions from a trail

The Oracle GoldenGate delivery adapters can be used to apply transactional changes to targets other than a relational database: for example, ETL tools (DataStage, Ab Initio, Informatica), JMS messaging, or custom APIs. There are a variety of options for integration with Oracle GoldenGate:

- *Flat file integration*: predominantly for ETL, proprietary or legacy applications, Oracle GoldenGate for Flat File can write micro batches to disk to be consumed by tools that expect batch file input. The data is formatted to the specifications of the target application such as delimiter separated values, length delimited values, or binary. Near real-time feeds to these systems are accomplished by decreasing the time window for batch file rollover to minutes or even seconds.
- *Messaging*: transactions or operations can be published as messages (e.g. in XML) to JMS. The JMS provider is configurable; examples include ActiveMQ, JBoss Messaging, TIBCO, WebLogic JMS, WebSphere MQ and others.
- *Java API*: custom event handlers can be written in Java to process the transaction, operation and metadata changes captured by Oracle GoldenGate on the source system. These custom Java handlers can apply these changes to a third-party Java API exposed by the target system.

All three options have been implemented as extensions to the core Oracle GoldenGate product using Oracle GoldenGate's user exit interface, a C API.

- For the flat file integration, Oracle GoldenGate for Flat File provides filewriter libraries (implemented natively in C) that can be dynamically linked into the Oracle GoldenGate Extract process. No programming is required and they can be customized using a properties file.
- For Java integration using either JMS or the Java API, use Oracle GoldenGate for Java.

This guide describes how to use Oracle GoldenGate for Flat File.

Oracle GoldenGate for Flat File

Oracle GoldenGate for Flat File is used to output transactional data captured by Oracle GoldenGate to rolling flat files to be consumed by a third party product.Oracle GoldenGate for Flat File is implemented as a user exit provided as a shared library (.so or .dll) that integrates into the Oracle GoldenGate Extract process.

The user exit supports two modes of output:

- DSV – Delimiter Separated Values (commas are an example)
- LDV – Length Delimited Values

It can output data:

- All to one file
- One file per table
- One file per operation code

The user exit can roll over based on time and/or size criteria and flushes files and maintains checkpoints whenever Oracle GoldenGate checkpoints to ensure recovery. It

writes a control file containing a list of rolled over files for synchronization with the supported data integration product and can also produce a summary file for use in auditing. Additional properties control formatting (delimiters, other values), directories, file extensions, metadata columns (such as table name, file position, etc.) and data options.

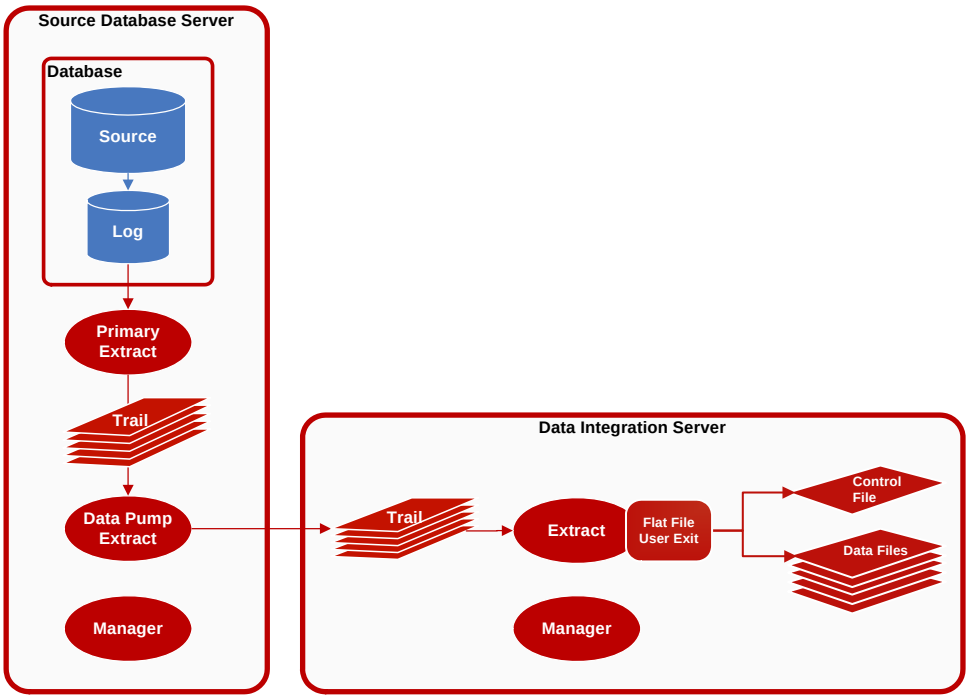


Figure 1 Oracle GoldenGate for Flat File

Oracle GoldenGate documentation

For information on installing and configuring the core Oracle GoldenGate software for use with the Oracle GoldenGate Flat File or Java adapters, see the Oracle GoldenGate documentation:

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

CHAPTER 2

Installing and Configuring Oracle GoldenGate for Flat File

Installing

The Oracle GoldenGate for Flat File comes prebuilt for a particular platform.

The core Oracle GoldenGate should be fully installed and tested prior to installation of the Oracle GoldenGate for Flat Files user exit code. Both installations must be on a Windows, UNIX or Linux operating system. The user should have the necessary file permissions to write output data to the required output directories.

Oracle GoldenGate for Flat File is shipped:

- On Windows as a zip file
- On UNIX as a .tar.gz file

To install, unzip or gunzip; tar xvf the file in the Oracle GoldenGate install directory. The file contains:

- Shared library (flatfilewriter.dll on windows, flatfilewriter.so on UNIX)
- Sample user exit properties file (ffwriter.properties)
- Sample Extract parameter file (ffwriter.prm)

Configuring

Figure 2 “Typical Configuration” on page 8 shows a typical Oracle GoldenGate for Flat File configuration.

Source database system configuration:

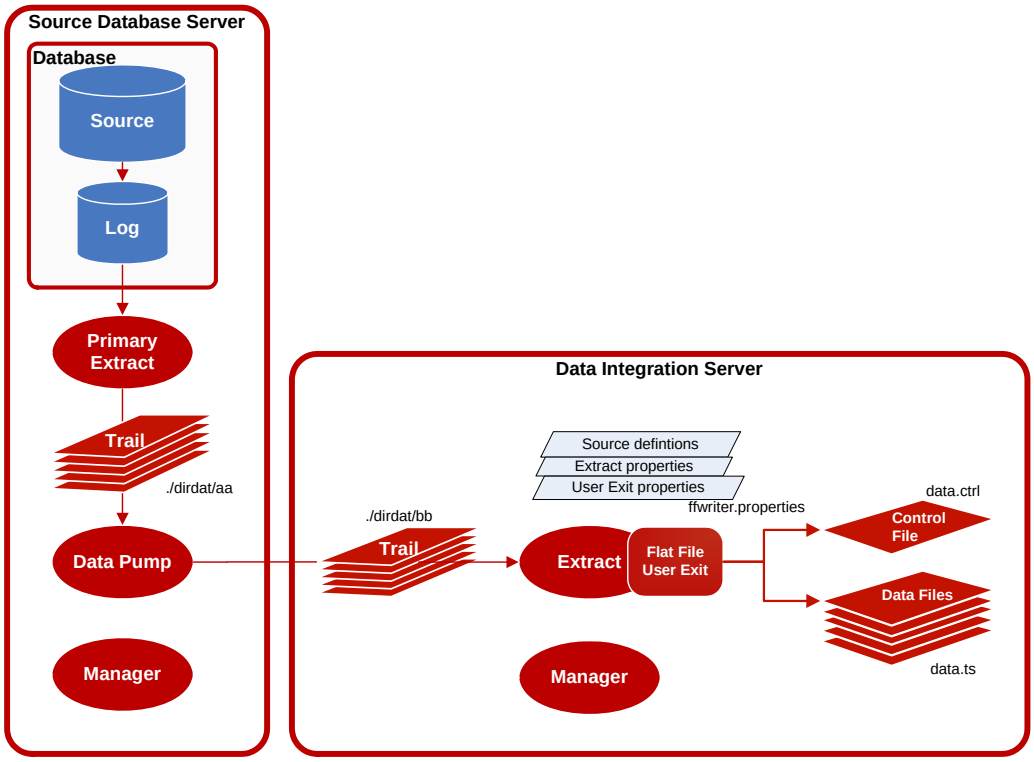
```
GGSCI > ADD EXTRACT pump, EXTTRAILSOURCE ./dirdat/aa
GGSCI > ADD RMTTRAIL ./dirdat/bb, EXTRACT pump, MEGABYTES 20
```

Data integration system configuration:

```
GGSCI > ADD EXTRACT ffwriter, EXTTRAILSOURCE ./dirdat/bb
```

The sample process names and trail names used above can be replaced with any valid name. Process names need to be 8 characters or less, trail names need to be two characters.

Figure 2 Typical Configuration



User exit Extract parameters

The user exit Extract parameters (ffwriter.prm) are as follows.

Parameter	Description
`EXTRACT FFWRITER`	All Extract parameter files start with the Extract name. In this case it is the user exit's file writer name.
`SOURCEDEFS .\dirdef\hr_ora.def`	A source definitions file to determine trail contents.
`CUSEREXIT .\flatfilewriter.dll CUSEREXIT PASSTHRU, INCLUDEUPDATEBEFORES, PARAMS ffwriter.properties`	The `CUSEREXIT` parameter options: `.\flatfilewriter.dll` is the location of the user exit .dll or .so library, in this case. `CUSEREXIT` is the name of the user exit routine that will be invoked. `PASSTHRU` specifies that the Extract process does not need to write a trail. `INCLUDEUPDATEBEFORES` allows both the before and after image to be included in the output. It is also required for consistency purposes and transaction tracking. `PARAMS` allows you to specify the name of the user exit properties file.
`TABLE HR.*;`	Specifies a list of tables to process

User exit properties

The user exit reads properties from a file named as an option to CUSEREXIT PARAMS. The default is to read from ffwriter.properties.

The properties file contains details of how the user exit should operate. For more information on individual properties see "Properties" on page 13.

Recommended data integration approach

To take best advantage of the micro-batch capabilities, customers should do the following in their data integration tool:

1. Wait on the control file
2. Read list of files to process from the control file
3. Rename the control file
4. Iterate over the comma-delimited list of files read from the control file
5. Process each data file, deleting the data file when complete
6. Delete the renamed control file

On startup, the data integration tool should check for the renamed control file to see if it needs to recover from previously failed processing

When the control file is renamed, the user exit will write a new one on the first file rollover, which will contain the list of files for the next batch.

If the user exit has been configured to also output a summary file, the data integration tool can optionally also read that summary file and cross-check the number of operations it has processed with the data in the summary file for each processed data file.

Operation

Producing data files

Data files are produced by configuring a *writer* in the user exit properties. A single user exit properties file can have multiple writers, which allows for the generation of multiple differently formatted output data files for the same input data.

Writers are added by name to the `goldengate.flatfilewriter.writers` property. For example:

```
goldengate.flatfilewriter.writers=dsvwriter,diffswriter,binarywriter
```

The remainder of the properties file contains detailed properties for each of the named writers where the properties are prefixed by the writers name. For example:

```
dsvwriter.files.onepertable=true
binarywriter.files.onepertable=false
binarywriter.files.oneperopcode=true
```

Each writer can output all the data to a single (rolling) data file, or produce one (rolling) data file per input table or operation type. This is controlled by the `files.onepertable` and `files.oneperopcode` properties as shown in the example above.

The data written by each writer can be in one or two output formats controlled by the mode property. This can either be:

- DSV – Delimiter Separated Values
- LDV – Length Delimited Values

For example:

```
dsvwriter.mode=dsv
binarywriter.mode=ldv
```

When data files are first written to disk, they have a temporary extension. Once the file meets rollover criteria, the extension is switched to the rolled extension. If control files are used, the final file name is added to the list in the control file, creating the control file if necessary. Also, if a file level statistics summary is being generated, it will be created on rollover of the file.

The output directory (for data files and control files separately), temporary extension, rolled extension, control extension and statistical summary extension can all be configured through properties. For output configuration details see “Output file properties” on page 17.

Each data file written follows a naming convention which depends on the output style. For files written one per table, the name includes the table name, for example:

```
MY.TABLE_2007-08-03_11:30:00_data.dsv
```

For files written with all data in one file, the name does not include the table name, for example:

```
output_2007-08-03_11:30:00_data.dsv
```

In addition to the basic data contents, additional *metadata columns* can be added to the output data to aid in data consumption. This includes the schema (owner) and table information, source commit timestamp, Oracle GoldenGate read position and more. For a detailed description of metadata columns see “Metadata columns” on page 22.

The contents of the data file depend on the mode, the input data, and the various properties determining which (if any) metadata columns are added, whether column names are included, whether before images are included etc. For full details of all properties governing the output data see section “Data content properties” on page 21.

Working with control files

Control files contain information on which data files have rolled over. If the control file exists, it will be appended to, if it does not exist it will be created. For writers that output all data to one file, a single control file will be created. If the writer is outputting to one file per table or operation type, a control file will also be created per table or operation type.

The generation of a control file, its output directory, prefix and extension are controlled by the properties defined in “Output file properties” on page 17.

Each control file will contain a comma-delimited list of data files that have been rolled over since the control file was created, in the order that the files were rolled over. This allows Data Integration tools to ensure that data files are read in the correct order and that they have all been consumed.

Working with statistical summaries

In addition to control files, summary statistics about the data production process can also be obtained. This statistically summary information can be written to the Oracle GoldenGate report file and/or individual summary files.

When writing to the report file, the user can decide if this information should be written when files are rolled over, or periodically based on a time period. Information written to the report file is output in a standard fashion, and contains total records, totals for each database operation type, deltas since the last report, rate information, and detail information for each table.

When writing to summary files, a summary file will be created for each rolled over file. This file contains information pertaining to that particular file in a delimited fashion. The extension of this file, the data to be output, the delimiter and line delimiter can all be controlled.

"Statistics and reporting" on page 29 contains detailed property information about statistics and summary files.

Managing Oracle GoldenGate processes

The processes involved in a typical data integration solution include:

- A primary Extract process, capturing transactional data from the source database
- A PASSTHRU data pump Extract moving the captured transactional data across the network from the source database machine to the data integration machine
- A delivery data pump Extract configured to run the user exit

Typically, the original capture and PASSTHRU data pump are part of one Oracle GoldenGate installation and the delivery data pump is part of a second installation. Both of these installations will also need to have an Oracle GoldenGate Manager process running.

Processes within these installations are managed through the Oracle GoldenGate GGSCI command line with simple commands like start and stop. Full details of managing these processes and their configuration can be found in the *Oracle GoldenGate Administration Guide*.

Handling errors

There are three types of errors that could occur in the operation of the Oracle GoldenGate for Flat File:

1. The Extract process running the user exit does not start
2. The process starts, but abends at some point later
3. The process runs successfully, but the data is incorrect or non-existent

In the first two cases, there are a number of places to look for error messages:

- The standard ggserr.log file, which contains basic information about Oracle GoldenGate processes, their run history and a brief error message if any error occurred.

- The Oracle GoldenGate report file for the Extract process running the user exit, found in the dirrpt subdirectory. For example, if the process name is `ffwriter`, the report file would be ffwriter.rpt. This may contain more detailed information about the error, especially if it is a problem in the Oracle GoldenGate core product rather than the user exit.
- In the user exits log file, the name of which depends on the log.logname property. If this file does not exist, the user exit most likely did not start up and the report file should help isolate that problem.

"Troubleshooting" on page 12 contains more information on error handling.

Troubleshooting

Before checking for specific issues related to the Oracle GoldenGate for Flat File, ensure that Oracle GoldenGate is configured correctly and any standard Oracle GoldenGate errors have been resolved. For further information, see the *Oracle GoldenGate Troubleshooting and Performance Tuning Guide*.

In addition, check the following:

1. Is the shared library (.so or .dll) in the Extract parameter file correct? Is it in the path specified and accessible?
2. Is the `SOURCEDEFS` file specified in the Extract parameter file correct? Is it in the specified path and accessible?
3. Does the `SOURCEDEFS` file contain all the necessary tables?
4. Is the ffwriter.properties user exit properties file in the Oracle GoldenGate install directory, or does it have the correct name and path specified in the GG_USEREXIT_PROPFILE environment variable?
5. Do the output directories specified in the user exit properties file exist?
6. Are file permissions correct to write to that directory?

Now check the user exit log file: logname_yyyymmdd.log:

7. Does the user exit properties file parse successfully? Are any invalid properties mentioned in the log file?
8. Are any other errors or warnings in the log?

If the problem is still not resolved:

9. Set log.level=DEBUG
10. Restart and save the log file

Before contacting Oracle GoldenGate Support, be prepared to send the log file, source trail file, sourcedefs file, user exit properties file and Extract parameter file, together with any data files that have been written.

CHAPTER 3

Properties

Using properties

All properties in the property file are of the form:

```
fully.qualified.name=value
```

The value may be single or comma-delimited strings, an integer, or a boolean value. Comments can be entered in the properties file with the # prefix at the beginning of the line. For example:

```
# This is a property comment
some.property=value
```

Properties themselves can also be commented, which allows testing configurations without losing previous property settings.

Logging properties

Logging is controlled by the following properties.

log.logname

Specifies the prefix to the log file name. This must be a valid ASCII string. The log file name has the current date appended to it, in yyyymmdd format, together with the .log extension.

The following example will create a log file of name writer_20100803.log on August 3, 2010. The log file will roll over each day, independent of the stopping and starting of the process.

```
# log file prefix
log.logname=writer
```

The following example will create a log file of name msgv_20100803.log on August 3, 2010.

```
# log file prefix
log.logname=msgv
```

log.level

Specifies the overall log level for all modules. The syntax is:

```
log.level=ERROR|WARN|INFO|DEBUG
```

The log levels are defined as follows:

ERROR – Only write messages if errors occur

WARN – Write error and warning messages

INFO – Write error, warning and informational messages

DEBUG – Write all messages, including debug ones.

The default logging level is INFO. The messages in this case will be produced on startup, shutdown and periodically during operation. If the level is switched to DEBUG, large volumes of messages may occur which could impact performance. For example, the following sets the global logging level to INFO:

```
# global logging level
log.level=INFO
```

log.tostdout

Controls whether or not log information is written to standard out. This setting is useful if the Extract process is running with a VAM started from the command line or on an operating system where `stdout` is piped into the report file. Generally Oracle GoldenGate processes run as background processes however.

The syntax is:

```
goldengate.log.tostdout=true|false
```

The default is false.

log.tofile

Controls whether or not log information is written to the specified log file.The syntax is:

```
log.tofile=true|false
```

The default is false. Log output is written to the specified log file when set to true.

log.modules, log.level.{module}

Specifies the log level of the individual source modules that comprise the user exit. This is typically used only for advanced debugging. It is possible to increase the logging level to DEBUG on a per module basis to help troubleshoot issues. The default levels should not be changed unless asked to do so by Oracle GoldenGate support.

General properties

goldengate.flatfilewriter.writers

Specifies the name of the writer that will run within the user exit. Enter multiple string values to enable multiple named writers to run within the same user exit.

For example:

```
goldengate.flatfilewriter.writers=dsvwriter,diffswriter,binwriter
```

Ensure there are no spaces before or after the equal sign or the commas. All other

properties in the file should be prefixed by one of the writer names.

goldengate.userexit.buffertxs

Controls whether entire transactions are read before being output.When set to true, an entire transaction is read from the trail before being output. For example:

```
goldengate.userexit.buffertxs=true
```

The default is false. Setting this to true is useful only if the `numops` metadata column is used. Currently the only way to calculate the `numops` value is to buffer transactions and output one transaction at a time.

goldengate.userexit.chkptprefix

Specifies a string value as the prefix to be added to the checkpoint file name. When running multiple data pumps the checkpoint prefix should be set to the name of the process. For example:

```
goldengate.userexit.chkptprefix=pump1_
```

goldengate.userexit.chkpt.ontxend

Controls whether the need to roll files over is checked after every transaction or only when the Extract process checkpoints. If set to `true` the adapter checks if a file is due to be rolled over after it has processed a transaction. If due, the rollover is performed and the checkpoint file updated. This is useful if tight control over the contents of output files is required. For example, if all data up to midnight should be written to files before rolling over at midnight, it is important that the check occurs on every transaction. For example:

```
goldengate.userexit.chkpt.ontxend=true
```

The default is false.If set to `false` the adapter will only check for rollover when Extract checkpoints (every 10 seconds by default).

goldengate.userexit.datetime.removecolon

Controls whether or not a colon is written between the date and time. When set to false, the date and time column values are written to the output files in the default format of the Oracle GoldenGate trail, YYYY-MM-DD:HH:MI:SS.FFFF. When set to true, the format is changed to YYYY-MM-DD HH:MI:SS.FFF with no colon between date and time. The default is false.

```
goldengate.userexit.datetime.removecolon=true
```

goldengate.userexit.timestamp

Controls whether the record timestamp is output as local time or Coordinated Universal Time (UTC). When this is not set to `utc` the record timestamp is output as local time using the local time zone. The default is local.

```
goldengate.userexit.timestamp=utc
```

goldengate.userexit.datetime.maxlen

Controls the maximum output length of a datetime column. Setting this to an integer value truncates the column value to that length. Since the date and time format is YYYY-MM-DD:HH:MI:SS.F(9) the maximum length of a date and time column is 29 characters.

For example:

```
goldengate.userexit.datetime.maxlen=19
```

Setting `goldengate.userexit.maxlen=19` truncates to date and time with no fractional seconds. Setting `goldengate.userexit.maxlen=10` truncates to date only. The default is to output the full date and time column value.

Output format properties

mode

Controls whether the output format is DSV or LDV.

- **DSV** – Delimiter Separated Values, for example:
  `POSITIONÇOPCODEÇTIMESTAMPÇCOLVALAÇCOLVALBÇETC`

  NOTE This is not limited to comma separated values (CSV), so the delimiter does not have to be a comma.

- **LDV** – Length Delimited Values, for example:
  `0109TIMESTAMPI302MY05TABLEP042000P03ETC`

  NOTE Lengths can be ASCII or binary, some metadata columns can be fixed length (see “Metadata columns” on page 22) and this format will support unicode multi-byte data.

  For example:

  ```
  dsvwriter.mode=dsv
  binarywriter.mode=ldv
  ```

  NOTE For backward compatibility, csv is accepted instead of dsv, binary instead of ldv. There is no difference in the output formats when using the alternate options.

groupcols

Controls whether or not the column names, before values and after values are grouped together.

The syntax is:

```
{writer}.groupcols=true|false
```

The default is false. This results in a set of name, before value and after value listed together, as shown in this example for COL1 and COL2:

```
“COL1”, COL1_B4, COL1, “COL2”, COL2_B4, COL2
```

With the property set to `true`, the columns are grouped into sets of all names, all before

values, and all after values:

```
"COL1", "COL2", COL1_B4, COL2_B4, COL1, COL2
```

Output file properties

The following properties control how files are written, where to, and what their extensions will be. This is independent of the writer mode and data contents.

files.onepertable

Controls whether data is split over multiple rolling files (one per table in the input data) or all data is written to one rolling file. The default is true.

The syntax is:

```
{writer}.files.onepertable=true|false
```

In the following example the `dvswriter` will create one file per table and the `binarywriter` will write all data to one file.

```
dsvwriter.files.onepertable=true
binarywriter.files.onepertable=false
```

files.oneperopcode

Controls whether or not data is split based on the insert, update, delete operation codes.

For example, the following setting will create separate output files for inserts, updates and deletes:

```
dvswriter.files.oneperopcode=true
```

The default is false; output all records to the same files independent of the type of operation.

In addition to this property, you must also modify the `files.formatstring` property to accept the `%O` placeholder. This indicates the position to write the operation code when the file name is created if the `files.oneperopcode` property is set. The default filename should also include the operation code if that property is set.

files.prefix

Specifies a value to be used as the prefix for data files and control files. This property only applies if the writer is not in one per table mode (`files.onepertable=true`). For data files, the prefix is ignored if the `property files.formatstring` is being used.

By default, the prefix is set to the string `output`. A file named `data1` will become `outputdata1`by default. The file name will be `test_data1` using the following example.

```
dsvwriter.files.prefix=test_
```

files.data.rootdir, files.data.ext, files.data.tmpext

Specifies the location and extension of all data files. Before rolling over the files will have the `tmpext` extension, after rolling over they will have the `ext` extension. The extension does not have to be just an .`ext` format, additional characters can be appended to the file

name before the extension to differentiate the data output. You should ensure the named output directory exists, and that the user running the Oracle GoldenGate processes has the correct permissions to write to that directory. For example:

```
# specify the root directory for outputting data files
dsvwriter.files.data.rootdir=./out

# determine the extension for data files when rolled over
dsvwriter.files.data.ext=_data.dsv

# determine the extension for data files before rolling over
dsvwriter.files.data.tmpext=_data.dsv.temp
```

files.control.use, files.control.rootdir, files.control.ext

`files.control.use` is a boolean true or false value that defaults to true. The others are ASCII values. These properties determine the user, location and extension of control files. Control files will share the same name prefix as the data files they are related to, but will have the defined extension. By default `files.control.ext` is `.control`. For example:

```
# specify whether or not to output a control file
dsvwriter.files.control.use=true

# specify the extension to use for control files
dsvwriter.files.control.ext=_data.control

# directory in which to place control files, defaults to data directory
dsvwriter.files.control.rootdir=./out
```

files.formatstring

Specifies the filename format string to be used in creating the filenames for data files. The format string overrides the `files.prefix` property. This filename format string is similar in syntax to standard C formatting except the following placeholders can be added to the filename:

%s = schema

%t = table

%n = seqno

%d = timestamp

%o = opcode

The format of the `seqno` can be specified. For example %05n means 5 digits will be displayed and padded with 0's. The `seqno` starts at zero and is incremented by one each time a file rolls over. It is stored as a `long int` and therefore the maximum value is platform dependent. For example on a 64 bit machine the largest value is 2^64-1.

These placeholders can be intermingled with user specified text in any order desired. For example:

```
dsvwriter.files.formatstring=myext_%d_%010n_%s_%
```

files.data.bom.code

Specifies a hexadecimal value as the byte order marker (BOM) to be written to the beginning of the file. The user is responsible for ensuring the BOM matches the data in the files. If no hexadecimal value is specified the marker is not written.

The following example results in the UTF8 BOM `efbbf` written as the first bytes of all output files.

```
dsvwriter.files.data.bom.code=efbbbf
```

File rollover properties

The following properties determine the policies for rolling over files.

files.data.rollover.time

Specifies the maximum number of seconds of elapsed time that must pass from the first record written to the file before the file is rolled over. For example:

```
# number of seconds before rolling over
dsvwriter.files.data.rollover.time=10
```

files.data.rollover.size

Specifies the maximum number of bytes that must be written to the file before the file is rolled over.

This example sets the maximum to 10,000 bytes:

```
# max file size in KB before rolling over
dsvwriter.files.data.rollover.size=10000
```

files.data.norecords.timeout

Specifies the maximum number of elapsed seconds since data was written to a file to wait before rolling over the file. The default is 120 seconds.

This example sets the timeout interval to 10 seconds:

```
# roll over in case no records for a period of time
dsvwriter.files.data.norecords.timeout=10
```

files.rolloveronshutdown

Controls the policy for roll over when the Extract process stops. If this value is false, all empty temporary files will be deleted, but any that have data will be left as temporary files. If this property is true, all non-empty temporary files will be rolled over to their rolled file name, a checkpoint written and empty temporary files deleted. For example:

```
# roll over non-empty and delete all empty files when Extract stops
binarywriter.files.rolloveronshutdown=true
```

NOTE You can use time and/or size. If you use both, the first reached will cause a roll over. The timeout interval ensures files are rolled over if they contain data, even if there

are no records to be processed. If neither time or size are specified, files will roll over after a default maximum size of 1MB.

files.data.rollover.timetype

Controls whether to use the Julian commit timestamp rather than the system time to trigger file roll over. The syntax is:

```
{writer}.files.data.rollover.timetype=commit|system
```

The following example will use the commit timestamp of the source trail records to determine roll over:

```
dsvwriter.files.data.rollover.timetype=commit
```

The default is to use the system time to determine when to roll over files.

files.data.rollover.multiple

Controls whether or not all files will be rolled over simultaneously independent of when they first received records. Normally files are rolled over individually based on the time or size properties. The time is based on the roll over period, so it depends on the time records were first written to a particular file. In some cases, especially when outputting data with one file per table, you may want to roll over all currently open files at the same time, independent of when data was first written to that file.

The following example instructs the adapter to roll over all files simultaneously.

```
dsvwriter.files.data.rollover.multiple=true
```

The default value is `false`.

files.data.rollover.attime

Specifies a time for the adapter to roll over files. Enter the specified times in 24 hour format (HH:MM). The wildcard (*) is supported for hours and multiple entries are supported. The syntax is:

```
{writer}.files.data.rollover.attime={time specifier} [, ...]
```

The following example will roll over to a new file every hour on the hour:

```
dsvwriter.files.data.rollover.attime=*:00
```

The following example will roll over every half hour at fifteen minutes before and after the hour:

```
dsvwriter.files.data.rollover.attime=*:15, *45
```

Note that the `rollover.timetype` property determines whether the time to use is system or commit time.

csvwriter.writebuffer.size

Specifies the write buffer chunk size. Use to reduce the number of system write calls. For example:

```
csvwriter.writebuffer.size=36863
```

Data content properties

The following properties determine which data is written to the data files. These properties are independent of the format of the output data.

rawchars

Controls whether character data retains its original binary form or is output as ASCII. The default is false. This property should be set if the input data contains Unicode multibyte data that should not be converted to ASCII. For example:

```
# whether to output characters as ASCII or binary (for Unicode data)
dsvwriter.rawchars=false
binarywriter.rawchars=true
```

includebefores

Controls whether or not both the before and after image of data is included in the output for update operations. The default is false. This is only relevant if the before images are available in the original data, and `getupdatebefores` is present in all Oracle GoldenGate parameter files in the processing chain. For example:

```
# whether to output update before images
dsvwriter.includebefores=true
```

This produces …`"VAL_BEFORE_1","VAL_1","VAL_BEFORE_2","VAL_2"`…

afterfirst

Controls whether or not the after images is written before the before image when `includebefores` is set to `true`.

For example:

```
dsvwriter.afterfirst=true
```

This true setting results in the after image listed before the before image.

```
“VAL_1”, “VAL_BEFORE_1”, “VAL_2”, “VAL_BEFORE_2”
```

The default is `false. In this case` the after image is written after the before image.

includecolnames

Controls whether or not column names are output before the column values. The default is false. For example:

```
# whether to output column names
dsvwriter.includecolnames=true
```

This produces …`"COL_1","VAL_1","COL_2","VAL_2"`…

omitvalues

Controls whether or not column values are omitted in the output files. The default is false.

For example:

```
# whether to output column values
dsvwriter.omitvalues=false
```

This produces …"COL_1","COL_2"…, if `includecolnames` is also set to true

diffsonly

Controls whether all columns are output, or only those where the before image is different from the after image. The default is false. This only applies to updates and requires GETUPDATEBEFORES in all Oracle GoldenGate parameter files in the processing chain. This property is independent of the `includebefores` property. For example:

```
# whether to output only columns with differences between before and
# after images (deletes and inserts have all available columns)
dsvwriter.diffsonly=true
```

This produces …"VAL_1",,,"VAL_4",,,"VAL_7"…

omitplaceholders

Controls whether delimiters/lengths are included in the output for missing columns. The default is false. This applies to updates and deletes where the COMPRESSUPDATES or COMPRESSDELETES flag was present in a Oracle GoldenGate parameter file in the processing chain. In this case, values may be missing. Also, if `diffsonly` is true, values that are not different are said to be missing. For example:

```
# whether to skip record delimiters if columns are missing
dsvwriter.omitplaceholders=true
```

This changes …"VAL_1",,,"VAL_4",,,"VAL_7"… to …"VAL_1","VAL_4","VAL_7"…

Metadata columns

Metadata columns are optional Extract columns that contain data about a record, not actual record data. These columns are written at the beginning of the output record, before any column values.

Valid metadata columns

Valid metadata columns are:

position - A unique position indicator of records in a trail

opcode - I, U or D for Insert, Update and Delete records

txind - The general record position in a transaction (0 - begin, 1 - middle, 2 - end, 3 - only)

txoppos - Position of record in a transaction, starting from 0

schema - The schema (owner) name of the changed record

table - The table name of the changed record

schemaandtable - Both the schema and table name concatenated as schema.table

timestamp - The commit timestamp of the record

@<token name> - A token value defined in the Extract parameter file

$getenv - A `GETENV` value as documented in the *Oracle GoldenGate Reference Guide*; for example `$GGHEADER.OPCODE`

%COLNAME - The value of a data column

numops -The number of operations in the current transaction. This value will always be 1 if `goldengate.userexit.buffertxs` is not `true`

numcols - The number of columns to be output. This value is equal to the number of columns in the original record, minus the number of columns output as metadata columns up until the point this metadata column is used

"<value>" - Any literal value.

Using metadata columns

Some things to consider when using metadata columns:

- The ASCII values for `opcode` and `txind` can be overridden.
- For LDV, metadata columns can be variable or fixed length.
- The `position` can be written in hexadecimal or decimal.
- Any metadata column can be the internal value or it can be read from a column of the original data.
- A literal value is indicated by enclosing it in quotes. When a literal value is specified, that value will be output as a character string in the specified metadata column position using the appropriate quote policy.
- A column value is indicated by %COLNAME. When a column value is specified, that column value is output in the metadata section of the output record, rather than in the column values section. This may be used to ensure that the column is always output in the same position in the record, independent of the table being output.

The following properties apply to metadata columns.

metacols

Specifies the metadata columns to output in the order of output. Enter multiple names as ASCII values separated by commas. For example:

```
# which metacols to output and in which order
dsvwriter.metacols=timestamp,opcode,txind,position,schema,table
```

metacols.{metacol-name}.fixedlen

Specifies an integer value to determine the length of data to write for a metadata column. If the actual data is longer than the fixed length it will be truncated, if it is shorter the output will be padded. For example:

```
# timestamp is fixed length
dsvwriter.metacols.timestamp.fixedlen=23
```

This truncates `2007-08-03 10:30:51.123456` to `2007-08-03 10:30:51.123`

metacols.{metacol-name}.column

Specifies an ASCII value to use as the column name of data values instead of using the internal value for a metadata column. If set, this column name must exist in all tables processed by the user exit. There is currently no way to override this column name on a per table basis. For example, to override the internal timestamp from a column:

```
# timestamp is read from a column
dsvwriter.metacols.timestamp.column=MY_TIMESTAMP_COL
```

metacols.{token-name}.novalue.chars/code

Specifies values to represent characters or hexadecimal code to be used when the desired token value is not available. Use ASCII values for `chars and` hexadecimal values for `code`. The default value is {NO VALUE}. For example:

```
dsvwriter.metacols.TKN-SCN.novalue.chars=0
```

metacols.{metacol-name}.fixedjustify

Controls whether the justification for the metadata column value is to the left or right. By default all metadata columns will be justified to the left. For example, to justify a token to the right:

```
dsvwriter.metacols.TKN-SCN.fixedjustify=right
```

metacols.{metacol-name}.fixedpadchar.chars/code

Specifies either a character or code value to be used for padding a metadata column. Use ASCII values for `chars and` hexadecimal values for `code`. The default character used for padding is a space (" "). For example:

```
dsvwriter.metacols.TKN-SCN.fixedpadchar.chars=0
```

metacols.opcode.insert.chars/code, metacols.opcode.update.chars/code, metacols.opcode.delete.chars/code

Specifies override values for the default characters that identify insert, update and delete operations. Use ASCII values for `chars` and hexadecimal values for `code`. The default values are I for Insert, U for Update, D for delete.

The following example identifies instructs the adapter to use INS for inserts, UPD for updates and DEL for deletes.

```
# op code values are overridden
dsvwriter.metacols.opcode.insert.chars=INS
dsvwriter.metacols.opcode.update.chars=UPD
dsvwriter.metacols.opcode.delete.chars=DEL
```

metacols.txind.begin.chars/code, metacols.txind.middle.chars/code metacols.txind.end.chars/code, metacols.txind.whole.chars/code

Specifies the override values to use to identify the beginning, middle, end of transactions,

or if an operation that is the whole transaction. Use ASCII values for `chars` and hexadecimal values for `code`. The default values are 0 for Begin, 1 for Middle, 2 for End and 3 for Whole.

The following example overrides the 0, 1, 2, 3 with the letters B, M, E, and W.

```
# tx indicator values are overridden
dsvwriter.metacols.txind.begin.chars=B
dsvwriter.metacols.txind.middle.chars=M
dsvwriter.metacols.txind.end.chars=E
dsvwriter.metacols.txind.whole.chars=W
```

metacols.position.format

Controls whether the output of the of the `position` metadata column is in decimal or hexadecimal format. If hexadecimal, this will typically be a 16 character value, if decimal the length will vary. Currently this contains the sequence number and RBA of the Oracle GoldenGate trail that the Extract process is reading from. For example:

```
# position is in decimal format (seqno0000000rba)
dsvwriter.metacols.position.format=dec
```

This produces `120000012345` for seqno 12, rba 12345

```
binarywriter.metacols.position.format=hex
```

This produces `0000000c00003039` for seqno 12, rba 12345.

metacols.{COLNAME}.omit

Controls whether the COLNAME column can be used as metadata but not output.

The following example specifies that `numcols` can be used as metadata, but not output.

```
dsvwriter.metacols.numcols.omit=true
```

begintx.metacols, endtx.metacols

Specifies the metadata columns to use to mark the beginning and end of a transaction. These marker records are written (with end of line delimiters) to the output files before and after the operation records that make up the transaction.

The syntax is:

```
{writer}.begintx.metacols={metacols list}
```

The following example specifies marking the beginning of a transaction with the letter B and the number of operations in the transaction.

```
dvswriter.begintx.metacols="B",numops
```

In the following example, the end of the transaction marker will be the letter E.

```
dvswriter.endtx.metacols="E"
```

Any of the existing metadata columns can be used in the transaction begin and end markers. If you specify a column value or specific property of a record (such as table name)

for `begintx.metacols`, the value for the first record in the transaction is used. For `endtx.metacols`, the value for the last record is used.

For example, if the transaction has the following records:

```
rec=0,table=tabA,operation=insert,col1=val1,col2=val2
rec=1,table=tabA,operation=update,col1=val3,col2=val4
rec=2,table=tabA,operation=delete,col1=val5,col2=val6
rec=3,table=tabB,operation=update,col1=val7,col2=val8
```

And the properties are set as follows:

```
dvswriter.begintx.metacols="B",table,%col2
dvswriter.endtx.metacols="E",table,%col2
```

Then the begin transaction marker will be `"B","tabA","val2"` and the end marker will be `"E","tabB","val8"`.

If `numops` is used to ouptut the number of operations in a transaction for either the begin or end markers, the user must also set:

```
goldengate.userexit.buffertxs=true
```

NOTE When this property is set, the adapter buffers transactions in memory, so care should be taken to limit the number of operations in the transactions being handled by the system.

DSV specific properties

DSV files have the following record format:

$\{[METACOL][FD]\}^{n}\{[COL][FD]\}^{m}[LD]$

Where:

METACOL is any defined metadata column

COL is any data column

FD is the field delimiter

LD is the line delimiter

Column values may be quoted, e.g. `"2007-01-10 10:20:31","U","MY.TABLE", 2000,"DAVE"`

dsv.nullindicator.chars/code

Specifies the characters to use for NULL values in delimiter separated files. These values override the default NULL value of an empty string. Use ASCII values for `chars` and hexadecimal values for `code`. For example:

```
dsvwriter.dsv.nullindicator.chars=NULL
dsvwriter.dsv.nullindicator.code=0a0a0a0a
```

dsv.fielddelim.chars/code

Specifies an override value for the field delimiter. The default is a comma (,). Use ASCII

values for `chars and` hexadecimal values for `code`. For example:

```
# define the characters to use for field delimiters in DSV files
dsvwriter.dsv.fielddelim.chars=Ç
```

dsv.linedelim.chars/code

Specifies an override value for the line delimiter. The default is a newline appropriate to the operating system. Use ASCII values for `chars` and hexadecimal values for `code`. For example:

```
# define the characters to use for line delimiters in DSV files
dsvwriter.dsv.linedelim.chars=\n
```

dsv.quote.chars/code

Specifies an override value for the quote character. The default is a double quote (“). Use ASCII values for `chars` and hexadecimal values for `code`.For example:

```
# define the characters to use for quotes in DSV files
dsvwriter.dsv.quotes.chars=’
```

dsv.quotes.policy

Controls the policy for applying quotes.

The syntax is:

```
{writer}.dsv.quotes.policy={default|none|always|datatypes}
```

Where: **default** – Only dates and chars are quoted

never – No metadata column or column values are quoted

always – All metadata columns and column values are quoted

datatypes – Only specific data types are quoted

If this property is set it will override the `dsv.quotealways` property.Use the `dsv.quotes.policy.datatypes` property to specify which data types should be quoted.

dsv.quotes.policy.datatypes

Controls whether integer, character, float, or datetime data types are to be quoted when `dsv.quotes.policy` is set to `datatype`. The syntax is:

```
{writer}.dsv.quotes.policy.datatypes=[char][,integer][,float][,date]
```

For example the following instructs the adapter to quote characters and datetime values only.

```
dsvwrite.dsv.quotes.policy.datatypes=char,date
```

If no data types are specified the data types option defaults to all data types, which is equivalent to always.

dsv.nullindicator/fielddelim/linedelim/quotes.escaped.chars/code

Specifies the escaped value for any of the above values. If set, all values will be checked for the none escaped value and replaced with the escaped value when output. Use ASCII values for `chars` and hexadecimal values for `code`. For example:

```
# (optionally) you can define the characters (or code) to use
# to escape these values if found in data values
dsvwriter.dsv.quotes.escaped.chars=""
```

This changes: “`some text`” to ““`some text`“”.

dsv.onecolperline

Controls whether or not each column value is forced onto a new line. Each line will also contain the metadata columns defined for this writer. The default is false. For example:

```
# Force each column onto a new line with its own meta cols
dsvwriter.dsv.onecolperline=true
```

This changes:`{metacols},val_1,val_2` to

```
{metacols},val1
{metacols},val2
```

dsv.quotealways

Controls whether or not each column is surrounded by quotes, even if it is a numeric value. The default is false.

NOTE This property has been superseded by `dsv.quotes.policy` and is supported only for backward compatibility. The value set for `dsv.quotealways` is ignored if `dsv.quotes policy` is set.

For example:

```
dsvwriter.dsv.quotealways=true
```

Changes: `…,1234,”Hello”,10` to

```
…,”1234”,”Hello”,”10”
```

LDV specific properties

LDV files have the following record format:

$[RECLEN][METACOLS]\{[FLAG][LEN][VALUE]\}^{n}$

Where:

RECLEN is the full record length in bytes

METACOLS are all selected metadata columns

FLAG can be (M)issing, (P)resent, or (N)ull

LEN is the column values length (0 for missing and null)

VALUE is the column value

For example:

```
0107 2007-01-10 10:20:31U302MY05TABLEP042000M00N00P04DAVE
```

ldv.vals.missing.chars/code, ldv.vals.present.chars/code, ldv.vals.null.chars/code

Specifies override values for missing, present and null indicators. Use ASCII values for `chars` and hexadecimal values for `code`. For example:

```
binarywriter.ldv.vals.missing.chars=MI
binarywriter.ldv.vals.present.chars=PR
binarywriter.ldv.vals.null.chars=NL
```

ldv.lengths.record.mode, ldv.lengths.field.mode

Controls the output mode of record and field lengths. The value can be either `binary` or `ASCII`. The default is binary.

If `binary`, the number written to the file will be encoded in binary bytes. If `ASCII`, characters representing the decimal value of the length will be used. For example:

```
binarywriter.ldv.lengths.record.mode=binary
binarywriter.ldv.lengths.field.mode=binary
```

ldv.lengths.record.length, ldv.lengths.field.length

Specifies the record and field lengths as integer values. If the mode is `ASCII`, this represents the fixed number of decimal digits to use. If `binary` it represents the number of bytes.

In ASCII mode the lengths can be any value, but the exit will stop if a length exceeds the maximum. In binary mode, the lengths can be 2,4, or 8 bytes, but record length must be greater than field length. For example:

```
# Lengths can be binary (2,4, or 8 bytes) or ASCII (any length)
binarywriter.ldv.lengths.record.length=4
binarywriter.ldv.lengths.field.length=2
```

Statistics and reporting

There are two ways that statistics regarding the data written to data files can be obtained:

- As a report written to the Oracle GoldenGate report file
- As a separate summary file associated with a data file on rollover

These two mechanisms can be used together or separately.

The data that can be obtained includes, 1) the total records processed, broken down to inserts, updates, deletes; 2) records processed per table, also broken down; 3) total rate and rate per table; 4) delta for these since last report. Reporting can be time based, or synced to file rollover

This data can be written to the report file or as a summary file linked to a data file on

rollover. The reporting format is fixed. The summary file contains a delimited format with the above data, but related to the contents of a particular data file that can be used by a data integration product to cross-check processing. It will have the same name as the data file, but different extension.

statistics.toreportfile

Controls whether or not statistics are output to the Oracle GoldenGate report file. For example:

```
binarywriter.statistics.tosummaryfile=true
```

statistics.period

Specifies the time period for statistics. The value can be either `timebased` or `onrollover`.

For example:

```
dsvwriter.statistics.period=onrollover
dsvwriter.statistics.period=timebased
```

If `timebased`, the time period should be set in `statistics.time`.

NOTE These values are valid only for outputting statistics to the report file. Statistics will be outputted to the summary file only on rollover.

statistics.time

Specifies a time interval in seconds after which statistics will be reported.

For example:

```
binarywriter.statistics.time=5
```

statistics.tosummaryfile

Controls whether or not a summary file containing statistics for each data file will be created on rollover.

The following example creates the summary file.

```
binarywriter.statistics.tosummaryfile=true
```

statistics.summary.fileformat

Controls the content of the summary files and the order in which the content is written. Multiple comma separated ASCII values can be specified.

Valid values are:

schema – schema or owner of the table that the statistics relate to

table – table that the statistics relate to

schemaandtable – schema and table in one column separated by a period '.'

gtotal – total number of records output for the specified table since the user exit was started

gtotaldetail – total number of inserts, updates and deletes separated by the delimiter since the user exit was started

gctimestamp – minimum and maximum commit timestamp for the specified table since user exit was started

ctimestamp – minimum and maximum commit timestamps for the specified table in the related data file.

total – total number of records output for the specified table in the related data file

totaldetail – total number of inserts, updates and deletes output for the specified table in the related data file

rate – average rate of output of data for the specified table in the related data file in records per second

ratedetail – average rate of inserts, updates and deletes for the specified table in the related data file in records per second

For example:

```
binarywriter.statistics.summary.fileformat=
             schema,table,total,totaldetail,gctimestamp,ctimestamp
```

statistics.overall

Controls whether or not an additional statistics row is written to the summary files. This row contains the overall (across all tables) statistics defined by the user using the `statistics.summary.fileformat` property.

The following example will write this row.

```
binarywriter.statistics.overall=true
```

statistics.summary.delimiter.chars/code, statistics.summary.eol.chars/code

Specifies override values for the field delimiter and end of line delimiter for the summary files. Use ASCII values for `chars` and hexadecimal values for `code`. The default is a comma ',' delimiter and newline character. For example:

```
binarywriter.statistics.summary.delimiter.chars=|
binarywriter.statistics.summary.eol.code=0a0c
```

statistics.summary.extension

Specifies the override extension to use for the statistics summary file output per data file. The default is `stats`.

The following example changes the extension from `.stats` to `.statistics`.

```
binarywriter.statistics.summary.extension=.statistics
```

CHAPTER 4

Property Templates

There are a large number of properties that can be set for every file writer being configured. To make this task easier, Oracle GoldenGate supplies property templates that include standard properties for particular file consumers.

Using property templates

Using the template presets certain properties based on common usage. The template settings can be overridden by setting the property in the properties file. For each property in the template, the system first checks to see if that property is set in the properties file itself. If the user hasn't specified it, the template setting is used.

{writer}.template={template_name}

Use the `template` property to specify the name of the template of standard properties that is to be used. The syntax is:

```
{writer}.template={template_name}
```

Where: The template name corresponds to a named file of default property settings for a particular flat file consumer. Valid template names include:

- SIEBEL - Template to create one DSV format output file with transaction information for consumption by Siebel Remote.
- ABINITIO - Template to create LDV format output for consumption by Ab Initio.
- GREENPLUM - Template to create one DSV format output file per table for consumption by Greenplum.
- NETEZZA - Template to create one DSV format output file per table for consumption by Netezza.
- COMMADELIM - Template to create one comma delimited output file per table.

Default properties

All writers use the following properties. The values shown for each property are the defaults.

```
{writer}.files.data.rootdir=./out
{writer}.files.data.rollover.time=10
{writer}.files.data.rollover.size=100000
{writer}.files.data.norecords.timeout=10
{writer}.files.control.use=true
{writer}.files.control.ext=.ctrl
{writer}.files.control.rootdir=./out
```

Siebel Remote

```
goldengate.userexit.outputmode=txs
goldengate.userexit.buffertxs=true
goldengate.userexit.datetime.removecolon=true
goldengate.userexit.timestamp=utc
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=true
{writer}.includecolnames=true
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=true
{writer}.files.onepertable=false
{writer}.files.data.ext=_data.csv
{writer}.files.data.tmpext=_data.csv.temp
{writer}.files.data.bom.code=efbbbf
{writer}.dsv.nullindicator.chars=NULL
{writer}.dsv.nullindicator.escaped.chars=
{writer}.dsv.fielddelim.chars=,
{writer}.dsv.fielddelim.escaped.chars=
{writer}.dsv.linedelim.chars=\n
{writer}.dsv.linedelim.escaped.chars=
{writer}.dsv.quotes.chars="
{writer}.dsv.quotes.escaped.chars=""
{writer}.dsv.quotealways=true
{writer}.groupcols=true
{writer}.afterfirst=true
{writer}.begintx.metacols="B","S",position,"GGMC",%LAST_UPD_BY,"1",
   numops
{writer}.metacols="R",opcode,%ROW_ID,%LAST_UPD_BY,%LAST_UPD,
%MODIFICATION_NUM,%CONFLICT_ID,position,txoppos,table,"","","","","",
   "",%DB_LAST_UPD,%DB_LAST_UPD_SRC,numcols
{writer}.metacols.DB_LAST_UPD.omit=true
{writer}.metacols.DB_LAST_UPD_SRC.omit=true
{writer}.metacols.opcode.updatepk.chars=U
{writer}.metacols.position.format=dec
{writer}.endtx.metacols="E"
```

Ab Initio

```
{writer}.mode=LDV
{writer}.files.onepertable=false
{writer}.files.data.ext=.data
{writer}.files.data.tmpext=.temp
{writer}.metacols=position,timestamp,opcode,txind,schema,table
{writer}.metacols.timestamp.fixedlen=26
{writer}.metacols.schema.fixedjustify=right
{writer}.metacols.schema.fixedpadchar.chars=Y
{writer}.metacols.opcode.fixedlen=1
{writer}.metacols.opcode.insert.chars=I
{writer}.metacols.opcode.update.chars=U
{writer}.metacols.opcode.delete.chars=D
{writer}.metacols.txind.fixedlen=1
{writer}.metacols.txind.begin.chars=B
{writer}.metacols.txind.middle.chars=M
{writer}.metacols.txind.end.chars=E
{writer}.metacols.txind.whole.chars=W
{writer}.metacols.position.format=dec
{writer}.ldv.vals.missing.chars=M
{writer}.ldv.vals.present.chars=P
{writer}.ldv.vals.null.chars=N
{writer}.ldv.lengths.record.mode=binary
{writer}.ldv.lengths.record.length=4
{writer}.ldv.lengths.field.mode=binary
{writer}.ldv.lengths.field.length=2
{writer}.statistics.period=onrollover
{writer}.statistics.tosummaryfile=true
{writer}.statistics.overall=true
{writer}.statistics.summary.fileformat=schema,table,schemaandtable,
   total,gctimestamp,ctimestamp
{writer}.statistics.summary.delimiter.chars=|
{writer}.statistics.summary.eol.chars=\n
```

Netezza

```
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=false
{writer}.includecolnames=false
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=false
{writer}.files.onepertable=true
{writer}.files.data.ext=_data.dsv
{writer}.files.data.tmpext=_data.dsv.temp
{writer}.dsv.nullindicator.chars=
{writer}.dsv.fielddelim.chars=;
{writer}.dsv.fielddelim.escaped.chars=
```

Greenplum

```
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=false
{writer}.includecolnames=false
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=false
{writer}.files.onepertable=true
{writer}.files.data.ext=_data.dsv
{writer}.files.data.tmpext=_data.dsv.temp
{writer}.dsv.nullindicator.chars=
{writer}.dsv.fielddelim.chars=|
{writer}.dsv.fielddelim.escaped.chars=
{writer}.metacols=opcode,timestamp
{writer}.metacols.opcode.insert.chars=I
{writer}.metacols.opcode.update.chars=U
{writer}.metacols.opcode.delete.chars=D
```

Comma Delimited

```
{writer}.mode=DSV
{writer}.rawchars=false
{writer}.includebefores=false
{writer}.includecolnames=false
{writer}.omitvalues=false
{writer}.diffsonly=false
{writer}.omitplaceholders=false
{writer}.files.onepertable=true
{writer}.files.data.ext=_data.dsv
{writer}.files.data.tmpext=_data.dsv.temp
{writer}.dsv.nullindicator.chars=NULL
{writer}.dsv.fielddelim.chars=,
{writer}.dsv.linedelim.chars=\n
{writer}.dsv.quotes.chars="
{writer}.dsv.quotes.escaped.chars=""
{writer}.metacols=position,txind,opcode,timestamp,schema,table
{writer}.statistics.period=onrollover
{writer}.statistics.overall=true
```

Index

C

control files 10
CUSEREXIT parameter 8
  INCLUDEUPDATEBEFORES 8
  PARAMS 8
  PASSTHRU 8
CUSEREXIT PARAMS
  specifying the property file 8

D

data content properties 21
data files 9
Delimiter Separated Values 10, 16
DSV 10, 16
DSV specific properties 26

E

error handling 11
ETL tools 5
Extract parameters 8
  CUSEREXIT 8
  EXTRACT 8
  SOURCEDEFS 8
  TABLE 8

F

ffwriter.prm 8
file rollover properties 19
filewriter 5
flat file integration 5
Flat File User Exit
  operation 9
  properties 12

G

general properties 14

I

installing 7, 13

J

Java API 5
Java integration 5

L

LDV 10, 16
LDV specific properties 28
Length Delimited Values 10, 16
logging properties 13

M

metadata column properties 22
metadata columns 22

O

operation
  Flat File User Exit 9
Oracle GoldenGate
  core application 7
  documentation 6
Oracle GoldenGate processes 11
output file properties 17
output format properties 16

P

processes, Oracle GoldenGate 11
properties
data content 21
DVS 26
file rollover 19
general 14
LDV 28
logging 13
metadata columns 22
output file 17
output formats 16
properties, Flat File User Exit 12
property templates 32
supplied by Oracle GoldenGate 32

R

reporting 29

S

statistical summaries 11
statistics 29

T

troubleshooting 12

W

writer 9
writers, multiple 14