Oracle® Fusion Middleware

Integrating Oracle GoldenGate for Big Data

Release 12*c* (12.2.0.1)

E65148-03

June 2016

Oracle Fusion Middleware Integrating Oracle GoldenGate for Big Data, Release 12c (12.2.0.1)

E65148-03

Contents

Preface

This book contains information about configuring and running Oracle GoldenGate for Big Data.

Audience

This guide is intended for system administrators who are configuring and running Oracle GoldenGate for Big Data.

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 Documents

The Oracle GoldenGate for Big Data documentation set includes the following components:

- *Release Notes for Oracle GoldenGate for Big Data*
- *Integrating Oracle GoldenGate for Big Data*
- *Installing Oracle GoldenGate Big Data*
- *Administering Oracle GoldenGate for Big Data*

The complete Oracle GoldenGate documentation set includes the following components:

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview

This chapter describes Oracle GoldenGate for Big Data, how to set up its environment, use it with Replicat and Extract, logging data, and other configuration details. It contains the following sections:

1.1 Overview

The Oracle GoldenGate for Big Data integrations run as pluggable functionality into the Oracle GoldenGate Java Delivery framework, also referred to as the Java Adapters framework. This functionality extends the Java Delivery functionality. Oracle recommends that you review the Java Delivery documentation in the Oracle GoldenGate Application Adapters Guide. Much of the Big Data functionality employs and extends the Java Delivery functionality.

1.2 Java Environment Setup

The Oracle GoldenGate for Big Data integrations create an instance of the Java virtual machine at runtime. Oracle GoldenGate for Big Data requires Java 7. It is recommended that you set the `JAVA_HOME` environment variable to point to Java 7 installation directory. Additionally, the Java Delivery process needs to load the `libjvm.so` (`libjvm.dll` on Windows) and `libjsig.so` (`libjsig.dll` on Windows) Java shared libraries. These libraries are installed as part of the JRE. The location of these shared libraries need to be resolved and the appropriate environmental variable set to resolve the dynamic libraries needs to be set so the libraries can be loaded at runtime (that is, `LD_LIBRARY_PATH`, `PATH`, or `LIBPATH`).

1.3 Properties Files

There are two Oracle GoldenGate properties files required to run the Oracle GoldenGate Java Deliver user exit (alternatively called the Oracle GoldenGate Java Adapter). It is the Oracle GoldenGate Java Delivery that hosts Java integrations including the Big Data integrations. The Oracle GoldenGate Java Delivery can run with either the Oracle GoldenGate Replicat or Extract process, although running with the Replicat process is considered the better practice. A Replicat or Extract properties file is required in order to run the Replicat or Extract process. The required naming convention for the Replicat or Extract file name is the `*process_name*.prm`. You exit syntax in the Replicat or Extract properties file provides the name and location of the Java Adapter Properties file. It is the Java Adapter Properties file where the configuration properties for the Java adapter include Big Data integrations. Both properties files are required to run Oracle GoldenGate for Big Data integrations. Alternatively the Java Adapters Properties can be resolved using the default syntax, `*process_name*.properties`. It you use the default naming for the Java Adapter Properties file then the name of the Java Adapter Properties file can be omitted from the Replicat or Extract properties file.

Samples of the properties files for Oracle GoldenGate for Big Data integrations can be found in the subdirectories of the following directory:

`*GoldenGate_install_dir*/AdapterExamples/big-data`

1.4 Transaction Grouping

The principal way to improve performance in Oracle GoldenGate for Big Data integrations is by the use of transaction grouping. In transaction grouping, the operations of multiple transactions are grouped together in a single larger transaction. The application of a larger grouped transaction is typically much more efficient than the application of individual smaller transactions. Transaction grouping is possible with both the Replicat and Extract processes and will be discussed in the following sections detailing running with Replicat or Extract.

1.5 Running with Replicat

This section explains how to run the Java Adapter with the Oracle GoldenGate Replicat process.

1.5.1 Replicat Configuration

The following is an example of a Replicat process properties file for Java Adapter.

```
REPLICAT hdfs
TARGETDB LIBFILE libggjava.so SET property=dirprm/hdfs.properties
--SOURCEDEFS ./dirdef/dbo.def
DDL INCLUDE ALL
GROUPTRANSOPS 1000
MAPEXCLUDE dbo.excludetable
MAP dbo.*, TARGET dbo.*;
```

The following is explanation of the Replicat configuration entries:

`REPLICAT hdfs` - The name of the Replicat process.

`TARGETDB LIBFILE libggjava.so SET property=dirprm/hdfs.properties` - Names the target database as you exit `libggjava.so` and sets the Java Adapters Property file to `dirprm/hdfs.properties`

`--SOURCEDEFS ./dirdef/dbo.def` - Sets a source database definitions file. Commented out because Oracle GoldenGate 12.2.0.1 trail files provide metadata in trail.

`GROUPTRANSOPS 1000` - To group 1000 transactions from the source trail files into a single target transaction. This is the default and improves the performance of Big Data integrations.

`MAPEXCLUDE dbo.excludetable` - To identify tables to exclude.

`MAP dbo.*, TARGET dbo.*;` - Shows the mapping of input to output tables.

1.5.2 Adding the Replicat Process

The command to add and start the Replicat process in `ggsci` is the following:

```
ADD REPLICAT hdfs, EXTTRAIL ./dirdat/gg
START hdfs
```

1.5.3 Replicat Grouping

The Replicat process provides the Replicat configuration property `GROUPTRANSOPS` to control transaction grouping. By default, the Replicat process implements transaction grouping of 1000 source transactions into a single target transaction. If you want to turn off transaction grouping then the `GROUPTRANSOPS` Replicat property should be set to `1`.

1.5.4 Replicat Checkpointing

`CHECKPOINTTABLE` and `NODBCHECKPOINT` are not applicable for Java Delivery with Replicat. Beside Replicat checkpoint file (`.cpr`), additional checkpoint file (`dirchk/<group>.cpj`) will be created that contains information similar to `CHECKPOINTTABLE` in Replicat for RDBMS.

1.5.5 Unsupported Replicat Features

The following Replicat features are not supported in this release:

- `BATCHSQL`
- `SQLEXEC`
- Stored procedure
- Conflict resolution and detection (CDR)
- `REPERROR`

1.5.6 Mapping Functionality

The Oracle GoldenGate Replicat process supports mapping functionality to custom target schemas. This functionality is not available using the Oracle GoldenGate Extract process. You must use the Metadata Provider functionality to define a target schema or schemas and then use the standard Replicat mapping syntax in the Replicat configuration file to define the mapping. Refer to the Oracle GoldenGate Replicat documentation to understand the Replicat mapping syntax in the Replication configuration file. For instructions on setting up the Metadata Provider, see Using the Metadata Provider.

1.6 Running with Extract

This section explains how to run Java Adapter with the Oracle GoldenGate Extract process.

1.6.1 Extract Configuration

The following

```
EXTRACT hdfs
discardfile ./dirrpt/avro1.dsc, purge
--SOURCEDEFS ./dirdef/dbo.def
CUSEREXIT libjavaue.so CUSEREXIT PASSTHRU, INCLUDEUPDATEBEFORES, PARAMS "dirprm/
hdfs.props"
GETUPDATEBEFORES
TABLE dbo.*;
```

The following is explanation of the Replicat configuration entries:

`EXTRACT hdfs` - The Extract process name.

`discardfile ./dirrpt/avro1.dsc, purge` - Set the discard file

`--SOURCEDEFS ./dirdef/dbo.def` - Source definitions are not required for 12.2 trial files.

`CUSEREXIT libjavaue.so CUSEREXIT PASSTHRU, INCLUDEUPDATEBEFORES, PARAMS "dirprm/hdfs.props"` - Set you exit shared library, and point to the Java Adapter Properties file

`GETUPDATEBEFORES` - Get update before images.

`TABLE dbo.*;` - Select which tables to replicate or exclude to filter.

1.6.2 Adding the Extract Process

```
ADD EXTRACT hdfs, EXTTRAILSOURCE ./dirdat/gg
START hdfs
```

1.6.3 Extract Grouping

The Extract process provides no functionality for transaction grouping. However, transaction grouping is still possible when integrating Java Delivery with the Extract process. The Java Delivery layer enables transaction grouping with configuration in the Java Adapter properties file.

1. `gg.handler.name.mode`

   To enable grouping, the value of this property must be set to tx.

2. `gg.handler.name.maxGroupSize`

   Controls the maximum number of operations that can be held by an operation group - irrespective of whether the operation group holds operations from a single transaction or multiple transactions.

   The operation group will send a transaction commit and end the group as soon as this number of operations is reached. This property leads to splitting of transactions across multiple operation groups.

3. `gg.handler.name.minGroupSize`

   This is the minimum number of operations that must exist in a group before the group can end.

   This property helps to avoid groups that are too small by grouping multiple small transactions into one operation group so that it can be more efficiently processed.

Note:

`maxGroupSize` should always be greater than or equal to `minGroupSize`; that is, `maxGroupSize >= minGroupSize`.

Note:

It is *not* recommended to use the Java layer transaction grouping when running Java Delivery with the Replicat process. If running with the Replicat process, you should use Replicat transaction grouping controlled by the `GROUPTRANSOPS` Replicat property.

1.7 Logging

Logging is essential to troubleshooting Oracle GoldenGate for Big Data integrations with Big Data targets. This section covers how Oracle GoldenGate for Big Data integration log and the best practices for logging.

1.7.1 Extract or Replicat Process Logging

Oracle GoldenGate for Big Data integrations leverage the Java Delivery functionality described in the *Oracle GoldenGate Application Adapters Guide*. In this setup, either a Oracle GoldenGate Replicat or Extract process loads a user exit shared library. This shared library then loads a Java virtual machine to thereby interface with targets providing a Java interface. So the flow of data is as follows:

Extract Process > User Exit > Java Layer

or

Replicat Process >User Exit > Java Layer

It is important that all layers log correctly so that users can review the logs to troubleshoot new installations and integrations. Additionally, if a customer has a problem that requires contacting Oracle Support, the log files are a key piece of information to be provided to Oracle Support so that the problem can be efficiently resolved.

A running Replicat or Extract process creates or appends log files into the `<GG Home>/dirrpt` directory that adheres to the following naming convention: <Replicat or Extract process name>.rpt. If a problem is encountered when deploying a new Oracle GoldenGate process, this is likely the first log file to examine for problems. The Java layer provides much of the heavy lifting for integrations with Big Data applications. Therefore are many things that can go wrong in the Java layer when performing the initial setup of a Oracle GoldenGate Big Data integration. You therefore need to understand how to control logging in the Java layer.

1.7.2 Java Layer Logging

The Oracle GoldenGate for Big Data product provides flexibility for logging from the Java layer. The recommended best practice is to use Log4j logging to log from the Java layer. Enabling simple Log4j logging requires the setting of two configuration values in the Java Adapters configuration file.

```
gg.log=log4j
gg.log.level=INFO
```

These `gg.log` settings will result in a Log4j file to be created in the *`GoldenGate_Home/`*`dirrpt` directory that adheres to this naming convention, *`Replicat or Extract process name_log level`*`_log4j.log`. The supported Log4j log levels are in the following list in order of increasing logging granularity.

- OFF
- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

Selection of a logging level will include all of the coarser logging levels as well (that is, selection of `WARN` means that log messages of `FATAL`, `ERROR` and `WARN` will be written to the log file). The Log4j logging can additionally be controlled by separate Log4j properties files. These separate Log4j properties files can be enabled by editing the `bootoptions` property in the Java Adapter Properties file. Three example Log4j properties files are included with the installation and, are included in the classpath:

```
log4j-default.properties
log4j-debug.properites
log4j-trace.properties
```

Any one of these files can be modifying the `bootoptions` as follows:

```
javawriter.bootoptions=-Xmx512m -Xms64m -
Djava.class.path=.:ggjava/ggjava.jar -
Dlog4j.configuration=samplelog4j.properties
```

You can use their own customized Log4j properties file to control logging. The customized Log4j properties file must be available in the Java classpath so that it can be located and loaded by the JVM. The contents of a sample custom Log4j properties file is the following:

```
# Root logger option
log4j.rootLogger=INFO, file

# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=sample.log
log4j.appender.file.MaxFileSize=1GB
log4j.appender.file.MaxBackupIndex=10
```

```
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -
%m%n
```

There are two important requirements when you use a custom Log4j properties file. First, the path to the custom Log4j properties file must be included in the `javawriter.bootoptions` property. Logging initializes immediately when the JVM is initialized while the contents of the `gg.classpath` property is actually appended to the `classloader` after the logging is initialized. Second, the `classpath` to correctly load a properties file must be the directory containing the properties file without wildcards appended.

1.8 Metadata Change Events

A new feature of Oracle GoldenGate 12.2 is to propagate metadata change events from the source database to the trail file. This functionality is limited to Oracle Database replication sources for the 12.2 release. Refer to the Oracle GoldenGate for Oracle Database documentation for information on how to enable this functionality.

The Oracle GoldenGate for Big Data Handlers and Formatters provide functionality to take action when a metadata change event is encountered. The ability to take action in the case of metadata change events depends on the metadata change events being available in the source trail file. Oracle GoldenGate 12.2 supports metadata in trail and the propagation of DDL data from a source Oracle Database. If the source trail file does not have metadata in trail and DDL data (metadata change events) then it is not possible for Oracle GoldenGate for Big Data to provide and metadata change event handling.

1.9 Configuration Property `CDATA[]` Wrapping

The Big Data Handlers and Formatters support the configuration of many parameters in the Java properties file the value of which may be interpreted as white space. The configuration handling of the Java Adapter is such that it will trim white space from configuration values from the Java configuration file. This behavior of trimming whitespace may be desirable for some configuration values and undesirable for other configuration values. The default functionality of trimming the whitespace was left in place. New functionality was added whereby you can wrap white space values inside of special syntax in order to preserve the whites pace for selected configuration variables. Oracle for Big Data borrows the XML syntax of `CDATA[]` to preserve white space. Values that would be considered to be white space can be wrapped inside of `CDATA[]`.

The following is an example attempting to set a new-line delimiter for the Delimited Text Formatter:

```
gg.handler.{name}.format.lineDelimiter=\n
```

This configuration will not be successful. The new-line character is interpreted as white space and will be trimmed from the configuration value. Therefore the `gg.handler` setting effectively results in the line delimiter being set to an empty string.

In order to preserve the configuration of the new-line character simply wrap the character in the `CDATA[]` wrapper as follows:

```
gg.handler.{name}.format.lineDelimiter=CDATA[\n]
```

Configuring the parameter with the `CDATA[]` wrapping will preserve the white space and the line delimiter will now be a new-line character. Parameters that support `CDATA[]` wrapping are explicitly listed in this documentation.

1.10 Using Regular Expression Search and Replace

You can perform more powerful search and replace operations of both schema data (catalog names, schema names, table names, and column names) and column value data, which are separately configured. Regular expressions (regex) are characters that customize a search string through pattern matching. You can match a string against a pattern or extract parts of the match. Oracle GoldenGate for Big Data uses the standard Oracle Java regular expressions package, `java.util.regex`. For more information, see "Regular Expressions" in the Base Definitions volume at **The Single UNIX Specification, Version 4**.

1.10.1 Using Schema Data Replace

You can replace schema data using the `gg.schemareplaceregex` and `gg.schemareplacestring` parameters. Use `gg.schemareplaceregex` to set a regular expression, and then use it to search catalog names, schema names, table names, and column names for corresponding matches. Matches are then replaced with the content of the `gg.schemareplacestring` value. The default value of `gg.schemareplacestring` is an empty string or `""`.

For example, some system table names start with a dollar sign like `$mytable`. You may want to replicate these tables even though most Big Data targets do not allow dollar signs in table names. To remove the dollar sign, you could configure the following replace strings:

```
gg.schemareplaceregex=[$]
gg.schemareplacestring=
```

The resulting example of searched and replaced table name is `mytable`. These parameters also support `CDATA[]` wrapping to preserve whitespace in the value of configuration values. So the equivalent of the preceding example using `CDATA[]` wrapping use is:

```
gg.schemareplaceregex=CDATA[[$]]
gg.schemareplacestring=CDATA[]
```

The schema search and replace functionality only supports a single search regular expression and a single replacement string.

1.10.2 Using Content Data Replace

You can replace content data using the `gg.contentreplaceregex` and `gg.contentreplacestring` parameters to search the column values using the configured regular expression and replace matches with the replacement string. For example, this is useful to replace line feed characters in column values. If the delimited text formatter is used then line feeds occurring in the data will be incorrectly interpreted as line delimiters by analytic tools.

You can configure *n* number of content replacement regex search values. The regex search and replacements are done in the order of configuration. Configured values must follow a given order as follows:

```
gg.conentreplaceregex=some_regex
gg.conentreplacestring=some_value
gg.conentreplaceregex1=some_regex
gg.conentreplacestring1=some_value
gg.conentreplaceregex2=some_regex
gg.conentreplacestring2=some_value
```

Configuring a subscript of 3 without a subscript of 2 would cause the subscript 3 configuration to be ignored.

Attention:

Regular express searches and replacements require computer processing and can reduce the performance of the Oracle GoldenGate for Big Data process.

To replace line feeds with a blank character you could use the following parameter configurations:

```
gg.contentreplaceregex=[\n]
gg.contentreplacestring=CDATA[ ]
```

This changes the column value from:

```
this is
me
```

to :

```
this is me
```

Both values support `CDATA` wrapping. The second value must be wrapped in a `CDATA[]` wrapper because a single blank space will be interpreted as whitespace and trimmed by the Oracle GoldenGate for Big Data configuration layer. In addition, you can configure multiple search a replace strings. For example, you may also want to trim leading and trailing white space out of column values in addition to trimming line feeds from:

```
^\\s+|\\s+$

gg.contentreplaceregex1=^\\s+|\\s+$
gg.contentreplacestring1=CDATA[]
```

to:

1.11 Using Identities in Oracle GoldenGate Credential Store

The Oracle GoldenGate credential store manages user IDs and their encrypted passwords (together known as credentials) that are used by Oracle GoldenGate processes to interact with the local database. The credential store eliminates the need to specify user names and clear-text passwords in the Oracle GoldenGate parameter files. An optional alias can be used in the parameter file instead of the user ID to map to a userid-password pair in the credential store. The credential store is implemented as an autologin wallet within the Oracle Credential Store Framework (CSF). The use of an LDAP directory is not supported for the Oracle GoldenGate credential store. The autologin wallet supports automated restarts of Oracle GoldenGate processes without requiring human intervention to supply the necessary passwords.

In Oracle GoldenGate for Big Data, you specify the alias and domain in the property file not the actual user ID or password.

User credentials are maintained in secure wallet storage

1.11.1 Creating a Credential Store

You can create a credential store for your Big Data environment.

Run the GGSCI `ADD CREDENTIALSTORE` command to create a file called `cwallet.sso` in the `dircrd/` subdirectory of your Oracle GoldenGate installation directory (the default).

You can the location of the credential store (`cwallet.sso` file by specifying the desired location with the `CREDENTIALSTORELOCATION` parameter in the `GLOBALS` file.

For more information about credential store commands, see *Reference for Oracle GoldenGate for Windows and UNIX*.

> **Note:**
>
> Only one credential store can be used for each Oracle GoldenGate instance.

1.11.2 Adding Users to a Credential Store

After you create a credential store for your Big Data environment, you can added users to the store.

Run the GGSCI `ALTER CREDENTIALSTORE ADD USER` *`userid`* `PASSWORD` *`password`* `[ALIAS` *`alias`*`] [DOMAIN` *`domain`*`]` command to create each user, where:

- *`userid`* is the user name. Only one instance of a user name can exist in the credential store unless the `ALIAS` or `DOMAIN` option is used.
- *`password`* is the user's password. The password is echoed (not obfuscated) when this option is used. If this option is omitted, the command prompts for the password, which is obfuscated as it is typed (recommended because it is more secure).
- *`alias`* is an alias for the user name. The alias substitutes for the credential in parameters and commands where a login credential is required. If the `ALIAS` option is omitted, the alias defaults to the user name.

For example:

```
ALTER CREDENTIALSTORE ADD USER scott PASSWORD tiger ALIAS scsm2 domain ggadapters
```

For more information about credential store commands, see *Reference for Oracle GoldenGate for Windows and UNIX*.

1.11.3 Configuring Properties to Access the Credential Store

The Oracle GoldenGate Java Adapter properties file requires specific syntax to resolve user name and password entries in the Credential Store at runtime. For resolving a user name the syntax is the following:

```
ORACLEWALLETUSERNAME alias domain_name
```

For resolving a password the syntax required is the following:

```
ORACLEWALLETPASSWORD alias domain_name
```

The following example illustrate how to configure a Credential Store entry with an alias of `myalias` and a domain of `mydomain`.

Note:

With HDFS Hive JDBCthe user name and password is encrypted.

```
gg.handler.hdfs.hiveJdbcUsername=ORACLEWALLETUSERNAME[myalias mydomain]
gg.handler.hdfs.hiveJdbcPassword=ORACLEWALLETPASSWORD[myalias mydomain]
```

Although the Credential Store is intended to store user name and password pair type credentials, you can apply this functionality more generically. Consider the user name and password entries as accessible values in the Credential Store. Any configuration parameter resolved in the Java Adapter layer (not accessed in the C user exit layer) can be resolved from the Credential Store. This feature is developed to allow you more flexibility to be creative in how you protect sensitive configuration entries.

2 Using the HDFS Handler

This chapter explains the HDFS functionality, and includes examples that you can use to understand this functionality. The Oracle GoldenGate for Big Data Handler for HDFS is designed to stream change capture data into the Hadoop Distributed File System (HDFS).

This chapter includes the following sections:

2.1 Overview

Hadoop Distributed File System (HDFS) is the primary application for Big Data. Hadoop is typically installed on multiple machines which work together as a Hadoop cluster. Hadoop allows users to store very large amounts of data in the cluster that is horizontally scaled across the machines in the cluster. You can then perform analytics on that data using a variety of Big Data applications.

2.2 Hive Handler Support

The Oracle GoldenGate for Big Data 12.2.0.1 release does not include a Hive Handler as was included in the Oracle GoldenGate for Big Data 12.1.2.1.x releases. The 12.1.2.1.x Hive Handler actually provided no direct integration with Hive. The functionality of the Hive Handler was to load operation data from the source trail file into HDFS, partitioned by table, in a Hive friendly delimited text format. The 12.2.0.1 HDFS Handler provides all of the functionality of the previous 12.1.2.1.x Hive Handler.

Hive integration to create tables and update table definitions in the case of DDL events is possible. This functionality is limited to only data formatted as Avro Object Container File format. For more information, see Writing in HDFS in Avro Object Container File Format and HDFS Handler Configuration.

2.3 Writing into HDFS in Sequence File Format

The HDFS SequenceFile is a flat file consisting of binary key and value pairs. You can enable writing data in SequenceFile format by setting the `gg.handler.`*`name`*`.format` property to `sequencefile`. The `key` part of the record is set to null and the actual data is set in the `value` part.

For information about Hadoop SequenceFile, see https://wiki.apache.org/hadoop/SequenceFile.

2.3.1 Integrating with Hive

DDL to create Hive tables should include `STORED as sequencefile` for Hive to consume Sequence Files. Following is a sample create table script:

```
CREATE EXTERNAL TABLE table_name (
  col1 string,
  ...
  ...
  col2 string)
ROW FORMAT DELIMITED
STORED as sequencefile
LOCATION '/path/to/hdfs/file';
```

Note:

If files are intended to be consumed by Hive, then the `gg.handler.`*`name`*`.partitionByTable` property should be set to `true`.

2.3.2 Understanding the Data Format

The data written in the `value` part of each record and is in delimited text format. All of the options described in the Delimited Text Formatter section are applicable to HDFS SequenceFile when writing data to it.

For example:

```
gg.handler.name.format=sequencefile
gg.handler.name.format.includeColumnNames=true
gg.handler.name.format.includeOpType=true
gg.handler.name.format.includeCurrentTimestamp=true
gg.handler.name.format.updateOpKey=U
```

2.4 Runtime Prerequisites

In order to successfully run the HDFS Handler, a Hadoop single instance or Hadoop cluster must be installed, running, and network accessible from the machine running the HDFS Handler. Apache Hadoop is open source and available for download at `http://hadoop.apache.org/`. Follow the Getting Started links for information on how to install a single-node cluster (also called pseudo-distributed operation mode) or a clustered setup (also called fully-distributed operation mode).

2.4.1 Classpath Configuration

Two things must be configured in the `gg.classpath` configuration variable in order for the HDFS Handler to connect to HDFS and run. The first thing is the `HDFS core-`

`site.xml` file and the second are the HDFS client jars. The HDFS client jars must match the version of HDFS that the HDFS Handler is connecting. For a listing of the required client JAR files by version, see HDFS Handler Client Dependencies.

The default location of the `core-site.xml` file is the follow:

`Hadoop_Home``/etc/hadoop`

The default location of the HDFS client jars are the following directories:

`Hadoop_Home``/share/hadoop/common/lib/*`

`Hadoop_Home``/share/hadoop/common/*`

`Hadoop_Home``/share/hadoop/hdfs/lib/*`

`Hadoop_Home``/share/hadoop/hdfs/*`

The `gg.classpath` must be configured exactly as shown. Pathing to the `core-site.xml` should simply contain the path to the directory containing the `core-site.xml`file with no wild card appended. The inclusion of the * wildcard in the path to the `core-site.xml` file will cause it not to be picked up. Conversely, pathing to the dependency jars should include the * wild card character in order to include all of the jar files in that directory in the associated classpath. Do not use `*.jar`. An example of a correctly configured `gg.classpath` variable is the following:

```
gg.classpath=/ggwork/hadoop/hadoop-2.6.0/etc/hadoop:/ggwork/hadoop/hadoop-2.6.0/
share/hadoop/common/lib/*:/ggwork/hadoop/hadoop-2.6.0/share/hadoop/common/*:/ggwork/
hadoop/hadoop-2.6.0/share/hadoop/hdfs/*:/ggwork/hadoop/hadoop-2.6.0/share/hadoop/
hdfs/lib/*
```

The HDFS configuration file `hdfs-site.xml` is also required to be in the classpath if Kerberos security is enabled. The `hdfs-site.xml` file is by default located in the *`Hadoop_Home`*`/etc/hadoop` directory. Either or both files can be copied to another machine if the HDFS Handler is not collocated with Hadoop.

2.4.2 Pluggable Formatters

The HDFS Handler supports all of the Big Data pluggable handlers including which includes:

- JSON
- Delimited Text
- Avro Row
- Avro Operation
- Avro Object Container File Row
- Avro Object Container File Operation
- XML

For more information about formatters, see Using the Pluggable Formatters

2.4.3 HDFS Handler Configuration

The configuration properties of the Oracle GoldenGate for Big Data HDFS Handler are detailed in this section.

Table 2-1 HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handlerlist`	Required	Any string	None	Provides a name for the HDFS Handler. The HDFS Handler name then becomes part of the property names listed in this table.
`gg.handler.`*`name`*`.type=hdfs`	Required	-	-	Selects the HDFS Handler for streaming change data capture into HDFS.
`gg.handler.`*`name`*`.mode`	Optional	`tx | op`	`op`	Selects operation (`op`) mode or transaction (`tx`) mode for the handler. In almost all scenarios, transaction mode results in better performance.
`gg.handler.`*`name`*`.maxFileSize`	Optional	Default unit of measure is bytes. You can stipulate `k`, `m`, or `g` to signify kilobytes, megabytes, or gigabytes respectively. Examples of legal values include `10000`, `10k`, `100m`, `1.1g`.	`1g`	Selects the maximum file size of created HDFS files.
`gg.handler.`*`name`*`.rootFilePath`	Optional	Any path name legal in HDFS.	`/ogg`	The HDFS Handler will create subdirectories and files under this directory in HDFS to store the data streaming into HDFS.

Table 2-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.fileRollInterval`	Optional	The default unit of measure is milliseconds. You can stipulate `ms`, `s`, `m`, `h` to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include `10000`, 10000ms, `10s`, `10m`, or `1.5h`. Values of `0` or less indicate that file rolling on time is turned off.	File rolling on time is off.	The timer starts when an HDFS file is created. If the file is still open when the interval elapses then the file will be closed. A new file will not be immediately opened. New HDFS files are created on a just in time basis.
`gg.handler.`*`name`*`.inactivityRollInterval`	Optional	The default unit of measure is milliseconds. You can stipulate `ms`, `s`, `m`, `h` to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include `10000`, `10000ms`, `10s`, `10.5m`, or `1h`. Values of `0` or less indicate that file inactivity rolling on time is turned off.	File inactivity rolling on time is off.	The timer starts from the latest write to an HDFS file. New writes to an HDFS file restart the counter. If the file is still open when the counter elapses the HDFS file will be closed. A new file will not be immediately opened. New HDFS files are created on a just in time basis.

Table 2-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.fileSuffix`	Optional	Any string conforming to HDFS file name restrictions.	`.txt`	This is a suffix that is added on to the end of the HDFS file names. File names typically follow the format, `{fully qualified table name}{current time stamp}{suffix}`.
`gg.handler.`*`name`*`.partitionByTable`	Optional	`true` | `false`	`true` (data is partitioned by table)	Determines if data written into HDFS should be partitioned by table. If set to `true`, then data for different tables are written to different HDFS files. If se to `false`, then data from different tables is interlaced in the same HDFS file. Must be set to `true` to use the Avro Object Container File Formatter. Set to `false` results in a configuration exception at initialization.
`gg.handler.`*`name`*`.rollOnMetadataChange`	Optional	`true` | `false`	`true` (HDFS files are rolled on a metadata change event)	Determines if HDFS files should be rolled in the case of a metadata change. True means the HDFS file is rolled, false means the HDFS file is not rolled. Must be set to `true` to use the Avro Object Container File Formatter. Set to `false` results in a configuration exception at initialization.
`gg.handler.`*`name`*`.format`	Optional	`delimitedtext` | `json` | `xml` | `avro_row` | `avro_op` | `avro_row_ocf` | `avro_op_ocf` | `sequencefile`	`delimitedtext`	Selects the formatter for the HDFS Handler for how output data will be formatted • `delimitedtext` - Delimited text • `json` - JSON • `xml` - XML • `avro_row` - Avro in row compact format • `avro_op` - Avro in operation more verbose format. • `avro_row_ocf` - Avro in the row compact format written into HDFS in the Avro Object Container File format. • `avro_op_ocf` - Avro in the more verbose format written into HDFS in the Avro Object Container File format. • `sequencefile` - Delimited text written in sequence into HDFS is sequence file format.
`gg.handler.`*`name`*`.includeTokens`	Optional	`true` | `false`	`false`	Set to `true` to include the tokens field and tokens key/values in the output, `false` to suppress tokens output.
`gg.handler.`*`name`*`.partitioner.`*`fully_qualified_table_ name`* Equals one or more column names separated by commas.	Optional	Fully qualified table name and column names must exist.	-	This partitions the data into subdirectories in HDFS in the following format, `par_{`*`column name`*`}={`*`column value`*`}`

Table 2-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.authType`	Optional	`kerberos`	`none`	Setting this property to `kerberos` enables Kerberos authentication.
`gg.handler.`*`name`*`.kerberosKeytabFile`	Optional (Required if `authType=Kerberos`)	Relative or absolute path to a Kerberos `keytab` file.	-	The `keytab` file allows the HDFS Handler to access a password to perform a `kinit` operation for Kerberos security.
`gg.handler.`*`name`*`.kerberosPrincipal`	Optional (Required if `authType=Kerberos`)	A legal Kerberos principal name like `user/FQDN@MY.REALM`.	-	The Kerberos principal name for Kerberos authentication.
`gg.handler.`*`name`*`.schemaFilePath`	Optional		`null`	Set to a legal path in HDFS so that schemas (if available) are written in that HDFS directory. Schemas are currently only available for Avro and JSON formatters. In the case of a metadata change event, the schema will be overwritten to reflect the schema change.
`gg.handler.`*`name`*`.compressionType` Applicable to Sequence File Format only.	Optional	`block | none | record`	`none`	Hadoop Sequence File Compression Type. applicable only if `gg.handler.`*`name.`*`format` is set to `sequencefile`

Table 2-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.compressionCodec` Applicable to Sequence File and writing to HDFS is Avro OCF formats only.	Optional	`org.apache.hadoop.io.compress.DefaultCodec | org.apache.hadoop.io.compress.BZip2Codec | org.apache.hadoop.io.compress.SnappyCodec | org.apache.hadoop.io.compress.GzipCodec`	`org.apache.hadoop.io.compress.DefaultCodec`	Hadoop Sequence File Compression Codec. applicable only if `gg.handler.`*`name.`*`format` is set to `sequencefile`
	Optional	`null | snappy | bzip2 | xz | deflate`	`null`	Avro OCF Formatter Compression Code. This configuration controls the selection of the compression library to be used for Avro OCF files generated. Snappy includes native binaries in the Snappy JAR file and performs a Java-native traversal when performing compression or decompression. Use of Snappy may introduce runtime issue and platform porting issues that you may not experience when working with Java. You may need to perform additional testing to ensure Snappy works on all of your required platforms. Snappy is an open source library so Oracle cannot guarantee its ability to operate on all of your required platforms.

Table 2-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.hiveJdbcUrl`	Optional	A legal URL for connecting to Hive using the Hive JDBC interface.	`null` (Hive integration disabled)	Only applicable to the Avro Object Container File (OCF) Formatter. This configuration value provides a JDBC URL for connectivity to Hive through the Hive JDBC interface. Use of this property requires that you include the Hive JDBC library in the `gg.classpath`. Hive JDBC connectivity can be secured through basic credentials, SSL/TLS, or Kerberos. Configuration properties are provided for the user name and password for basic credentials. See the Hive documentation for how to generate a Hive JDBC URL for SSL/TLS. See the Hive documentation for how to generate a Hive JDBC URL for Kerberos. (If Kerberos is used for Hive JDBC security, it must be enabled for HDFS connectivity. Then the Hive JDBC connection can piggyback on the HDFS Kerberos functionality by using the same Kerberos principal.)
`gg.handler.`*`name`*`.hiveJdbcUserName`	Optional	A legal user name if the Hive JDBC connection is secured through credentials.	Java call result from `System.getProperty(user.name)`	Only applicable to the Avro Object Container File (OCF) Formatter. This property is only relevant if the `hiveJdbcUrl`property is set. It may be required in your environment when the Hive JDBC connection is secured through credentials. Hive requires that Hive DDL operations be associated with a user. If you do not set the value, it defaults to the result of the Java call `System.getProperty(user.name)`
`gg.handler.`*`name`*`.hiveJdbcPassword`	Optional	The fully qualified Hive JDBC driver class name.	`org.apache.hive.jdbc.HiveDriver`	Only applicable to the Avro Object Container File (OCF) Formatter. This property is only relevant if the `hiveJdbcUrl` property is set. The default is the Hive Hadoop2 JDBC driver name. Typically, this property does not require configuration and is provided for use if Apache Hive introduces a new JDBC driver class.

2.4.4 Sample Configuration

The following is sample configuration for the HDFS Handler from the Java Adapter properties file:

```
gg.handlerlist=hdfs
gg.handler.hdfs.type=hdfs
gg.handler.hdfs.mode=tx
gg.handler.hdfs.includeTokens=false
gg.handler.hdfs.maxFileSize=1g
gg.handler.hdfs.rootFilePath=/ogg
gg.handler.hdfs.fileRollInterval=0
gg.handler.hdfs.inactivityRollInterval=0
gg.handler.hdfs.fileSuffix=.txt
gg.handler.hdfs.partitionByTable=true
gg.handler.hdfs.rollOnMetadataChange=true
gg.handler.hdfs.authType=none
gg.handler.hdfs.format=delimitedtext
```

A sample Replicat configuration and a Java Adapter Properties file for an HDFS integration can be found at the following directory:

`GoldenGate_install_directory/AdapterExamples/big-data/hdfs`

2.4.5 Troubleshooting the HDFS Handler

Troubleshooting of the HDFS Handler begins with the contents for the Java `log4j` file. Follow the directions in the Java Logging Configuration to configured the runtime to correctly generate the Java `log4j` log file.

2.4.5.1 Java Classpath

As previously stated, issues with the Java classpath are one of the most common problems. The usual indication of a Java classpath problem is a `ClassNotFoundException` in the Java `log4j` log file. The Java `log4j` log file can be used to troubleshoot this issue. Setting the log level to `DEBUG` allows for logging of each of the jars referenced in the `gg.classpath` object to be logged to the log file. In this way, you can ensure that all of the required dependency jars are resolved. Simply enable `DEBUG` level logging and search the log file for messages like the following:

```
2015-09-21 10:05:10 DEBUG ConfigClassPath:74 - ...adding to classpath: url="file:/
ggwork/hadoop/hadoop-2.6.0/share/hadoop/common/lib/guava-11.0.2.jar
```

2.4.5.2 HDFS Connection Properties

The contents of the HDFS `core-site.xml` file (including default settings) are output to the Java `log4j` log file when the logging level is set to `DEBUG` or `TRACE`. This will show the connection properties to HDFS. Search for the following in the Java `log4j` log file:

```
2015-09-21 10:05:11 DEBUG HDFSConfiguration:58 - Begin - HDFS configuration object
contents for connection troubleshooting.
```

If the `fs.defaultFS` property is set as follows (pointing at the local file system) then the `core-site.xml` file is not properly set in the `gg.classpath` property.

```
  Key: [fs.defaultFS] Value: [file:///].
```

This shows to the `fs.defaultFS` property properly pointed at and HDFS host and port.

```
Key: [fs.defaultFS] Value: [hdfs://hdfshost:9000].
```

2.4.5.3 Handler and Formatter Configuration

The Java `log4j` log file contains information on the configuration state of the HDFS Handler and the selected formatter. This information is output at the `INFO` log level. Sample output appears as follows:

```
2015-09-21 10:05:11 INFO  AvroRowFormatter:156 - **** Begin Avro Row Formatter -
 Configuration Summary ****
  Operation types are always included in the Avro formatter output.
```

```
    The key for insert operations is [I].
    The key for update operations is [U].
    The key for delete operations is [D].
    The key for truncate operations is [T].
  Column type mapping has been configured to map source column types to an
 appropriate corresponding Avro type.
  Created Avro schemas will be output to the directory [./dirdef].
  Created Avro schemas will be encoded using the [UTF-8] character set.
  In the event of a primary key update, the Avro Formatter will ABEND.
  Avro row messages will not be wrapped inside a generic Avro message.
  No delimiter will be inserted after each generated Avro message.
**** End Avro Row Formatter - Configuration Summary ****

2015-09-21 10:05:11 INFO  HDFSHandler:207 - **** Begin HDFS Handler -
 Configuration Summary ****
  Mode of operation is set to tx.
  Data streamed to HDFS will be partitioned by table.
  Tokens will be included in the output.
  The HDFS root directory for writing is set to [/ogg].
  The maximum HDFS file size has been set to 1073741824 bytes.
  Rolling of HDFS files based on time is configured as off.
  Rolling of HDFS files based on write inactivity is configured as off.
  Rolling of HDFS files in the case of a metadata change event is enabled.
  HDFS partitioning information:
    The HDFS partitioning object contains no partitioning information.
HDFS Handler Authentication type has been configured to use [none]
**** End HDFS Handler - Configuration Summary ****
```

2.4.6 Performance Considerations

The HDFS Handler calls the HDFS flush method on the HDFS write stream to flush data to the HDFS datanodes at the end of each transaction in order to maintain write durability. This is an expensive call. Performance can be adversely affected especially in the case of transactions of one or few operations that results in numerous HDFS flush calls.

Performance of the HDFS Handler can be greatly improved by batching multiple small transactions into a single larger transaction. If you have requirements for high performance, you should configure batching functionality provided by either the `Extract` process or the `Replicat` process. For more information, see the Replicat Grouping section.

The HDFS client libraries spawn threads for every HDFS file stream opened by the HDFS Handler. The result is that the number threads executing in the JMV grows proportionally to the number HDFS file streams that are open. Performance of the HDFS Handler can degrade as more HDFS file streams are opened. Configuring the HDFS Handler to write to many HDFS files due to many source replication tables or extensive use of partitioning can result in degraded performance. If the use case requires writing to many tables, then you are advised to enable the roll on time or roll on inactivity features to close HDFS file streams. Closing an HDFS file stream causes the HDFS client threads to terminate and the associated resources can be reclaimed by the JVM.

2.4.7 Security

The HDFS cluster can be secured using Kerberos authentication. Refer to the HDFS documentation for how to secure a Hadoop cluster using Kerberos. The HDFS Handler can connect to Kerberos secured cluster. The HDFS `core-site.xml` should be in the handlers classpath with the `hadoop.security.authentication` property set to `kerberos` and `hadoop.security.authorization` property set to `true`. Additionally, you must set the following properties in the HDFS Handler Java configuration file:

```
gg.handler.name.authType=kerberos
gg.handler.name.keberosPrincipalName=legal Kerberos principal name
gg.handler.name.kerberosKeytabFile=path to a keytab file that contains the password
for the Kerberos principal so that the HDFS Handler can programmatically perform the
Kerberos kinit operations to obtain a Kerberos ticket
```

2.5 Writing in HDFS in Avro Object Container File Format

The HDFS Handler includes specialized functionality to write to HDFS in Avro Object Container File (OCF) format. This Avro OCF is part of the Avro specification and is detailed in the Avro Documentation at

https://avro.apache.org/docs/current/spec.html#Object+Container+Files

Avro OCF format may be a good choice for you because it

- integrates with Apache Hive (raw Avro written to HDFS is not supported by Hive)
- and provides good support for schema evolution. Configure the following to enable writing to HDFS in Avro OCF format.

To write row data to HDFS in Avro OCF format configure the `gg.handler.name.format=avro_row_ocf` property.

To write operation data to HDFS is Avro OCF format configure the `gg.handler.name.format=avro_op_ocf` property.

The HDFS/Avro OCF integration includes optional functionality to create the corresponding tables in Hive and update the schema for metadata change events. The configuration section provides information on the properties to enable integration with Hive. The Oracle GoldenGate Hive integration accesses Hive using the JDBC interface so the Hive JDBC server *must* be running to enable this integration.

2.6 HDFS Handler Certification Matrix

The 12.2.0.1 Oracle GoldenGate for Big Data HDFS Handler is designed to work with the following versions of Apache Hadoop:

- 2.7.x
- 2.6.0
- 2.5.x
- 2.4.x
- 2.3.0
- 2.2.0

The HDFS Handler also works with the following versions of the Hortonworks Data Platform (HDP) that simply packages Apache Hadoop with it:

- HDP 2.4 (HDFS 2.7.1)
- HDP 2.3 (HDFS 2.7.1)

- HDP 2.2 (HDFS 2.6.0)
- HDP 2.1 (HDFS 2.4.0)
- HDP 2.0 (HDFS 2.2.0)

The HDFS Handler also works with the following versions of Cloudera Distribution including Apache Hadoop (CDH):

- CDH 5.7.x (HDFS 2.6.0)
- CDH 5.6.x (HDFS 2.6.0)
- CDH 5.5.x (HDFS 2.6.0)
- CDH 5.4.x (HDFS 2.6.0)
- CDH 5.3. (HDFS 2.5.0)
- CDH 5.2.x (HDFS 2.5.0)
- CDH 5.1.x (HDFS 2.3.0)

2.7 Metadata Change Events

Metadata change events are now handled in the HDFS Handler. The default behavior of the HDFS Handler is to roll the current relevant file in the event of a metadata change event. This behavior allows for the results of metadata changes to at least be separated into different files. File rolling on metadata change is configurable and can be turned off.

To support metadata change events the process capturing changes in the source database must support both DDL changes and metadata in trail. Oracle GoldenGate does not support DDL replication for all database implementations. You should consult the Oracle GoldenGate documentation for their database implementation to understand if DDL replication is supported.

2.8 Partitioning

The HDFS Handler supports partitioning of table data by one or more column values. The configuration syntax to enable partitioning is the following:

```
gg.handler.name.partitioner.fully qualified table name=one mor more column names
separated by commas
```

Consider the following example:

```
gg.handler.hdfs.partitioner.dbo.orders=sales_region
```

This example can result in the following breakdown of files in HDFS:

```
/ogg/dbo.orders/par_sales_region=west/data files
/ogg/dbo.orders/par_sales_region=east/data files
/ogg/dbo.orders/par_sales_region=north/data files
/ogg/dbo.orders/par_sales_region=south/data files
```

Care should be exercised when choosing columns for partitioning. The key is to choose columns that contain only a few (10 or less) possible values and those values are also meaningful for the grouping and analysis of the data. An example of a good partitioning column might be sales regions. An example of a poor partitioning column might be customer date of birth. Configuring partitioning on a column that has many possible values can be problematic. A poor choice can result in hundreds of HDFS file streams being opened and performance can degrade for the reasons discussed in the Performance section. Additionally, poor partitioning can result in problems while performing analysis on the data. Apache Hive requires that all where clauses specify partition criteria if the Hive data is partitioned.

2.9 Common Pitfalls

The most common problems encountered are Java classpath issues. The Oracle HDFS Handler requires certain HDFS client libraries to be resolved in its classpath as a prerequisite for streaming data to HDFS.

For a listing of the required client JAR files by version, see HDFS Handler Client Dependencies. The HDFS client jars *do not* ship with the Oracle GoldenGate for Big Data product. The HDFS Handler supports multiple versions of HDFS and it is required that the HDFS client jars be the same version as the HDFS version to which the HDFS Handler is connecting. The HDFS client jars are open source and freely available to download from sites such as the Apache Hadoop site or the maven central repository.

In order to establish connectivity to HDFS, the HDFS `core-site.xml` file needs to be in the classpath of the HDFS Handler. If the `core-site.xml` file is not in the classpath the HDFS client code defaults to a mode that attempts to write to the local file system. Writing to the local file system instead of HDFS can in fact be an advantageous for troubleshooting, building a point of contact (POC), or as a step in the process of building an HDFS integration.

Another common concern is that data streamed to HDFS using the HDFS Handler is often not immediately available to Big Data analytic tools such as Hive. This behavior commonly occurs when the HDFS Handler is in possession of an open write stream to an HDFS file. HDFS writes in blocks of 128MB by default. HDFS blocks under construction are not always visible to analytic tools. Additionally, inconsistencies between file sizes when using the `-ls`, `-cat`, and `-get` commands in the HDFS shell are commonly seen. This is an anomaly of HDFS streaming and is discussed in the HDFS specification. This anomaly of HDFS leads to a potential 128MB per file blind spot in analytic data. This may not be an issue if you have a steady stream of Replication data and do not require low levels of latency for analytic data from HDFS. However, this may be a problem in some use cases. Closing the HDFS write stream causes the block writing to finalize. Data is immediately visible to analytic tools and file sizing metrics become consistent again. So the new file rolling feature in the HDFS Handler can be used to close HDFS writes streams thus making all data visible.

Caution:

The file rolling solution may present its own potential problems. Extensive use of file rolling can result in lots of small files in HDFS. Lots of small files in HDFS can be its own problem resulting in performance issues in analytic tools.

You may also notice the HDFS inconsistency problem in the following scenarios.

- The HDFS Handler process crashes.
- A forced shutdown is called on the HDFS Handler process.

- A network outage or some other issue causes the HDFS Handler process to abend.

In each of these scenarios it is possible for the HDFS Handler to end without explicitly closing the HDFS write stream and finalizing the writing block. HDFS in its internal process will ultimately recognize that the write stream has been broken and HDFS will finalize the write block. However, in this scenario, users may experience a short term delay before the HDFS process finalizes the write block.

2.10 Best Practices

It is considered a Big Data best practice for the HDFS cluster to operate on dedicated servers called cluster nodes. Edge nodes are server machines that host the applications to stream data to and retrieve data from the HDFS cluster nodes. This physical architecture delineation between the HDFS cluster nodes and the edge nodes provides a number of benefits including the following:

- The HDFS cluster nodes are not competing for resources with the applications interfacing with the cluster.
- HDFS cluster nodes and edge nodes likely have different requirements. This physical topology allows the appropriate hardware to be tailored to the specific need.

It is a best practice for the HDFS Handler to be installed and running on an edge node and streaming data to the HDFS cluster using network connection. The HDFS Handler can run on any machine that has network visibility to the HDFS cluster. The installation of the HDFS Handler on an edge node requires that the `core-site.xml` files and the dependency jars be copied to the edge node so that the HDFS Handler can access them. The HDFS Handler can also run collocated on a HDFS cluster node if required.

3

Using the HBase Handler

The Oracle GoldenGate for Big Data Handler for HBase allows you to populate HBase tables from existing Oracle GoldenGate supported sources.

This chapter contains the following sections:

- Overview
- HBase Handler Certification Matrix
- Detailed Functionality
- Runtime Prerequisites
- Metadata Change Events
- Common Pitfalls

3.1 Overview

HBase is an open source Big Data application that emulates much of the functionality of a relational database management system (RDBMS). Hadoop is specifically designed to store large amounts of unstructured data. Conversely, data stored in databases and being replicated through Oracle GoldenGate is highly structured. HBase provides a method of maintaining the important structure of data, while taking advantage of the horizontal scaling that is offered by the Hadoop Distributed File System (HDFS).

3.2 HBase Handler Certification Matrix

Cloudera HBase 5.4.*x* and later did not fully adopt the Apache HBase 1.0.0 client interface so it is not fully in sync with the Apache HBase code line to provide reverse compatibility in that HBase client interface. This means that Cloudera HBase broke binary compatibility with the new HBase 1.0.0 interface resulting in `NoSuchMethodError` when integrating with the Oracle GoldenGate for Big Data HBase Handler. This can be solved one of the following two ways:

- Configure the HBase Handler to use the 0.98.x HBase interface by setting the HBase Handler configuration property, `hBase98Compatible`, to `true`.
- Alternatively, you can use the Apache HBase client libraries when connecting to CDH 5.4.*x* and later HBase.

The 12.2.0.1 Oracle GoldenGate for Big Data HBase Handler is designed to work with the following:

Distribution	Version
Apache HBase	0.98.*x* and 0.96.*x* when you set the `hBase98Compatible` property to true 1.1.*x* and 1.0.*x*
Hortonworks Data Platform (HDP) including Apache HBase	HDP 2.4 (HBase 1.1.2) HDP 2.3 (HBase 1.1.1) HDP 2.2 (HBase 0.98.4) when you set the `hBase98Compatible` property to `true`.
Cloudera Apache Hadoop (CDH)	CDH 5.7.*x* (HBase 1.2.0) when you set the `hBase98Compatible` property to `true`. CDH 5.6.*x* (HBase 1.0.0) when you set the `hBase98Compatible` property to `true`. CDH 5.5.*x* (HBase 1.0.0) when you set the `hBase98Compatible` property to `true`. CDH 5.4.*x* (HBase 1.0.0) when you set the `hBase98Compatible` property to `true`. CDH 5.3.*x* (HBase 0.98.6) when you set the `hBase98Compatible` property to `true`. CDH 5.2.*x* (HBase 0.98.6) when you set the `hBase98Compatible` property to `true`. CDH 5.1.*x* (HBase 9.98.1) when you set the `hBase98Compatible` property to `true`.

3.3 Detailed Functionality

The HBase Handler takes operations from the source trail file and creates corresponding tables in HBase, and then loads change capture data into those tables.

HBase Table Names

Table names created in an HBase map to the corresponding table name of the operation from the source trail file. It is case-sensitive.

HBase Table Namespace

For two part table names (schema name and table name), the schema name maps to the HBase table namespace. For a three part table name like `Catalog.Schema.MyTable`, the create HBase namespace would be `Catalog_Schema`. HBase table namespaces are case sensitive. A NULL schema name is supported and maps to the default HBase namespace.

HBase Row Key

HBase has a similar concept of the database primary keys called the HBase row key. The HBase row key is the unique identifier for a table row. HBase only supports a single row key per row and it cannot be empty or NULL. The HBase Handler maps the primary key value into the HBase row key value. If the source table has multiple primary keys, then the primary key values are concatenated, separated by a pipe delimiter (`|`).You can configure the HBase row key delimiter.

The source table *must* have at least one primary key column. Replication of a table without a primary key causes the HBase Handler to abend.

HBase Column Family

HBase has the concept of a column family. A column family is a grouping mechanism for column data. Only a single column family is supported. Every HBase column must belong to a single column family. The HBase Handler provides a single column family per table that defaults to `cf`. The column family name is configurable by you. However, once a table is created with a specific column family name, reconfiguration of the column family name in the HBase example without first modify or dropping the table results in an abend of the Oracle GoldenGate Extract and Replicat processes.

3.4 Runtime Prerequisites

HBase must be up and running either collocated with the HBase Handler process or on a machine that is network connectable from the machine hosting the HBase Handler process. Additionally the underlying HDFS single instance or clustered instance serving as the repository for HBase data must be up and running.

3.4.1 Classpath Configuration

You must include two things in the `gg.classpath` configuration variable in order for the HBase Handler to connect to HBase and stream data. The first is the `hbase-site.xml` file and the second are the HBase client jars. The HBase client jars must match the version of HBase to which the HBase Handler is connecting. The HBase client jars are *not* shipped with the Oracle GoldenGate for Big Data product.

HBase Handler Client Dependencies includes the listing of required HBase client jars by version.

The default location of the `hbase-site.xml` file is `*HBase_Home*/conf`.

The default location of the HBase client JARs is `*HBase_Home*/lib/*`.

If the HBase Handler is running on Windows, follow the Windows classpathing syntax.

The `gg.classpath` must be configured exactly as described. Pathing to the `hbase-site.xml` should simply contain the path with no wild card appended. The inclusion of the * wildcard in the path to the `hbase-site.xml` file will cause it not to be accessible. Conversely, pathing to the dependency jars should include the * wild card character in order to include all of the jar files in that directory in the associated classpath. Do not use `*.jar`. An example of a correctly configured `gg.classpath` variable is the following:

```
gg.classpath=/var/lib/hbase/lib/*:/var/lib/hbase/conf
```

3.4.2 Pluggable Formatters

Pluggable formatters are not applicable to the HBase Handler. Data is streamed to HBase using the proprietary HBase client interface.

3.4.3 HBase Handler Configuration

Table 3-1   HBase Handler Configuration Parameters

Table 3-1 (Cont.) HBase Handler Configuration Parameters

Parameters	Required/ Optional	Legal Values	Default	Explanation
`gg.handlerlist`	Required	Any string	None	Provides a name for the HBase Handler. The HBase Handler name is then becomes part of the property names listed in this table.
`gg.handler.`*`name`*`.type=hbase`	Required	-	-	Selects the HBase Handler for streaming change data capture into HBase
`gg.handler.`*`name`*`.hBaseColumnFamilyName`	Optional	Any String legal for an HBase column family name	`cf`	Column family is a grouping mechanism for columns in HBase. The HBase Handler only supports a single column family in the 12.2 release.
`gg.handler.`*`name`*`.includeTokens`	Optional	`true | false`	`false`	True indicates that token values will be included in the output to HBase. False means token values will be not be included.
`gg.handler.`*`name`*`.keyValueDelimiter`	Optional	Any string	`=`	Provides a delimiter between key values in a map. For example, `key=value,key1=value1,key2=value2`. Tokens are mapped values. Configuration value supports `CDATA[]` wrapping.
`gg.handler.`*`name`*`.keyValuePairDelimiter`	Optional	Any string	`,`	Provides a delimiter between key value pairs in a map. For example, `key=value,key1=value1,key2=value2key=value,key1=value1,key2=value2`. Tokens are mapped values. Configuration value supports `CDATA[]` wrapping.

Table 3-1 (Cont.) HBase Handler Configuration Parameters

Parameters	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.encoding`	Optional	Any encoding name or alias supported by Java.[1] For a list of supported options, visit the Oracle Java Documentation website at `https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html`	The native system encoding of the machine hosting the Oracle GoldenGate process	Determines the encoding of values written the HBase. HBase values are written as bytes.
`gg.handler.`*`name`*`.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Provides configuration for how the HBase Handler should handle update operations that change a primary key. Primary key operations can be problematic for the HBase Handler and require special consideration by you. • `abend` - indicates the process will abend • `update` - indicates the process will treat this as a normal update • `delete-insert` - indicates the process will treat this as a delete and an insert. The full before image is required for this feature to work properly. This can be achieved by using full supplemental logging in Oracle Database. Without full before and after row images the insert data will be incomplete.

Table 3-1 (Cont.) HBase Handler Configuration Parameters

Parameters	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.nullValueRepresentation`	Optional	Any string	`NULL`	Allows you to configure what will be sent to HBase in the case of a NULL column value. The default is `NULL`. Configuration value supports `CDATA[]` wrapping.
`gg.handler.name.authType`	Optional	`kerberos`	None	Setting this property to `kerberos` enables Kerberos authentication.
`gg.handler.name.kerberosKeytabFile`	Optional (Required if `authType=kerberos`)	Relative or absolute path to a Kerberos `keytab` file	-	The `keytab` file allows the HDFS Handler to access a password to perform a `kinit` operation for Kerberos security.
`gg.handler.name.kerberosPrincipal`	Optional (Required if `authType=kerberos`)	A legal Kerberos principal name (for example, `user/FQDN@MY.REALM`)	-	The Kerberos principal name for Kerberos authentication.
`gg.handler.name.hBase98Compatible`	Optional	`true` | `false`	`false`	Set this configuration property to `true` to enable integration with the HBase 0.98.*x* and 0.96.*x* releases. You can use this to solve compatibility problems with Cloudera CDH 5.7.*x*, 5.6.*x*, 5.5.*x* and 5.4.*x*. For more information, see HBase Handler Certification Matrix
`gg.handler.name.rowkeyDelimiter`	Optional	Any string	|	Configures the delimiter between primary key values from the source table when generating the HBase `rowkey`. This property supports `CDATA[]` wrapping of the value to preserve whitespace if the user wishes to delimit incoming primary key values with a character or characters determined to be whitespace.

[1] For more Java information, see *Java Internalization Support* at `https://docs.oracle.com/javase/8/docs/technotes/guides/intl/`.

3.4.4 Sample Configuration

The following is sample configuration for the HBase Handler from the Java Adapter properties file:

```
gg.handlerlist=hbase
gg.handler.hbase.type=hbase
gg.handler.hbase.mode=tx
gg.handler.hbase.hBaseColumnFamilyName=cf
gg.handler.hbase.includeTokens=true
gg.handler.hbase.keyValueDelimiter=CDATA[=]
gg.handler.hbase.keyValuePairDelimiter=CDATA[,]
gg.handler.hbase.encoding=UTF-8
gg.handler.hbase.pkUpdateHandling=abend
gg.handler.hbase.nullValueRepresentation=CDATA[NULL]
gg.handler.hbase.authType=none
```

A sample Replicat configuration and a Java Adapter properties file for an HBase integration can be found at the following directory:

`GoldenGate_install_directory/``AdapterExamples/big-data/hbase`

3.4.5 Troubleshooting the HBase Handler

Troubleshooting of the HBase Handler begins with the contents for the Java `log4j` file. Follow the directions in the Java Logging Configuration to configure the runtime to correctly generate the Java `log4j` log file.

3.4.5.1 Java Classpath

Issues with the Java classpath are one of the most common problems. An indication of a classpath problem is a `ClassNotFoundException` in the Java `log4j` log file. The Java `log4j` log file can be used to troubleshoot this issue. Setting the log level to `DEBUG` allows for logging of each of the jars referenced in the `gg.classpath` object to be logged to the log file. You can make sure that all of the required dependency jars are resolved. Simply enable `DEBUG` level logging and search the log file for messages like the following:

```
2015-09-29 13:04:26 DEBUG ConfigClassPath:74 -  ...adding to classpath:
 url="file:/ggwork/hbase/hbase-1.0.1.1/lib/hbase-server-1.0.1.1.jar"
```

3.4.5.2 HBase Connection Properties

The contents of the HDFS `hbase-site.xml` file (including default settings) are output to the Java `log4j` log file when the logging level is set to `DEBUG` or `TRACE`. This will show the connection properties to HBase. Search for the following in the Java `log4j` log file.

```
2015-09-29 13:04:27 DEBUG HBaseWriter:449 - Begin - HBase configuration object
contents for connection troubleshooting.
Key: [hbase.auth.token.max.lifetime] Value: [604800000].
```

A common error is for the `hbase-site.xml` file to be either not included in the classpath or a pathing error to the `hbase-site.xml`. In this case the HBase Handler will not be able to establish a connection to HBase and the Oracle GoldenGate process will abend. The following error will be reported in the Java `log4j` log.

```
2015-09-29 12:49:29 ERROR HBaseHandler:207 - Failed to initialize the HBase handler.
org.apache.hadoop.hbase.ZooKeeperConnectionException: Can't connect to ZooKeeper
```

Verify that the classpath correctly includes the `hbase-site.xml` file and that HBase is running.

3.4.5.3 Logging of Handler Configuration

The Java `log4j` log file contains information on the configuration state of the HBase Handler. This information is output at the `INFO` log level. Sample output appears as follows:

```
2015-09-29 12:45:53 INFO HBaseHandler:194 - **** Begin HBase Handler - Configuration
Summary ****
  Mode of operation is set to tx.
  HBase data will be encoded using the native system encoding.
  In the event of a primary key update, the HBase Handler will ABEND.
  HBase column data will use the column family name [cf].
  The HBase Handler will not include tokens in the HBase data.
  The HBase Handler has been configured to use [=] as the delimiter between keys and
values.
  The HBase Handler has been configured to use [,] as the delimiter between key
values pairs.
  The HBase Handler has been configured to output [NULL] for null values.
Hbase Handler Authentication type has been configured to use [none]
```

3.4.6 Performance Considerations

At each transaction commit, the HBase Handler performs a flush call to flush any buffered data to the HBase region server. This must be done to maintain write durability. Flushing to the HBase region server is an expensive call and performance can be greatly improved by employing the Java Adapter batching functionality if using the Extract process, or by using the native Replicat `GROUPTRANSOPS` batching functionality. The result is similar in both cases. The Extract process batch functionality uses Java Adapter based grouping controlled in the Java Adapter Properties file. The Replicat based batching uses the `GROUPTRANSOPS` configuration syntax in the Replicat configuration file.

Operations from multiple transactions are grouped together into a larger transaction, and it is only at the end of the grouped transaction that transaction commit is executed.

3.4.7 Security

HBase connectivity can be secured using Kerberos authentication. Follow the associated documentation for the HBase release to secure the HBase cluster. The Oracle GoldenGate HBase Handler can connect to Kerberos secured cluster. The HBase `hbase-site.xml` should be in handlers classpath with "`hbase.security.authentication`" property set to `kerberos` and "`hbase.security.authorization`" property set to `true`.

Additionally, you must set the following properties in the Oracle GoldenGate HBase Handler Java configuration file:

```
gg.handler.{name}.authType=kerberos
gg.handler.{name}.keberosPrincipalName={legal Kerberos principal name}
gg.handler.{name}.kerberosKeytabFile={path to a keytab file that contains the
password for the Kerberos principal so that the Oracle GoldenGate HDFS handler can
programmatically perform the Kerberos kinit operations to obtain a Kerberos ticket}.
```

3.5 Metadata Change Events

Oracle GoldenGate 12.2 includes metadata in trail and can handle metadata change events at runtime. The HBase Handler can handle metadata change events at runtime as well. One of the most common scenarios is the addition of a new column. The result in HBase will be that the new column and its associated data will begin being streamed to HBase after the metadata change event.

Note:

Oracle GoldenGate 12.2 metadata change events are only written to the trail by Oracle Database.

It is important to understand that in order to enable metadata change events the entire Replication chain must be upgraded to Oracle GoldenGate 12.2. The 12.2 HBase Handler can work with trail files produced by Oracle GoldenGate 12.1 and greater. However, these trail files do not include metadata in trail and therefore metadata change events cannot be handled at runtime.

3.6 Common Pitfalls

HBase has been experiencing changes to the client interface in the last few releases. HBase 1.0.0 introduced a new recommended client interface and the 12.2 HBase Handler has moved to the new interface to keep abreast of the most current changes. However, this does create a backward compatibility issue. The HBase Handler is not compatible with HBase versions older than 1.0.0. If an Oracle GoldenGate integration is required with 0.99.x or older version of HBase, this can be accomplished using the 12.1.2.1.x HBase Handler. Contact Oracle Support to obtain a ZIP file of the 12.1.2.1.x HBase Handler.

Common errors on the initial setup of the HBase Handler are classpathing issues. The typical indicator of such a problem is occurrences of the ClassNotFoundException in the Java `log4j` log file. The HBase client jars do *not* ship with the Oracle GoldenGate for Big Data product. You must resolve the required HBase client jars. Appendix (a reference here) includes the listing of HBase client jars for each supported version. Either the `hbase-site.xml` or one or more of the required client jars are not included in the classpath. For instructions on configuring the classpath of the HBase Handler, see section Classpath Configuration.

4 Using the Flume Handler

The chapter includes the following sections:

4.1 Overview

The Oracle GoldenGate for Big Data Flume Handler is designed to stream change capture data from a Oracle GoldenGate trail to a Flume source. Apache Flume is an open source application for which the primary purpose is streaming data into Big Data applications. The Flume architecture contains three main components namely, Sources, Channels and Sinks which collectively make a pipeline for data. A Flume source publishes the data to a Flume channel. A Flume sink retrieves the data out of a Flume channel and streams the data to different targets. A Flume Agent is a container process that owns and manages a source, channel and sink. A single Flume installation can host many agent processes. The Flume Handler can stream data from a trail file to Avro or Thrift RPC Flume sources.

4.2 Runtime Prerequisites

In order to run the Flume Handler, a Flume Agent configured with an Avro or Thrift Flume source must be up and running. Oracle GoldenGate can be collocated with Flume or located on a different machine. If located on a different machine the host and port of the Flume source must be reachable via network connection. For instructions on how to configure and start a Flume Agent process, see the *Flume User Guide* at

`https://flume.apache.org/releases/content/1.6.0/FlumeUserGuide.pdf`

4.3 Classpath Configuration

You must configure two things in the `gg.classpathconfiguration` variable for the Flume Handler to connect to the Flume source and run. The first thing is the Flume Agent configuration file and the second are the Flume client jars. The Flume Handler uses the contents of the Flume Agent configuration file to resolve the host, port, and source type for the connection to Flume source. The Flume client libraries do *not* ship with Oracle GoldenGate for Big Data. The Flume client library versions must match the version of Flume to which the Flume Handler is connecting. For a listing for the required Flume client JAR files by version, see Flume Handler Client Dependencies.

The Oracle GoldenGate property, `gg.classpath`, needs to be set to include the following default locations:

- The default location of the `core-site.xml` file is `Flume_Home/conf`.
- The default location of the Flume client jars is `Flume_Home/lib/*`.

The `gg.classpath` must be configured exactly as shown in the preceding example. Pathing to the Flume Agent configuration file should simply contain the path with no wild card appended. The inclusion of the `*wildcard` in the path to the Flume Agent configuration file will cause it not to be accessible. Conversely, pathing to the dependency jars should include the * wild card character in order to include all of the jar files in that directory in the associated classpath. Do not use `*.jar`. An example of a correctly configured `gg.classpath` variable is the following:

```
gg.classpath=dirprm/:/var/lib/flume/lib/*
```

If the Oracle GoldenGate for Big Data Flume Handler and Flume are not collocated, then the Flume Agent configuration file and the Flume client libraries will need to be copied to the machine hosting the Oracle GoldenGate for Big Data Flume Handler process.

4.4 Pluggable Formatters

The Oracle GoldenGate for Big Data Flume Handler supports all of the Big Data formatters included with the Oracle GoldenGate for Big Data release. The formatters are:

- Avro Row
- Avro Operation
- JSON
- XML
- Delimited Text

4.5 Flume Handler Configuration

The configuration properties for 12.2.0.1 Flume Handler are outlined as follows:

Property Name	Property Value	Mandatory	Description
`gg.handlerlist`	`flumehandler` (choice of any name)	Yes	List of handlers. Only one is allowed with grouping properties `ON`.
`gg.handler.flumehandler.type`	`flume`	Yes	Type of handler to use.
`gg.handler.flumehandler.format`	Formatter class or short code	No. Defaults to `delimitedtext`	The Formatter to be used. Can be one of the following: • `avro_row` • `avro_op` • `delimitedtext` • `xml` • `json` Alternatively, it is possible to write a custom formatter and include the fully qualified class name here.
`gg.handler.flumehandler.RpcClientPropertiesFile`	Any choice of filename	No. Defaults to `default-flume-rpc.properties`	Either the default `default-flume-rpc.properties` or a specified custom RPC client properties file should exist in the classpath.
`gg.handler.flumehandler.mode`	`op|tx`	No. Defaults to `op`	Operation mode or Transaction Mode. Java Adapter grouping options can be used only in tx mode.
`gg.handler.flumehandler.EventHeaderClass`	A custom implementation fully qualified class name	No. Defaults to `DefaultFlumeEventHeader`	Class to be used which defines what headers properties are to be added to a flume event.
`gg.handler.flumehandler.EventMapsTo`	`op|tx`	No. Defaults to `op`	Defines whether each flume event would represent an operation or a transaction. If `handler mode = op`, `EventMapsTo` will always be `op`.
`gg.handler.flumehandler.PropagateSchema`	`true|false`	No. Defaults to `false`	When set to `true`, the Flume handler will begin to publish schema events.
`gg.handler.flumehandler.includeTokens`	`true|false`	No. Defaults to `false`	When set to `true`, includes token data from the source trail files in the output. When set to `false` to excludes the token data from the source trail files in the output.

4.6 Sample Configuration

```
gg.handlerlist = flumehandler
gg.handler.flumehandler.type = flume
gg.handler.flumehandler.RpcClientPropertiesFile=custom-flume-rpc.properties
gg.handler.flumehandler.format =avro_op
gg.handler.flumehandler.mode =tx
gg.handler.flumehandler.EventMapsTo=tx
gg.handler.flumehandler.PropagateSchema =true
gg.handler.flumehandler.includeTokens=false
```

A sample Replicat configuration and a Java Adapter properties file for a Flume integration can be found at the following directory:

`*GoldenGate_install_directory*/AdapterExamples/big-data/flume`

4.7 Troubleshooting

4.7.1 Java Classpath

Issues with the Java classpath are one of the most common problems. The indication of a classpath problem is a `ClassNotFoundException` in the Oracle GoldenGate Java `log4j` log file. The Java `log4j` log file can be used to troubleshoot this issue. Setting the log level to `DEBUG` allows for logging of each of the jars referenced in the `gg.classpath` object to be logged to the log file. In this way you can make sure that all of the required dependency jars are resolved.

4.7.2 Flume Flow Control Issues

The Flume Handler may in certain scenarios write to the Flume source faster than the Flume sink can dispatch messages. In the scenario, the Flume Handler will work for a while but once Flume can no longer accept messages it will abend. The cause in the Oracle GoldenGate Java log file will likely be an `EventDeliveryException` indicating the Flume Handler was unable to send an event. Check the Flume log to for the exact cause of the problem. You may be able to reconfigure the Flume channel to increase capacity or increase the configuration for Java heap if the Flume Agent is experiencing an `OutOfMemoryException`. However, this may not entirely solve the problem. If the Flume Handler can push data to the Flume source faster than messages are dispatched by the Flume sink, any change may simply extend the period the Oracle GoldenGate Handler can run before failing.

4.7.3 Flume Agent Configuration File Not Found

The Flume Handler will abend at start up if the Flume Agent configuration file is not in the classpath. The result is generally a `ConfigException` listing the issue as an error loading the Flume producer properties. Check the `gg.handler.{*name*}. RpcClientProperites` configuration file to ensure that the naming of the Flume Agent Properties file is correct. Check the GoldenGate `gg.classpath` properties to ensure that the classpath contains the directory containing the Flume Agent properties file. Also check the classpath to ensure that the path to the Flume Agent properties file does not end with a wildcard "*" character.

4.7.4 Flume Connection Exception

The Flume Handler will abend at start up if it is unable to make a connection to the Flume source. The root cause of this problem will likely be reported as an `IOExeption` in the Oracle GoldenGate Java `log4j` file indicating a problem connecting to Flume at a given host and port. Check the following:

- The Flume Agent process is running.
- That the Flume agent configuration file that the Oracle for Big Data Flume Handler is accessing contains the correct host and port.

4.7.5 Other Failures

Review the contents of the Oracle GoldenGate Java `log4j` file.

4.8 Data Mapping of Operations to Flume Events

This section explains how operation data from the Oracle GoldenGate trail file is mapped by the Flume Handler into Flume Events based on different configurations. A Flume Event is a unit of data that flows through a Flume agent. The Event flows from Source to Channel to Sink, and is represented by an implementation of the Event interface. An Event carries a payload (byte array) that is accompanied by an optional set of headers (string attributes).

4.8.1 Operation Mode

The configuration for the Flume Handler is the following in the Oracle GoldenGate Java configuration file.

```
gg.handler.{name}.mode=op
```

The data for each individual operation from Oracle GoldenGate trail file maps into a single Flume Event. Each event is immediately flushed to Flume. Each Flume Event will have the following headers.

- `TABLE_NAME:` The table name for the operation.
- `SCHEMA_NAME`: The catalog name (if available) and the schema name of the operation.
- `SCHEMA_HASH`: The hash code of the Avro schema. (Only applicable for Avro Row and Avro Operation formatters.)

4.8.2 Transaction Mode and `EventMapsTo` Operation

The configuration for the Flume Handler is the following in the Oracle GoldenGate Java configuration file.

```
gg.handler.flume_handler_name.mode=tx
gg.handler.flume_handler_name.EventMapsTo=op
```

The data for each individual operation from Oracle GoldenGate trail file maps into a single Flume Event. Events are flushed to Flume at transaction commit. Each Flume Event will have the following headers.

- `TABLE_NAME`: The table name for the operation.
- `SCHEMA_NAME`: The catalog name (if available) and the schema name of the operation.
- `SCHEMA_HASH`: The hash code of the Avro schema. (Only applicable for Avro Row and Avro Operation formatters.)

It is suggested to use this mode when formatting data as Avro or delimited text. It is important to understand that configuring Extract or Replicat batching functionality will increase the number of operations processed in a transaction.

4.8.3 Transaction Mode and `EventMapsTo` Transaction

The configuration for the Flume Handler is the following in the Oracle GoldenGate Java configuration file.

```
gg.handler.flume_handler_name.mode=tx
gg.handler.flume_handler_name.EventMapsTo=tx
```

The data for all operations for a transaction from the source trail file are concatenated and mapped into a single Flume Event. The event is flushed at transaction commit. Each Flume Event has the following headers.

- `GG_TRANID`: The transaction ID of the transaction
- `OP_COUNT`: The number of operations contained in this Flume payload event

It is suggested to use this mode only when using self describing formats such as JSON or XML. In is important to understand that configuring Extract or Replicat batching functionality will increase the number of operations processed in a transaction.

4.9 Flume Handler Certification Matrix

The Oracle GoldenGate for Big Data Flume Handler works with versions 1.6.x, 1.5.x and 1.4.x of Apache Flume. Compatibility with versions of Flume before 1.4.0 is not guaranteed.

The Flume Handler is compatible with the following versions of the Hortonworks Data Platform (HDP):

- HDP 2.4 (Flume 1.5.2)
- HDP 2.3 (Flume 1.5.2)
- HDP 2.2 (Flume 1.5.2)
- HDP 2.1 (Flume 1.4.0)

The Flume Handler is compatible with the following versions of the Cloudera Distributions of Hadoop (CDH):

- CDH 5.7.*x* (Flume 1.6.0)
- CDH 5.6.*x* (Flume 1.6.0)
- CDH 5.5.*x* (Flume 1.6.0)
- CDH 5.4.*x* (Flume 1.5.0)
- CDH 5.3.*x* (Flume 1.5.0)
- CDH 5.2.*x* (Flume 1.5.0)
- CDH 5.1.*x* (Flume 1.5.0)

4.10 Performance Considerations

- Replicat based grouping is recommended to be used to improve performance.
- Extract based grouping uses the grouping in the Java Adapter. Message size based grouping with Java Adapter may be slower than operation count based grouping. If Adapter based grouping is really needed, operation count based grouping is recommended.
- Transaction mode with `gg.handler.`*`flume_handler_name`*`.EventMapsTo=tx` setting is recommended for best performance.

- The maximum heap size of the Flume Handler may affect performance. Too little heap may results in frequent garbage collections by the JVM. Increasing the maximum heap size of the JVM in the Oracle GoldenGate Java properties file may improve performance.

4.11 Metadata Change Events

The Oracle GoldenGate for Big Data 12.2.0.1 Flume Handler is adaptive to changes in DDL at the source. However, this functionality depends on the source replicated database and the upstream Oracle GoldenGate Capture process to capture and replicate DDL events. This feature is not immediately available for all database implementations in Oracle GoldenGate 12.2. Refer to the Oracle GoldenGate documentation for your database implementation for information about DDL replication.

Whenever a metadata change occurs at the source, the flume handler will notify the associated formatter of the metadata change event. Any cached schema that the formatter is holding for that table will be deleted. The next time the associated formatter encounters an operation for that table the schema will be regenerated.

4.12 Example Flume Source Configuration

4.12.1 Avro Flume Source

The following is sample configuration for an Avro Flume source from the Flume Agent configuration file:

```
client.type = default
hosts = h1
hosts.h1 = host_ip:host_port
batch-size = 100
connect-timeout = 20000
request-timeout = 20000
```

4.12.2 Thrift Flume Source

The following is sample configuration for an Avro Flume source from the Flume Agent configuration file:

```
client.type = thrift
hosts = h1
hosts.h1 = host_ip:host_port
```

4.13 Advanced Features

4.13.1 Schema Propagation

The Flume Handler can propagate schemas to Flume. This is currently only supported for the Avro Row and Operation formatters. To enable this feature set the following property:

```
gg.handler.flume_handler_name.propagateSchema=true
```

The Avro Row or Operation Formatters generate Avro schemas on a just in time basis. Avro schemas are generated the first time an operation for a table is encountered. A metadata change event results in the schema reference being for a table being cleared and thereby a new schema is generated the next time an operation is encountered for that table.

When schema propagation is enabled the Flume Handler will propagate schemas an Avro Event when they are encountered.

Default Flume Schema Event headers for Avro include the following information:

- `SCHEMA_EVENT`: TRUE
- `GENERIC_WRAPPER`: TRUE/FALSE
- `TABLE_NAME`: The table name as seen in the trail
- `SCHEMA_NAME`: The catalog name (if available) and the schema name
- `SCHEMA_HASH`: The hash code of the Avro schema

4.13.2 Security

Kerberos authentication for the Oracle GoldenGate for Big Data Flume Handler connection to the Flume source is possible, but this feature is only supported in Flume 1.6.0 (and assumed higher) using the Thrift Flume source. This feature is enabled solely by changing the configuration of the Flume source in the Flume Agent configuration file. Following is an example of the Flume source configuration from the Flume Agent configuration file showing how to enable Kerberos authentication. The Kerberos principal name of the client and the server need to be provided. The path to a Kerberos keytab file must be provided so that the password of the client principal can be resolved at runtime. For information on how to administrate Kerberos, Kerberos principals and their associated passwords, and the creation of a Kerberos keytab file, refer to the Kerberos documentation.

```
client.type = thrift
hosts = h1
hosts.h1 =host_ip:host_port
kerberos=true
client-principal=flumeclient/client.example.org@EXAMPLE.ORG
client-keytab=/tmp/flumeclient.keytab
server-principal=flume/server.example.org@EXAMPLE.ORG
```

4.13.3 Fail Over Functionality

It is possible to configure the Flume Handler so that it will fail over in the event that the primary Flume source becomes unavailable. This feature is currently only supported in Flume 1.6.0 (and assumed higher) using the Avro Flume source. This feature is enabled solely with Flume source configuration in the Flume Agent configuration file. The following is sample configuration for enabling fail over functionality:

```
client.type=default_failover
hosts=h1 h2 h3
hosts.h1=host_ip1:host_port1
hosts.h2=host_ip2:host_port2
hosts.h3=host_ip3:host_port3
max-attempts = 3
batch-size = 100
connect-timeout = 20000
request-timeout = 20000
```

4.13.4 Load Balancing Functionality

It is possible to configure the Oracle GoldenGate for Big Data Flume Handler so that produced Flume events will be load balanced across multiple Flume sources. This feature is currently only supported in Flume 1.6.0 (and assumed higher) using the Avro Flume source. This feature is enabled solely with Flume source configuration in the Flume Agent configuration file. The following is sample configuration for enabling load balancing functionality:

```
client.type = default_loadbalance
hosts = h1 h2 h3
hosts.h1 = host_ip1:host_port1
hosts.h2 = host_ip2:host_port2
hosts.h3 = host_ip3:host_port3
backoff = false
maxBackoff = 0
host-selector = round_robin
batch-size = 100
connect-timeout = 20000
request-timeout = 20000
```

5

Using the Kafka Handler

The Oracle GoldenGate for Big Data Kafka Handler is designed to stream change capture data from a Oracle GoldenGate trail to a Kafka topic. Additionally, the Kafka Handler provides optional functionality to publish the associated schemas for messages to a separate schema topic. Schema publication is currently only supported for Avro schemas because of the direct dependency of Avro messages upon an Avro schema.

Apache Kafka is an open source, distributed, partitioned and replicated messaging service. Kafka and its associated documentation are available at www.kafka.apache.org.

Kafka can be run as a single instance or as a cluster on multiple servers. Each Kafka server instance is called a broker. A Kafka topic is a category or feed name to which messages are published by the producers and retrieved by consumers.

The Kafka Handler implements a Kafka producer that writes serialized change capture data from multiple tables to one topic.

This chapter contains the following sections:

5.1 Setup and Running

Instructions for setting up each of the Kafka Handler components and running the handler are described in the following sections.

5.1.1 Runtime Prerequisites

- Zookeeper, a prerequisite component for Kafka and Kafka broker (or brokers) should be up and running.

- It is highly recommended and considered a best practice that the data topic and the schema topic (if applicable) are preconfigured on the running Kafka brokers. It is possible to create Kafka topics dynamically; however, this relies on the Kafka brokers being configured to allow dynamic topics.
- If the Kafka broker is not collocated with the Oracle GoldenGate for Big Data Kafka Handler process, the remote host:port must be reachable from the machine running the Kafka Handler.

5.1.2 Classpath Configuration

Two things must be configured in the gg.classpath configuration variable in order for the Kafka Handler to connect to Kafka and run. The required items are the Kafka Producer Properties file and the Kafka client jars. The Kafka client jars must match the version of Kafka that the Kafka Handler is connecting to. For a listing of the required client JAR files by version, see Kafka Handler Client Dependencies.

The recommending storage location for the Kafka Producer Properties file is the Oracle GoldenGate `dirprm` directory.

The default location of the Kafka client jars is `*Kafka_Home*/libs/*.`

The `gg.classpath` must be configured exactly as shown. Pathing to the Kafka Producer Properties file should simply contain the path with no wild card appended. The inclusion of the * wildcard in the path to the Kafka Producer Properties file will cause it not to be picked up. Conversely, pathing to the dependency jars should include the * wild card character in order to include all of the jar files in that directory in the associated classpath. Do not use `*.jar.` The following is an example of the correctly configured classpath:

```
gg.classpath=dirprm:/ggwork/kafka/lib/*
```

5.1.3 Pluggable Formatters

The Kafka Handler supports all the big data formatters which includes:

- Avro Row
- Avro Operation
- JSON
- XML
- Delimited Text

5.1.4 Kafka Handler Configuration

The following are the configurable values for the Kafka Handler. These properties are located in the `Java Adapter` properties file and not in the Replicat properties file.

Table 5-1 Configuration Properties for 12.2.0.1 Kafka Handler

Property Name	Property Value	Mandatory	Description
`gg.handlerlist`	*`kafkahandler`* (choice of any name)	Yes	List of handlers to be used.

Table 5-1 (Cont.) Configuration Properties for 12.2.0.1 Kafka Handler

Property Name	Property Value	Mandatory	Description
`gg.handler.`*`kafkahandler`*`.Type`	`kafka`	Yes	Type of handler to use. For example, Kafka, Flume, HDFS.
`gg.handler.`*`kafkahandler`*`.KafkaProducerConfigFile`	Any custom file name	No. Defaults to `kafka-producer-default.properties`	Filename in classpath that holds Apache Kafka properties to configure the Apache Kafka producer.
`gg.handler.`*`kafkahandler`*`.TopicName`	TopicName	Yes	Name of the Kafka topic where payload records will be sent.
`gg.handler.`*`kafkahandler`*`.Format`	Formatter class or short code	No. Defaults to `delimitedtext`.	Formatter to use to format payload. Can be one of `xml`, `delimitedtext`, `json`, `avro_row`, `avro_op`
`gg.handler.`*`kafkahandler`*`.SchemaTopicName`	Name of the schema topic	Yes, when schema delivery is required.	Topic name where schema data will be delivered. If this property is not set, schema will not be propagated. Schemas will be propagated only for Avro formatters.
`gg.handler.`*`kafkahandler`*`.SchemaPrClassName`	Fully qualified class name of a custom class that implements Oracle GoldenGate for Big Data Kafka Handler's `CreateProducerRecord` Java Interface	No. Defaults to provided implementation class: `oracle.goldengate.handler.kafka`.Default `ProducerRecord`	Schema is also propagated as a `ProducerRecord`. The default key here is the fully qualified table name. If this needs to be changed for schema records, the custom implementation of the `CreateProducerRecord` interface needs to be created and this property needs to be set to point to the fully qualified name of the new class.
`gg.handler.`*`kafkahandler`*`.BlockingSend`	`true` | `false`	No. Defaults to `false`.	If this property is set to true, then delivery to Kafka is made to work in a completely synchronous model. The next payload will be sent only after the current payload has been written out to the intended topic and an acknowledgement has been received. In transaction mode, this provides exactly once semantics. If this property is set to false, then delivery to Kafka is made to work in an asynchronous model. Payloads are sent one after the other without waiting for acknowledgements. Kafka internal queues may buffer contents to increase throughput. Checkpoints are made only when acknowledgements are received from Kafka brokers using Java Callbacks.

Table 5-1 (Cont.) Configuration Properties for 12.2.0.1 Kafka Handler

Property Name	Property Value	Mandatory	Description
`gg.handler.`*`kafkahandler`*`.ProducerRecordClass`	Fully qualified class name of a custom class that implements Oracle GoldenGate for Big Data Kafka Handler's `CreateProducerRecord` Java Interface	No. Defaults to out-of-box provided implementation class:`oracle.goldengate.handler.kafka.DefaultProducerRecord`	The unit of data in Kafka - a `ProducerRecord` holds the key field with the value representing the payload. This key is used for partitioning a Kafka Producer record that holds change capture data. By default, the fully qualified table name is used to partition the records. In order to change this key or behavior, the `CreateProducerRecord` Kafka Handler Interface needs to be implemented and this property needs to be set to point to the fully qualified name of the custom `ProducerRecord` class.
`gg.handler.`*`kafkahandler`*`.Mode`	`tx/op`	No. Defaults to `tx`.	With Kafka Handler operation mode, each change capture data record (Insert, Update, Delete etc) payload will be represented as a Kafka Producer Record and will be flushed one at a time. With Kafka Handler in transaction mode, all operations within a source transaction will be represented by as a single Kafka Producer record. This combined byte payload will be flushed on a transaction Commit event.
`gg.handler.`*`kafkahandler.`*`topicPartitioning`	`none | table`	None	Controls whether data published into Kafka should be partitioned by table. Set to table, the data for different tables are written to different Kafka topics. Set to none, the data from different tables are interlaced in the same topic as configured in `topicName`property.

5.1.5 Sample Configuration

The properties files are described in the following sections.

5.1.5.1 Java Adapter Properties File

A sample configuration for the Kafka Handler from the Adapter properties file is:

```
gg.handlerlist = kafkahandler
gg.handler.kafkahandler.Type = kafka
gg.handler.kafkahandler.KafkaProducerConfigFile = custom_kafka_producer.properties
gg.handler.kafkahandler.TopicName = oggtopic
gg.handler.kafkahandler.Format = avro_op
gg.handler.kafkahandler.SchemaTopicName = oggSchemaTopic
gg.handler.kafkahandler.ProducerRecordClass = com.company.kafka.CustomProducerRecord
gg.handler.kafkahandler.SchemaPrClassName = com.company.kafkaProdRec.SchemaRecord
gg.handler.kafkahandler.Mode = tx
gg.handler.kafkahandler.BlockingSend = true
```

A sample Replicat configuration and a Java Adapter Properties file for a Kafka integration can be found at the following directory:

```
GoldenGate_install_directory/AdapterExamples/big-data/kafka
```

5.1.6 Kafka Producer Configuration File

The Kafka Handler must access a Kafka producer configuration file in order publish messages to Kafka. The file name of the Kafka producer configuration file is controlled by the following configuration in the Kafka Handler properties.

```
gg.handler.kafkahandler.KafkaProducerConfigFile=custom_kafka_producer.properties
```

The Kafka Handler will attempt to locate and load the Kafka producer configuration file using the Java classpath. Therefore the Java classpath must include the directory containing the Kafka Producer Configuration File.

The Kafka producer configuration file contains Kafka proprietary properties. The Kafka documentation provides configuration information for the 0.8.2.0 Kafka producer interface properties. The Kafka Handler used these properties to resolve the host and port of the Kafka brokers and properties in the Kafka producer configuration file control the behavior of the interaction between the Kafka producer client and the Kafka brokers.

A sample of configuration file for the Kafka producer is as follows:

```
bootstrap.servers=localhost:9092
acks = 1
compression.type = gzip
reconnect.backoff.ms = 1000

value.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
# 100KB per partition
batch.size = 102400
linger.ms = 10000
max.request.size = 5024000
send.buffer.bytes = 5024000
```

5.2 Detailed Functionality

This section details the modes of operation of the Kafka Handler.

5.2.1 Transaction versus Operation Mode

The Kafka Handler sends instances of the Kafka `ProducerRecord` class to the Kafka producer API which in turn publishes the `ProducerRecord` to a Kafka topic. The Kafka `ProducerRecord` effectively is the implementation of a Kafka message. The `ProducerRecord` has two components, a key and a value. Both the key and value are represented as byte arrays by the Kafka Handler. This section describes how the Kafka Handler publishes data.

Transaction Mode

Transaction mode is indicated by the following configuration of the Kafka Handler:

```
gg.handler.name.Mode=tx
```

In Transaction Mode the serialized data for every operation in a transaction from the source Oracle GoldenGate trail files is concatenated. The contents of the concatenated operation data is the value of the Kafka `ProducerRecord` object. The key of the Kafka `ProducerRecord` object is NULL. The result is that Kafka messages comprise the data from 1 to N operations, where `N` is the number of operations in the transaction. In the case of grouped transactions, all of the data for all of the operations for a grouped transaction are concatenated into a single Kafka message. The result can be very large Kafka messages containing a data for a large number of operations.

Operation Mode

Operation mode is indicated by the following configuration of the Kafka Handler:

```
gg.handler.name.Mode=op
```

In Operation Mode the serialized data for each operation is placed into an individual `ProducerRecord` object as the value. The `ProducerRecord` key is the fully qualified table name of the source operation. The `ProducerRecord` is immediately sent using the Kafka Producer API. This means there is a 1 to 1 relationship between the incoming operations and the number of Kafka messages produced.

5.2.2 Blocking versus Blocking Mode

The Kafka Handler can send messages to Kafka in either Blocking mode (synchronous) or Non-Blocking Mode (asynchronous).

Blocking Mode

Blocking mode is set by the following configuration of the Kafka Handler:

```
gg.handler.name.BlockingSend=true
```

In this mode messages are delivered to Kafka on a synchronous basis. The Kafka Handler will not send the next message until the current message has been written to the intended topic and an acknowledgement has been received. Blocking mode provides the best guarantee of message delivery; however, its cost is reduced performance.

You must never set the Kafka Producer linger.ms variable when in blocking mode as this will cause the Kafka producer to wait for the entire timeout period before sending the message to the Kafka broker. In this scenario the Kafka Handler is waiting for acknowledgement that the message has been sent while at the same time the Kafka Producer is buffering messages to be sent to the Kafka brokers. Therefore, these settings are at odds with each other.

Non-Blocking Mode

Non-Blocking mode is set by the following configuration of the Kafka Handler:

```
gg.handler.name.BlockingSend=false
```

In this mode message are delivered to Kafka on an asynchronous basis. Kafka messages are published one after the other without waiting for acknowledgements. The Kafka Producer client may buffer incoming messages in order to increase throughput.

On each transaction commit, we invoke a blocking call on the Kafka producer to flush all operations that Kafka producer client that may have buffered internally. This allows the Kafka Handler to safely checkpoint ensuring zero data loss. Each transaction commit call will block for a maximum of `linger.ms` duration in the worst case. It is recommended to use small `linger.ms` times in the order of millisecond intervals.

You can control when the Kafka Producer flushes data to the Kafka Broker by a number of configurable properties in the Kafka producer configuration file. In order to enable batch sending of messages by the Kafka Producer both the `batch.size` and `linger.ms` Kafka Producer properties must be set in the Kafka producer configuration file. The `batch.size` controls the maximum number of bytes to buffer before a send to Kafka while the `linger.ms` variable controls the maximum milliseconds to wait before sending data. Data will be sent to Kafka once the `batch.size` is reached or the `linger.ms` period expires, whichever comes first. Setting of the `batch.size` only variable will cause messages to be sent immediately to Kafka.

5.2.3 Publishing to Multiple Topics

The Kafka Handler allows operation data from the source trail file to be published to separate topics based on the corresponding table name of the operation data. This feature allows sorting of operation data from the source trail file by the source table name. The feature is enabled by setting the following configuration in the Java Adapter properties file as follows:

```
gg.handler.kafka.topicPartitioning=table
gg.handler.kafka.mode=op
```

The mode *must* be set to `op` and the Kafka topic name used is the fully qualified table name of the source operation.

You can publish to multiple topics using the Kafka Handler. For example, you could publish one topic per table by setting `gg.handler.`*`kafkahandler.`*`topicPartitioning` property to `table`.

The topics are automatically created and with the topic name equal to the fully-qualified table name.

Kafka Broker Settings

To enable the automatic creation of topics, set the `auto.create.topics.enable property` to `true` in the Kafka Broker Configuration. The default value for this property is `true`.

If the `auto.create.topics.enable` property is set to `false` in Kafka Broker configuration, then all the required topics should be created manually before starting the Replicat process.

Schema Propagation

The schema data for all tables is delivered to the schema topic configured with the `schemaTopicName` property. For more information , see Schema Propagation

NOTE: Multiple topics are supported in the *op* mode only. For example, when `gg.handler.`*`kafkahandler.`*`topicPartitioning` is set to `table` then `gg.handler.kafkahandler.mode` should be set to `op`.

5.3 Schema Propagation

The Kafka Handler provides the ability to publish schemas to a schema topic. Currently the Avro Row and Operation formatters are the only formatters which are enabled for schema publishing. If the Kafka Handler `schemaTopicName` property is set the schema will be published for the following events.

- The Avro schema for a specific table will be published the first time an operation for that table is encountered.

- If the Kafka Handler receives a metadata change event, the schema will be flushed. The regenerated Avro schema for a specific table will be published the next time an operation for that table is encountered.
- If the Avro wrapping functionality is enabled, the generic wrapper Avro schema will be published the first time any operation is encountered. The generic wrapper Avro schema functionality can be enabled in the Avro formatter configuration. Refer to the Avro Row Formatter or Avro Operation Formatter sections of this document for exact instructions.

The Kafka `ProducerRecord` value will be the schema and the key will be the fully qualified table name.

Avro over Kafka can be problematic because of the direct dependency of Avro messages on an Avro schema. Avro messages are binary and therefore are not human readable. In order to deserialize an Avro message the receiver must first have the correct Avro schema. Since each table from the source database results in a separate Avro schema this can be problematic. The receiver of a Kafka message has no way to determine which Avro schema to use to deserialize individual messages when the source Oracle GoldenGate trail file includes operations from multiple tables. In order to solve this problem, the functionality was provided to wrap the specialized Avro messages in a generic Avro message wrapper. This generic Avro wrapper provides the fully qualified table name, the hashcode of the schema string, and the wrapped Avro message. The receiver can use the fully qualified table name and the hashcode of the schema string to resolve the associated schema of the wrapped message and then use that schema to deserialize the wrapped message.

5.4 Troubleshooting

This section details troubleshooting options.

5.4.1 Verify Kafka Setup

Command line Kafka producer can be used to write dummy data to a Kafka topic and a Kafka consumer can be used to read this data from the Kafka topic. This can be used to verify the set up and read write permissions to Kafka topics on disk. For further details, refer to the online Kafka documentation at

`http://kafka.apache.org/documentation.html#quickstart`

5.4.2 Classpath Issues

One of the most common problems is Java classpath problems. This problem typically manifests itself as a `ClassNotFoundException` in the `log4j` log file, but also may manifest itself as an error resolving the classpath if there is a typo in the `gg.classpath` variable. The Kafka client libraries do not ship with the Oracle GoldenGate for Big Data product. The requirement is on you to obtain the correct version of the Kafka client libraries and to properly configure the `gg.classpath` property in the Java Adapter Properties file to correctly resolve the Java the Kafka client libraries.

5.4.3 Invalid Kafka Version

The Oracle GoldenGate for Big Data Kafka Handler utilizes the new recommended Kafka producer API introduced in Kafka 0.8.2. Attempting to connect to a version of Kafka older than 0.8.2 will cause a runtime failure. There is no workaround for this issue. Customers must integrate with Kafka 0.8.2 or higher.

5.4.4 Kafka Producer Properties File Not Found

This problem typically manifests itself in the following exception.

```
ERROR 2015-11-11 11:49:08,482 [main] Error loading the kafka producer properties
```

The `gg.handler.kafkahandler.KafkaProducerConfigFile` configuration variable should be verified that the Kafka Producer Configuration file name is correctly set. The `gg.classpath` variable should also be checked to verify that the classpath includes the path to the Kafka Producer Properties file and that the path to the properties file does not contain a * wildcard at the end.

5.4.5 Kafka Connection Problem

This problem occurs if the Kafka Handler is unable to connect to Kafka. This problem manifests itself with the following warnings:

```
WARN 2015-11-11 11:25:50,784 [kafka-producer-network-thread | producer-1] WARN
(Selector.java:276) - Error in I/O with localhost/127.0.0.1
java.net.ConnectException: Connection refused
```

Ultimately the connection retry interval will expire and the Kafka Handler process will abend. Check that the Kafka Brokers is running and that the host and port provided in the Kafka Producer Properties file is correct. Network shell commands (such as `netstat -l`) can be used on the machine hosting the Kafka broker to verify that Kafka is listening on the expected port.

5.5 Performance Considerations

It is advised not to use linger.ms setting in the Kafka producer config file when `gg.handler.{name}.BlockingSend=true`. This will cause each send to block for at least linger.ms leading to major performance issues. The problem is that the Kafka Handler configuration and the Kafka Producer configuration are in conflict with each other. This configuration results a temporary deadlock scenario where the Kafka Handler is waiting for send acknowledgement while the Kafka producer is waiting for more messages before sending. The deadlock will resolve once the linger.ms period has expired. This scenario will repeat for every message sent.

For the best performance it is recommended to set the Kafka handler to operate in transaction mode using non-blocking (asynchronous) calls to the Kafka producer. This is achieved by the following configuration in the Java Adapter file.

```
gg.handler.{name}.Mode = tx
gg.handler.{name}.BlockingSend = false
```

Additionally the recommendation is to set the batch.size and linger.ms values in the Kafka Producer properties file. The values to set the batch.size and linger.ms values are highly dependent upon the use case scenario. Generally higher values will result in better throughput but latency is increased. Smaller values in these parameters will reduce latency but overall throughput will decrease. If the use case is for high volume of input data from the source trial files, then you is advised to set the `batch.size` and `linger.ms` size to as high as is tolerable.

Use of the `Replicat` variable `GROUPTRANSOPS` will also improve performance. The recommended setting for that is `10000`.

If it is a requirement of the customer that the serialized operations from the source trail file be delivered in individual Kafka messages, the then Kafka handler must be set to operation mode.

```
gg.handler.{name}.Mode = op
```

The result will be many more Kafka messages and performance will be adversely affected.

5.6 Security

Kafka 0.8.2.2 and earlier does not provide support for security. Kafka 0.9.0.0 introduced security through SSL/TLS or Kerberos. The Oracle GoldenGate Kafka Handler can be secured using SSL/TLS or Kerberos. The Kafka producer client libraries provide an abstraction of security functionality from the integrations utilizing those libraries. The Oracle GoldenGate Kafka Handler is effectively abstracted from security functionality. Enabling security requires setting up security for the Kafka cluster, connecting machines, and then configuring the Kafka producer properties file (that the Oracle GoldenGate Kafka Handler uses for processing) with the required security properties. For detailed instructions about securing the Kafka cluster, see the Kafka documentation at

http://kafka.apache.org/documentation.html#security_configclients

5.7 Kafka Handler Certification Matrix

The Oracle GoldenGate for Big Data Kafka Handler implements the new recommended Kafka producer interface introduced in Kafka 0.8.2.0. The Kafka Handler is *not* compatible with Kafka version 0.8.1.0 and older.

The Kafka Handler is compatible with the following versions of Apache Kafka

- 0.9.0.*x*
- 0.8.2.*x*

The Kafka Handler is compatible with the following HDP 2.3 (Kafka 0.8.2.0) versions of the Hortonworks Data Platform (HDP):

- HDP 2.4 (Kafka 0.9.0)
- HDP 2.3 (Kafka 0.8.2.0)

Cloudera (CDH) does not currently include Kafka. Cloudera currently distributes Kafka separately as Cloudera Distribution of Apache Kafka. The Kafka Handler is compatible with the following CDH distributions:

- Cloudera Distribution of Apache Kafka 2.0.*x* (Kafka 0.9.0.0)
- Cloudera Distribution of Apache Kafka 1.*x* (Kafka 0.8.2.0)

5.8 Metadata Change Events

Metadata change events are now handled in the Kafka Handler. However, this is only relevant if you has configured a schema topic and the formatter used supports schema propagation (currently Avro row and Avro Operation formatters). The next time an operation is encountered for a table for which the schema has changed, the updated schema will be published to the schema topic.

To support metadata change events the Oracle GoldenGate process capturing changes in the source database must support both DDL changes and metadata in trail. GoldenGate does not support DDL replication for all database implementations. You are advised to consult the Oracle GoldenGate documentation for their database implementation to understand if DDL replication is supported.

5.9 Snappy Considerations

The Kafka Producer Configuration file supports the use of compression. One of the configurable options is Snappy. Snappy is an open source compression and decompression (codec) library that tends to provide performance than other codec libraries. However, Snappy has a shortcoming in that the Snappy jar does not run on all platforms. Snappy seems to universally work on Linux systems but it can be hit and miss on other Unix and Windows implementations. Customers using snappy compression are advised to test Snappy on all required systems before implementing compression using Snappy. If Snappy does not port to all required systems then Oracle suggests using an alternate codec library.

6

Using the Pluggable Formatters

Formatters provide the functionality to convert operations from the Oracle GoldenGate trail file info formatted messages that can then be sent to Big Data targets by one of the Oracle GoldenGate for Big Data Handlers. The Oracle GoldenGate for Big Data release ships with the following five pluggable formatters:

- Delimited Text
- JSON
- XML
- Avro Row
- Avro Operation

This chapter contains the following sections:

6.1 Operation versus Row Based Formatting

The Oracle GoldenGate for Big Data formatters are of two categories, operation based formatters and row based formatters. Operations represent the individual insert, update, and delete events that occur on table data in the source database. Insert operations only provide after change data (or images) since a new row is being added to the source database. Update operations provide both before and after change data which shows how existing row data is modified. Delete operations only provide before change data to provide identification of the row being deleted. The operation based formatters model the operation as it is exists in the source trail file. Operation based formats include fields for the before and after images. The row based formatters model the row data as it exists after the operation data is applied. Row based formatters only contain a single image of the data. The following represents what data is displayed for both the operation and row based formatters.

6.1.1 Operation Formatters

The formatters that support operation based formatting are JSON, Avro Operation, and XML. The output of operation based formatters are as follows:

- Insert Operation - Before image data is NULL. After image data is output.
- Update Operation - Both before and after image data is output.
- Delete Operation - Before image data is output. After image data is NULL.
- Truncate Operation - Both before and after image data is NULL.

6.1.2 Row Formatters

The formatters that support row based formatting area Delimited Text and Avro Row. Row based formatters output the following information for the following operations.

- Insert Operation - After image data only.
- Update Operation - After image data only. Primary key updates are a special case which will be discussed in individual sections for the specific formatters.
- Delete Operation - Before image data only.
- Truncate Operation - Table name is provided but both before and after image data are NULL. Truncate table is a DDL operation and it may not support different database implementations. Refer to the Oracle GoldenGate documentation for your database implementation.
- Table Row or Column Value States
- In an RDBMS, table data for a specific row and column can only have one of two states. Either the table row/column value has a value or the row/column value is NULL. However when data is transferred to the Oracle GoldenGate trail file by the Oracle GoldenGate capture process, this can expand to three possible states: the table row/column has a value, the row/column value is NULL, or the row/column value is missing.
- For an insert operation it is reasonable to expect that the after image contains data for all column values whether that column has a value or is NULL. However, the data included for update and delete operations may not always contain complete data for all columns. When replicating data to an RDBMS for an update operation the only data that is required to modify the data in the target database are the primary key values and the values of the columns that changed. Additionally, for a delete operation it is only necessary to have the primary key values to delete the row from the target database. Therefore, even though table row/column values have a value in the source database, the values may be missing in the source trail file. Because it is possible for row/column data in the source trail file to have three states, the Big Data Formatters must also be able to represent data in the three states.
- What row/column data is available in the Oracle GoldenGate trail file will have an impact on Big Data integrations. It is important for you to understand what data is required. You typically has control on what data is included for operations in the Oracle GoldenGate trail file. For Oracle Databases this is controlled by the supplemental logging level. Refer to the Oracle GoldenGate documentation for your specific source database implementation to understand how to control the row/column values that are included in the Oracle GoldenGate trail file.

6.2 Delimited Text Formatter

The Delimited Text Formatter is a row based formatter. It formats database operations from the source trail file into a delimited text output. Each insert, update, delete, or truncate operation from the source trail will be formatted into an individual delimited message. Delimited text output will be a fixed number of fields for each table separated by a field delimiter and terminated by a line delimiter. The fields are positionally relevant. Many Big Data analytical tools including Hive work well with HDFS files containing delimited text.

Column values for an operation from the source trail file can have one of three states: column has a value, column value is NULL, or column value is missing. By default the delimited text maps these column value states into the delimited text output as follows:

- Column has a value - The column value is output.
- Column value is NULL - The default output value is `NULL`. The output for the case of a NULL column value is configurable.
- Column value is missing - The default output value is "". The output for the case of a missing column value is configurable.

6.2.1 Message Formatting Details

The default format for output of data is the following:

First is the row metadata:

```
<OPERATION TYPE><FIELD DELIMITER><FULLY QUALIFIED TABLE NAME><FIELD
DELIMITER><OPERATION TIMESTAMP><FIELD DELIMITER><CURRENT TIMESTAMP><FIELD
DELIMITER><TRAIL POSITION><FIELD DELIMITER><TOKENS><FIELD DELIMITER>
```

Next is the row data:

```
<COLUMN 1 VALUE><FIELD DELIMITER><COLUMN N VALUE><LINE DELIMITER>
```

Optionally, the column name can be included before each column value that changes the output format for the row data:

```
<COLUMN 1 NAME><FIELD DELIMITER><COLUMN 1 VALUE><COLUMN N
NAME><FIELD DELIMITER><COLUMN N VALUE><LINE DELIMITER>
```

Operation Type - Operation type is the indicator of the type of database operation from the source trail file. Default values are `I` for insert, `U` for update, `D` for delete, `T` for truncate. Output of this field is suppressible.

Fully Qualified Table name - The fully qualified table name is the source database table include including the catalog name and schema name. The format of the fully qualified table name is *`CATALOG NAME.SCHEMA NAME.TABLE NAME`*. Output of this field is suppressible.

Operation Timestamp - The operation timestamp is the commit record timestamp from the source system. Therefore all operations in a transaction (unbatched transaction) should have the same operation timestamp. This timestamp is fixed, and the operation timestamp will be the same if the trail file is replayed. Output of this field is suppressible.

Current Timestamp - The current timestamp is a timestamp of the current time when delimited text formatter processes the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file will *not* result in the same timestamp for the same operation. Output of this field is suppressible.

Trail Position - This is the concatenated sequence number and RBA number from the source trail file. The trail position provides traceability of the operation back to the source trail file. The sequence number is the source trail file number. The RBA number is the offset in the trail file. Output of this field is suppressible.

Tokens - The tokens are the token key value pairs from the source trail file. The output of this field in the delimited text output is suppressed if the `includeTokens` configuration property on the corresponding handler is *not* explicitly set to `true`.

6.2.2 Sample Formatted Messages

The following sections contain sample messages from the Delimited Text Formatter. The default field delimiter has been changed to a pipe ("|") to more clearly display the message.

6.2.2.1 Sample Insert Message

```
I|GG.TCUSTORD|2013-06-02
22:14:36.000000|2015-09-18T13:23:01.612001|00000000000000001444|R=AADPkvAAEAAEqL2A
AA|WILL|1994-09-30:15:33:00|CAR|144|17520.00|3|100
```

6.2.2.2 Sample Update Message

```
U|GG.TCUSTORD|2013-06-02
22:14:41.000000|2015-09-18T13:23:01.987000|00000000000000002891|R=AADPkvAAEAAEqLzA
AA|BILL|1995-12-31:15:00:00|CAR|765|14000.00|3|100
```

6.2.2.3 Sample Delete Message

```
D|GG.TCUSTORD|2013-06-02
22:14:41.000000|2015-09-18T13:23:02.000000|00000000000000004338|L=206080450,6=9.0.
80330,R=AADPkvAAEAAEqLzAAC|DAVE|1993-11-03:07:51:35|PLANE|600|||
```

6.2.2.4 Sample Truncate Message

```
T|GG.TCUSTORD|2013-06-02
22:14:41.000000|2015-09-18T13:23:02.001000|00000000000000004515|R=AADPkvAAEAAEqL2A
AB|||||||
```

6.2.3 Common Pitfalls

Care should be exercised when choosing field and line delimiters. It is important to choose delimiter values that will not occur in content of the data.

The Java Adapter configuration functionality will trim out leading or trailing characters that are determined to be whitespace. Wrap the configuration value in a `CDATA[]` wrapper to preserve the whitespace when configuration values contain leading or trailing characters that are considered whitespace. For example a configuration value of `\n` should be configured as `CDATA[\n]`.

You can search column values using regular expressions then replace matches with a specified value. This search and replace functionality can be utilized in conjunction with the Delimited Text Formatter to ensure that there are no collisions between column value contents and field and line delimiters. For more information, see Using Regular Expression Search and Replace.

6.2.4 Logging of Output Format Summary

The Java `log4j` logging will log a summary of the delimited text output format if `INFO` level logging is enabled. A summary of the delimited fields will be logged for each source table encountered and occurs when the first operation for that table is received by the Delimited Text formatter. You may find this detailed explanation of the fields of the delimited text output useful when performing an initial setup. In the case of the metadata change event, the summary of the delimited fields will be regenerated and logged again at the first operation for that table after the metadata change event.

6.2.5 Delimited Text Format Configuration

Table 6-1 Configuration Options

Parameters	Optional / Required	Legal Values	Default	Explanation

`gg.handler.`*`name`*`.format.includeColumnNames`	Optional	`true | false`	`false`	Controls the output of writing the column names as a delimited field preceding the column value. If true output is like:
				`COL1_Name|COL1_Value|Col2_Name|Col2_Value`
				If the false output is like:
				`COL1_Value|Col2_Value`

`gg.handler.`*`name`*`.format.includeOpTimestamp`	Optional	`true | false`	`true`	A `false` value suppresses the output of the operation timestamp from the source trail file in the output.
`gg.handler.`*`name`*`.format.includeCurrentTimestamp`	Optional	`true | false`	`true`	A `false` value suppresses the output of the current timestamp in the output.
`gg.handler.`*`name`*`.format.includeOpType`	Optional	`true | false`	`true`	A `false` value suppresses the output of the operation type in the output.
`gg.handler.`*`name`*`.format.insertOpKey`	Optional	Any string	`I`	Indicator to be inserted into the output record to indicate an insert operation.
`gg.handler.`*`name`*`.format.updateOpKey`	Optional	Any string	`U`	Indicator to be inserted into the output record to indicate an update operation.
`gg.handler.`*`name`*`.format.deleteOpKey`	Optional	Any string	`D`	Indicator to be inserted into the output record to indicate a delete operation.

Table 6-1 (Cont.) Configuration Options

Parameters	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name.`*`format.truncateOpKey`	Optional	Any string	`T`	Indicator to be inserted into the output record to indicate a truncate operation.
`gg.handler.`*`name.`*`format.encoding`	Optional	Any encoding name or alias supported by Java.	The native system encoding of the machine hosting the Oracle Golden Gate process.	Determines the encoding of the output delimited text.
`gg.handler.`*`name.`*`format.fieldDelimiter`	Optional	Any String	`ASCII 001` (the default Hive delimiter)	The delimiter used between delimited fields. This value supports `CDATA[]` wrapping.
`gg.handler.`*`name.`*`format.lineDelimiter`	Optional	Any String	newline (the default Hive delimiter)	The delimiter used between records. This value supports `CDATA[]` wrapping.
`gg.handler.`*`name.`*`format.includeTableName`	Optional	`true | false`	`true`	Use `false` to suppress the output of the table name in the output delimited data.
`gg.handler.`*`name.`*`format.keyValueDelimiter`	Optional	Any string	`=`	Provides a delimiter between keys and values in a map. Key1=value1. Tokens are mapped values. Configuration value supports `CDATA[]` wrapping.
`gg.handler.`*`name.`*`format.keyValuePairDelimiter`	Optional	Any string	`,`	Provides a delimiter between key value pairs in a map. `Key1=Value1`,`Key2=Value2`. Tokens are mapped values. Configuration value supports `CDATA[]` wrapping.

Table 6-1 (Cont.) Configuration Options

Parameters	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name.`*`format.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Provides configuration for how the formatter should handle update operations that change a primary key. Primary key operations can be problematic for the text formatter and require special consideration by you. • `abend` - indicates the process will abend • `update` - indicates the process will treat this as a normal update • `delete-insert` - indicates the process will treat this as a delete and an insert. Full supplemental logging needs to be enabled for this to work. Without full before and after row images the insert data will be incomplete.
`gg.handler.`*`name.`*`format.nullValueRepresentation`	Optional	Any string	NULL	Allows you to configure what will be included in the delimited output in the case of a NULL value. Configuration value supports `CDATA[]` wrapping.
`gg.handler.`*`name.`*`format.missingValueRepresentation`	Optional	Any string	`""` (no value)	Allows you to configure what will be included in the delimited text output in the case of a missing value. Configuration value supports `CDATA[]` wrapping.
`gg.handler.`*`name.`*`format.includePosition`	Optional	`true` | `false`	`true`	Allows you to suppress the output of the operation position from the source trail file.
`gg.handler.`*`name.`*`format.includePosition`	Optional	`true` | `false`	true	Allows you to suppress the output of the operation position from the source trail file.
`gg.handler.`*`name.`*`format.iso8601Forma`	Optional	`true` | `false`	`true`	Controls the format of the current timestamp. The default is the ISO 8601 format. Set to `false` removes the `“T”` between the date and time in the current timestamp, which outputs `“ “` instead.

6.2.6 Sample Configuration

The following is the sample configuration for the Delimited Text formatter from the Java Adapter configuration file:

```
gg.handler.hdfs.format.includeColumnNames=false
gg.handler.hdfs.format.includeOpTimestamp=true
gg.handler.hdfs.format.includeCurrentTimestamp=true
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.encoding=UTF-8
gg.handler.hdfs.format.fieldDelimiter=CDATA[\u0001]
gg.handler.hdfs.format.lineDelimiter=CDATA[\n]
gg.handler.hdfs.format.includeTableName=true
gg.handler.hdfs.format.keyValueDelimiter=CDATA[=]
gg.handler.hdfs.format.kevValuePairDelimiter=CDATA[,]
gg.handler.hdfs.format.pkUpdateHandling=abend
gg.handler.hdfs.format.nullValueRepresentation=NULL
gg.handler.hdfs.format.missingValueRepresentation=CDATA[]
gg.handler.hdfs.format.includePosition=true
gg.handler.hdfs.format=delimitedtext
```

6.2.7 Metadata Change Events

Oracle GoldenGate for Big Data now handles metadata change events at runtime. This assumes the replicated database and upstream replication processes are propagating metadata change events.The Delimited Text Formatter will simply change the output format to accommodate the change and continue running.

However, it is important to understand that a metadata change may impact downstream applications. Delimited text formats are comprised of a fixed number of fields that are positionally relevant. Deleting a column in the source table can be handled seamlessly during Oracle GoldenGate runtime, but will result in a change in the total number of fields and potentially the positional relevance of some fields. Adding an additional column or columns is probably the least impactful metadata change event assuming the new column is added to the end. You should consider the impact of a metadata change event before executing the event. In a scenario where metadata change events will be frequent, it is recommended that you consider a more flexible and self describing format, such as JSON or XML.

6.2.8 Special Considerations

Big Data applications differ from RDBMSs in how data is stored. Update and delete operations in an RDBMS result in a change to the existing data. On the contrary, data is not changed in Big Data applications but simply appended to existing data. Therefore, the current state of a given row becomes a consolidation of all of the existing operations for that row in the HDFS system. This leads to some special scenarios.

6.2.8.1 Primary Key Updates

Primary key update operations require special consideration and planning for Big Data integrations. Primary key updates are update operations that modify one or more of the primary keys for the given row from the source database. Since data is simply appended in Big Data applications a primary key update operation looks more like a new insert than an update without any special handling. The Delimited Text formatter provides specialized handling for primary keys that is configurable to you. These are the configurable behaviors:

Table 6-2 Configurable Behavior

Value	Description
`abend`	The default behavior is that the delimited text formatter will abend in the case of a primary key update.

Table 6-2 (Cont.) Configurable Behavior

Value	Description
`update`	With this configuration the primary key update will be treated just like any other update operation. This configuration alternative should only be selected if you can guarantee that the primary key that is being changed is not being used as the selection criteria when selecting row data from a Big Data system.
`delete-insert`	Using this configuration the primary key update is treated as a special case of a delete using the before image data and an insert using the after image data. This configuration may more accurately model the effect of a primary key update in a Big Data application. However, if this configuration is selected it is important to have full supplemental logging enabled on replication at the source database. Without full supplemental logging, the delete operation will be correct, but the insert operation will not contain all of the data for all of the columns for a full representation of the row data in the Big Data application.

6.2.8.2 Data Consolidation

As previously stated, Big Data applications simply append data to the underlying storage. Analytic tools generally spawn map reduce programs that traverse the data files and consolidate all the operations for a given row into a single output. Therefore, it is important to have an indicator of the order of operations. The Delimited Text formatter provides a number of metadata fields to fulfill this need. The operation timestamp may be sufficient to fulfill this requirement. However, two update operations may have the same operation timestamp especially if they share a common transaction. The trail position can provide a tie breaking field on the operation timestamp. Lastly, the current timestamp may provide the best indicator of order of operations in Big Data.

6.3 JSON Formatter

The JSON Formatter is an operation based formatter. It formats operation data from the source trail file into a JSON object. Each individual insert, update, delete and truncate operation will be formatted into an individual JSON message.

6.3.1 Message Formatting Details

The following two subsections detail the contents of generated JSON messages. The first section details the operation metadata and the second section details the before and after image column data values.

6.3.1.1 Operation Metadata

JSON objects generated by the JSON Formatter contain the following metadata fields at the beginning of each message:

Table 6-3 JSON metadata

Table 6-3 (Cont.) JSON metadata

Value	Description
`table`	Contains fully qualified table name. The format of the fully qualified table name is: *`CATALOG NAME.SCHEMA NAME.TABLE NAME`*
`op_type`	Contains the operation type that is the indicator of the type of database operation from the source trail file. Default values are `I` for insert, `U` for update, `D` for delete, `T` for truncate.
`op_ts`	The operation timestamp is the timestamp of the operation from the source trail file. Since this timestamp is from the source trail it is fixed. Replaying the trail file will result in the same timestamp for the same operation.
`current_ts`	The current timestamp is a timestamp of the current time when delimited text formatter processes the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file will *not* result in the same timestamp for the same operation.
`pos`	This is the trail file position with is the concatenated sequence number and RBA number from the source trail file. The trail position provides traceability of the operation back to the source trail file. The sequence number is the source trail file number. The RBA number is the offset in the trail file.
`primary_keys`	An array variable holding the column names of the primary keys of the source table. The `primary_keys` field is only include in the JSON output if the `includePrimaryKeys` configuration property is set to `true`.
`tokens`	This member is an object whose members are the token key value pairs from the source trail file.

6.3.1.2 Operation Data

The data following the operation metadata is the operation data. This data is represented by `before` and `after` members that are objects. The objects contain members with the keys being the column names and the values being the column values.

Column values for an operation from the source trail file can have one of three states: column has a value, column value is NULL, or column value is missing. The JSON Formatter maps these column value states into the created JSON objects as follows:

- Column has a value - The column value is output. In the following example the member `STATE` has a value.

```
    "after":{       "CUST_CODE":"BILL",       "NAME":"BILL'S USED CARS",
"CITY":"DENVER",       "STATE":"CO"    }
```

- Column value is NULL - The default output value is a JSON NULL. In the following example the member `STATE` is NULL.

```
    "after":{       "CUST_CODE":"BILL",       "NAME":"BILL'S USED CARS",
"CITY":"DENVER",       "STATE":null    }
```

- Column value is missing - The JSON will contain no element for a missing column value. In the following example the member STATE is missing.

```
    "after":{       "CUST_CODE":"BILL",       "NAME":"BILL'S USED CARS",
"CITY":"DENVER",    }
```

The default setting of the JSON Formatter is to map the data types from the source trail file to the associated JSON data type. JSON supports few data types so this functionality largely results in the mapping of numeric fields from the source trail file to members typed as numbers. This data type mapping is configurable to alternatively treat all data as strings.

6.3.2 Sample JSON Messages

The following are sample JSON messages created by the JSON Formatter for insert, update, delete, and truncate operations.

6.3.2.1 Sample Insert Message

```
{
    "table":"GG.TCUSTORD",
    "op_type":"I",
    "op_ts":"2013-06-02 22:14:36.000000",
    "current_ts":"2015-09-18T13:39:35.447000",
    "pos":"00000000000000001444",
    "tokens":{
        "R":"AADPkvAAEAAEqL2AAA"
    },
    "after":{
        "CUST_CODE":"WILL",
        "ORDER_DATE":"1994-09-30:15:33:00",
        "PRODUCT_CODE":"CAR",
        "ORDER_ID":"144",
        "PRODUCT_PRICE":17520.00,
        "PRODUCT_AMOUNT":3,
        "TRANSACTION_ID":"100"
    }
}
```

6.3.2.2 Sample Update Message

```
{
    "table":"GG.TCUSTORD",
    "op_type":"U",
    "op_ts":"2013-06-02 22:14:41.000000",
    "current_ts":"2015-09-18T13:39:35.748000",
    "pos":"00000000000000002891",
    "tokens":{
        "R":"AADPkvAAEAAEqLzAAA"
    },
    "before":{
        "CUST_CODE":"BILL",
        "ORDER_DATE":"1995-12-31:15:00:00",
        "PRODUCT_CODE":"CAR",
        "ORDER_ID":"765",
        "PRODUCT_PRICE":15000.00,
        "PRODUCT_AMOUNT":3,
        "TRANSACTION_ID":"100"
    },
    "after":{
        "CUST_CODE":"BILL",
        "ORDER_DATE":"1995-12-31:15:00:00",
        "PRODUCT_CODE":"CAR",
        "ORDER_ID":"765",
        "PRODUCT_PRICE":14000.00,
        "PRODUCT_AMOUNT":3,
        "TRANSACTION_ID":"100"
    }
}
```

6.3.2.3 Sample Delete Message

```
{
    "table":"GG.TCUSTORD",
    "op_type":"D",
    "op_ts":"2013-06-02 22:14:41.000000",
    "current_ts":"2015-09-18T13:39:35.766000",
    "pos":"00000000000000004338",
    "tokens":{
        "L":"206080450",
        "6":"9.0.80330",
        "R":"AADPkvAAEAAEqLzAAC"
    },
    "before":{
        "CUST_CODE":"DAVE",
        "ORDER_DATE":"1993-11-03:07:51:35",
        "PRODUCT_CODE":"PLANE",
        "ORDER_ID":"600"
    }
}
```

6.3.2.4 Sample Truncate Message

```
{
    "table":"GG.TCUSTORD",
    "op_type":"T",
    "op_ts":"2013-06-02 22:14:41.000000",
    "current_ts":"2015-09-18T13:39:35.767000",
    "pos":"00000000000000004515",
    "tokens":{
        "R":"AADPkvAAEAAEqL2AAB"
    }
}
```

6.3.3 JSON Schemas

By default JSON schemas are generated for each source table encountered. JSON schemas are generated on a just in time basis when an operation for that table is first encountered. A JSON schema is not required to parse a JSON object. However, many JSON parsers can use a JSON schema to perform a validating parse of a JSON object. Alternatively, you can review the JSON schemas to understand the layout of output JSON objects. The JSON schemas are created in the `*GoldenGate_Home*/dirdef` directory by default and are named by the following convention: `*FULLY_QUALIFIED_TABLE_NAME*.schema.json`. The generation of the JSON schemas is suppressible. The following is an example of a JSON schema for the JSON object listed in the previous section.

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"GG.TCUSTORD",
    "description":"JSON schema for table GG.TCUSTORD",
    "definitions":{
        "row":{
            "type":"object",
            "properties":{
                "CUST_CODE":{
                    "type":[
                        "string",
                        "null"
                    ]
                },
                "ORDER_DATE":{
                    "type":[
                        "string",
                        "null"
                    ]
                },
                "PRODUCT_CODE":{
                    "type":[
                        "string",
                        "null"
                    ]
                },
                "ORDER_ID":{
                    "type":[
                        "string",
                        "null"
                    ]
                },
                "PRODUCT_PRICE":{
                    "type":[
                        "number",
                        "null"
                    ]
                },
                "PRODUCT_AMOUNT":{
                    "type":[
                        "number",
                        "null"
                    ]
                },
                "TRANSACTION_ID":{
                    "type":[
                        "string",
                        "null"
                    ]
                }
            },
            "additionalProperties":false
        },
        "tokens":{
            "type":"object",
            "description":"Token keys and values are free form key value pairs.",
            "properties":{
            },
            "additionalProperties":true
        }
    },
    "type":"object",
    "properties":{
        "table":{
            "description":"The fully qualified table name",
            "type":"string"
        },
        "op_type":{
            "description":"The operation type",
            "type":"string"
        },
        "op_ts":{
            "description":"The operation timestamp",
            "type":"string"
        },
        "current_ts":{
            "description":"The current processing timestamp",
            "type":"string"
        },
        "pos":{
            "description":"The position of the operation in the data source",
            "type":"string"
        },
        "tokens":{
            "$ref":"#/definitions/tokens"
        },
        "before":{
            "$ref":"#/definitions/row"
        },
        "after":{
            "$ref":"#/definitions/row"
        }
    },
    "required":[
        "table",
        "op_type",
        "op_ts",
        "current_ts",
        "pos"
    ],
    "additionalProperties":false
}
```

6.3.4 JSON Schema Configuration

Table 6-4 JSON Schema Configuration Parameters

Parameters	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.insertOpKey`	Optional	Any string	`I`	Indicator to be inserted into the output record to indicate an insert operation.
`gg.handler.`*`name`*`.format.updateOpKey`	Optional	Any string	`U`	Indicator to be inserted into the output record to indicate an update operation.
`gg.handler.`*`name`*`.format.deleteOpKey`	Optional	Any string	`D`	Indicator to be inserted into the output record to indicate a delete operation.
`gg.handler.`*`name`*`.format.truncateOpKey`	Optional	Any string	`T`	Indicator to be inserted into the output record to indicate a truncate operation.

Table 6-4 (Cont.) JSON Schema Configuration Parameters

Parameters	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.prettyPrintformat.prettyPrint`	Optional	`true` | `false`	`false`	Controls the output format of the JSON data. True is pretty print, formatted with white space to be more easily read by humans. False is not pretty print, more compact but very difficult for humans to read.
`gg.handler.`*`name`*`.format.jsonDelimiter`	Optional	Any string	`""` (no value)	Allows you to insert an optional delimiter between generated JSONs to allow them to be more easily parsed out of a continuous stream of data. Configuration value supports `CDATA[]` wrapping.
`gg.handler.`*`name`*`.format.generateSchema`	Optional	`true` | `false`	`true`	Controls the generation of JSON schemas for the generated JSON documents. JSON schemas are generated on a table by table basis. A JSON schema is not required to parse a JSON document. However, a JSON schema can provide you an indication of what the JSON documents will look like and can be used for a validating JSON parse.
`gg.handler.`*`name`*`.format.schemaDirectory`	Optional	Any legal, existing file system path	`./dirdef`	Controls the output location of generated JSON schemas.
`gg.handler.`*`name`*`.format.treatAllColumnsAsStrings`	Optional	`true` | `false`	`false`	Controls the output typing of generated JSON documents. If set to false then the formatter will attempt to map Oracle GoldenGate types to the corresponding JSON type. If set to true then all data will be treated as Strings in the generated JSONs and JSON schemas.
`gg.handler.`*`name`*`.format.encoding`	Optional	Any legal encoding name or alias supported by Java.	`UTF-8` (the JSON default)	Controls the output encoding of generated JSON schemas and documents.
`gg.handler.`*`name`*`.format.versionSchemas`	Optional	`true` | `false`	`false`	Controls the version of created schemas. Schema versioning causes a schema with a timestamp to be created in the schema directory on the local file system every time a new schema is created. True enables schema versioning. False disables schema versioning.

Table 6-4 (Cont.) JSON Schema Configuration Parameters

Parameters	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.iso8601Format`	Optional	`true` | `false`	`true`	Controls the format of the current timestamp. The default is the ISO 8601 format. Set to `false` removes the `"T"` between the date and time in the current timestamp, which outputs `" "` instead.
`gg.handler.`*`name`*`.format.includePrimaryKeys`	Optional	`true` | `false`	`false`	Set this configuration property to `true` to include an array of the primary key column names from the source table in the JSON output.

6.3.5 Sample Configuration

The following is sample configuration for the JSON Formatter from the Java Adapter configuration file:

```
gg.handler.hdfs.format=json
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.prettyPrint=false
gg.handler.hdfs.format.jsonDelimiter=CDATA[]
gg.handler.hdfs.format.generateSchema=true
gg.handler.hdfs.format.schemaDirectory=dirdef
gg.handler.hdfs.format.treatAllColumnsAsString
```

6.3.6 Metadata Change Events

Metadata change events are now handled at runtime. A metadata change event for a given table will result in the regeneration of the JSON schema the next time an operation for that table is encountered. The content of created JSON messages will be changed to reflect the metadata change. For example, if the metadata change is to add an additional column, the new column will be included in created JSON messages after the metadata change event.

6.3.7 Primary Key Updates

Since the JSON models the operation data primary key updates require no special treatment are treated just as any other update. The before and after values will reflect the change in the primary key.

6.4 Avro Row Formatter

Apache Avro is an open source data serialization/deserialization framework known for its flexibility, compactness of serialized data, and good serialization/ deserialization performance. Apache Avro is commonly used in Big Data applications.

The Avro Row Formatter formats operation data from the source trail file into messages in an Avro binary array format. Each individual insert, update, delete and truncate operation will be formatted into an individual Avro message. The source trail file will contain the before and after images of the operation data. The Avro Row Formatter takes that before and after image data and formats the data into an Avro binary representation of the operation data.

The Avro Row Formatter formats operations from the source trail file into a format that represents the row data. This format is more compact than the output from the Avro Operation Formatter for that the Avro messages model the change data operation.

The Avro Row Formatter may be a good choice when streaming Avro data to HDFS. Hive supports data files in HDFS in an Avro format.

6.4.1 Message Formatting Details

The following two subsections detail the contents of generated Avro row messages. The first section details the operation metadata and the second section details the column values data.

6.4.1.1 Operation Metadata

Avro messages generated by the Avro Row Formatter contain the following seven metadata fields that begin the message:

Table 6-5 Avro Formatter Metadata

Value	Description
`table`	The fully qualified table name. The format of the fully qualified table name is: *`CATALOG_NAME.SCHEMA_NAME.TABLE_NAME`*
`op_type`	The operation type that is the indicator of the type of database operation from the source trail file. Default values are `I` for insert, `U` for update, `D` for delete, `T` for truncate.
`op_ts`	The operation timestamp is the timestamp of the operation from the source trail file. Since this timestamp is from the source trail it is fixed. Replaying the trail file will result in the same timestamp for the same operation.
`current_ts`	The current timestamp is the current time when the formatter processed the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file will *NOT* result in the same timestamp for the same operation.
`pos`	The trail file position is the concatenated sequence number and rba number from the source trail file. The trail position provides traceability of the operation back to the source trail file. The sequence number is the source trail file number. The rba number is the offset in the trail file.
`primary_keys`	An array variable holding the column names of the primary keys of the source table.
`tokens`	A map variable holding the token key value pairs from the source trail file.

6.4.1.2 Operation Data

The data following the operation metadata is the operation data. This data is represented as individual fields identified by the column names.

Column values for an operation from the source trail file can have one of three states: column has a value, column value is NULL, or column value is missing. Avro attributes only support two states, column has a value or column value is NULL. Missing column values will be treated the same as NULL values. It is highly recommended that when using the Avro Row Formatter, you configures the Oracle GoldenGate capture process to provide full image data for all columns in the source trail file. Refer to the Oracle GoldenGate documentation for your specific RDBMS for instructions to enable this functionality.

The default setting of the Avro Row Formatter is to map the data types from the source trail file to the associated Avro data type. Avro supports few data types so this functionality largely results in the mapping of numeric fields from the source trail file to members typed as numbers. This data type mapping is configurable to alternatively treat all data as strings.

6.4.2 Sample Avro Row Messages

Avro messages are binary and therefore not human readable. For the sample messages the JSON representation of the messages are displayed here.

6.4.2.1 Sample Insert Message

```
{"table": "GG.TCUSTORD",
"op_type": "I",
"op_ts": "2013-06-02 22:14:36.000000",
"current_ts": "2015-09-18T10:13:11.172000",
"pos": "00000000000000001444",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"],
"tokens": {"R": "AADPkvAAEAAEqL2AAA"},
"CUST_CODE": "WILL",
"ORDER_DATE": "1994-09-30:15:33:00",
"PRODUCT_CODE": "CAR",
"ORDER_ID": "144",
"PRODUCT_PRICE": 17520.0,
"PRODUCT_AMOUNT": 3.0,
"TRANSACTION_ID": "100"}
```

6.4.2.2 Sample Update Message

```
{"table": "GG.TCUSTORD",
"op_type": "U",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:13:11.492000",
"pos": "00000000000000002891",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
 {"R": "AADPkvAAEAAEqLzAAA"},
"CUST_CODE": "BILL",
"ORDER_DATE": "1995-12-31:15:00:00",
"PRODUCT_CODE": "CAR",
"ORDER_ID": "765",
"PRODUCT_PRICE": 14000.0,
"PRODUCT_AMOUNT": 3.0,
"TRANSACTION_ID": "100"}
```

6.4.2.3 Sample Delete Message

```
{"table": "GG.TCUSTORD",
"op_type": "D",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:13:11.512000",
"pos": "00000000000000004338",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
 {"L": "206080450", "6": "9.0.80330", "R": "AADPkvAAEAAEqLzAAC"}, "CUST_CODE":
 "DAVE",
"ORDER_DATE": "1993-11-03:07:51:35",
"PRODUCT_CODE": "PLANE",
"ORDER_ID": "600",
"PRODUCT_PRICE": null,
"PRODUCT_AMOUNT": null,
"TRANSACTION_ID": null}
```

6.4.2.4 Sample Truncate Message

```
{"table": "GG.TCUSTORD",
"op_type": "T",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:13:11.514000",
"pos": "00000000000000004515",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
 {"R": "AADPkvAAEAAEqL2AAB"},
"CUST_CODE": null,
"ORDER_DATE": null,
"PRODUCT_CODE": null,
"ORDER_ID": null,
"PRODUCT_PRICE": null,
"PRODUCT_AMOUNT": null,
"TRANSACTION_ID": null}
```

6.4.3 Avro Schemas

Avro uses JSONs to represent schemas. Avro schemas define the format of generated Avro messages and are required to serialize and deserialize Avro messages. Schemas are generated on a JIT (just in time) basis when the first operation for a table is encountered. Generated Avro schemas are specific to a table definition that means that a separate Avro schema will be generated for every table encountered for processed operations. Avro schemas are by default written to the *GoldenGate_Home*/dirdef directory although the write location is configurable. Avro schema file names adhere to the following naming convention: *FullyQualifiedTableName*.avsc.

The following is a sample Avro schema for the Avro Row Format for the previous references examples:

```
{
  "type" : "record",
  "name" : "TCUSTORD",
  "namespace" : "GG",
  "fields" : [ {
    "name" : "table",
    "type" : "string"
  }, {
    "name" : "op_type",
    "type" : "string"
  }, {
    "name" : "op_ts",
    "type" : "string"
  }, {
    "name" : "current_ts",
    "type" : "string"
  }, {
    "name" : "pos",
    "type" : "string"
  }, {
    "name" : "primary_keys",
    "type" : {
      "type" : "array",
      "items" : "string"
    }
  }, {
    "name" : "tokens",
    "type" : {
      "type" : "map",
      "values" : "string"
    },
    "default" : { }
  }, {
    "name" : "CUST_CODE",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "ORDER_DATE",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "PRODUCT_CODE",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "ORDER_ID",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "PRODUCT_PRICE",
    "type" : [ "null", "double" ],
    "default" : null
  }, {
    "name" : "PRODUCT_AMOUNT",
    "type" : [ "null", "double" ],
    "default" : null
  }, {
    "name" : "TRANSACTION_ID",
    "type" : [ "null", "string" ],
    "default" : null
  } ]
}
```

6.4.4 Avro Row Configuration

Table 6-6 Avro Row Configuration Options

Table 6-6 (Cont.) Avro Row Configuration Options

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.insertOpKey`	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.
`gg.handler.`*`name`*`.format.updateOpKey`	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
`gg.handler.`*`name`*`.format.deleteOpKey`	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
`gg.handler.`*`name`*`.format.truncateOpKey`	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.
`gg.handler.`*`name`*`.format.encoding`	Optional	Any legal encoding name or alias supported by Java.	UTF-8 (the JSON default)	Controls the output encoding of generated Avro schema that is a JSON. The JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding.
`gg.handler.`*`name`*`.format.treatAllColumnsAsStrings`	Optional	`true` | `false`	`false`	Controls the output typing of generated Avro messages. If set to false then the formatter will attempt to map Oracle GoldenGate types to the corresponding AVRO type. If set to true then all data will be treated as Strings in the generated Avro messages and schemas.

Table 6-6 (Cont.) Avro Row Configuration Options

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name.`*`format.pkUpdateHandlingformat.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Provides configuration for how the formatter should handle update operations that change a primary key. Primary key operations can be problematic for the Avro Row formatter and require special consideration by you. • `abend` - indicates the process will abend. • `update` - indicates the process will treat this as a normal update. • `delete-insert` - indicates the process will treat this as a delete and an insert. Full supplemental logging needs to be enabled for this to work. Without full before and after row images the insert data will be incomplete.
`gg.handler.`*`name.`*`format.lineDelimiter`	Optional	Any string	no value	Optionally allows a user to insert a delimiter after each Avro message. This is not considered the best practice but in certain use cases customers may wish to parse a stream of data and extract individual Avro messages from the stream. This property allows the customer that option. Select a unique delimiter that cannot occur in any Avro message. This property supports `CDATA[]` wrapping.
`gg.handler.`*`name.`*`format.versionSchemas`	Optional	`true`| `false`	`false`	The created Avro schemas always follow the convention {fully qualified table name}`.avsc`. Setting this property to `true` creates an additional Avro schema in the schema directory named `{`*`fully qualified table name`*`}_{`*`current timestamp`*`}.avsc`. The additional Avro schema does not get destroyed or removed and thereby provides a history of schema evolution.

Table 6-6 (Cont.) Avro Row Configuration Options

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.wrapMessageInGenericAvroSchema`	Optional	`true|false`	`false`	Provides functionality to wrap the Avro messages for operations from the source trail file in a generic Avro wrapper message. For more information, see Generic Wrapper Functionality.
`gg.handler.`*`name`*`.format.schemaDirectory`	Optional	Any legal, existing file system path.	`./dirdef`	Controls the output location of generated Avro schemas.
`gg.handler.`*`name`*`.schemaFilePath=`	Optional	Any legal encoding name or alias supported by Java.	`./dirdef`	Controls the configuration property to a file directory inside of HDFS where you want schemas to be output. A metadata change event causes the schema to be overwritten when the next operation for the associated table is encountered. Schemas follow the same naming convention as schemas written to the local file system, *`catalog.schema.table`*`.avsc`.
`gg.handler.`*`name`*`.format.iso8601Format`	Optional	`true | false`	`true`	Controls the format of the current timestamp. The default is the ISO 8601 format. Set to `false` removes the "`T`" between the date and time in the current timestamp, which outputs " " instead.

6.4.5 Sample Configuration

The following is sample configuration for the Avro Row Formatter from the Java Adapter properties file:

```
gg.handler.hdfs.format=avro_row
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.encoding=UTF-8
gg.handler.hdfs.format.pkUpdateHandling=abend
gg.handler.hafs.format.wrapMessageInGenericAvroMessage=false
```

6.4.6 Metadata Change Events

The Avro Row Formatter is capable of taking action in the case of a metadata change event. This assumes that the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events. Metadata change events are of particular importance when formatting using Avro due to the tight dependency of Avro messages to its corresponding schema.

Metadata change events are handled seamlessly by the Avro Row Formatter and an updated Avro schema will be generated upon the first encounter of an operation of that table after the metadata change event. You should understand the impact of a metadata change event and change downstream targets to the new Avro schema. The tight dependency of Avro messages to Avro schemas may result in compatibility issues. Avro messages generated before the schema change may not be able to be deserialized with the newly generated Avro schema.

Conversely, Avro messages generated after the schema change may not be able to be deserialized with the previous Avro schema. It is a best practice to use the same version of the Avro schema that was used to generate the message. Consult the Apache Avro documentation for more details.

6.4.7 Special Considerations

This sections describes special considerations:

- Troubleshooting
- Primary Key Updates
- Generic Wrapper Functionality

6.4.7.1 Troubleshooting

Avro is a binary format therefore is not human readable. Since Avro messages are in binary format, it will be difficult to debug any issue. The Avro Row Formatter provides a special feature to mitigate this issue. When the `log4j` Java logging level is set to `TRACE` the created Avro messages will be deserialized and displayed in the log file as a JSON object. This allows you to view the structure and contents of the created Avro messages. `TRACE` should never be enabled in a production environment as it has substantial negative impact on performance. Alternatively, you may want to consider switching to use a formatter that produces human readable content for content troubleshooting. The XML or JSON formatters both produce content in human readable format that may facilitate troubleshooting.

6.4.7.2 Primary Key Updates

Primary key update operations require special consideration and planning for Big Data integrations. Primary key updates are update operations that modify one or more of the primary keys for the given row from the source database. Since data is simply appended in Big Data applications a primary key update operation looks more like a new insert than an update without any special handling. The Avro Row Formatter provides specialized handling for primary keys that is configurable by you. These are the configurable behaviors:

Table 6-7 Configurable behavior

Value	Description
`abend`	The default behavior is that the delimited text formatter will abend in the case of a primary key update.

Table 6-7 (Cont.) Configurable behavior

Value	Description
`update`	With this configuration the primary key update will be treated just like any other update operation. This configuration alternative should only be selected if you can guarantee that the primary key that is being changed is not being used as the selection criteria when selecting row data from a Big Data system.
`delete-insert`	Using this configuration the primary key update is treated as a special case of a delete using the before image data and an insert using the after image data. This configuration may more accurately model the effect of a primary key update in a Big Data application. However, if this configuration is selected it is important to have full supplemental logging enabled on Replication at the source database. Without full supplemental logging the delete operation will be correct, however, the insert operation will not contain all of the data for all of the columns for a full representation of the row data in the Big Data application.

6.4.7.3 Generic Wrapper Functionality

Avro messages are not self describing, which means that the receiver of the message must know the schema associated with the message before the message can be deserialized. Avro messages are binary and provide no consistent or reliable way to inspect the message contents in order to ascertain the message type. Therefore, Avro can be especially troublesome when messages are interlaced into a single stream of data like Kafka.

The Avro formatter provides a special feature to wrap the Avro message in a generic Avro message. This functionality is enabled by setting the following configuration parameter.

```
gg.handler.name.formatter.wrapMessageInGenericAvroMessage=true
```

The generic message is Avro message wrapping the Avro payload message that is common to all Avro messages that are output. The schema for the generic message is name `generic_wrapper.avsc` and is written to the output schema directory. This message has the following three fields.

- `table_name` - The fully qualified source table name.
- `schema_hash` - The hash code of the Avro schema generating the message.
- `payload` - The wrapped Avro message.

The following is the Avro Formatter generic wrapper schema.

```
{
  "type" : "record",
  "name" : "generic_wrapper",
  "namespace" : "oracle.goldengate",
  "fields" : [ {
    "name" : "table_name",
    "type" : "string"
  }, {
    "name" : "schema_hash",
    "type" : "int"
  }, {
    "name" : "payload",
    "type" : "bytes"
  } ]
}
```

6.5 Avro Operation Formatter

Apache Avro is an open source data serialization/deserialization framework known for its flexibility, compactness of serialized data, and good serialization/deserialization performance. Apache Avro is commonly used in Big Data applications.

The Avro Operation Formatter formats operation data from the source trail file into messages in an Avro binary array format. Each individual insert, update, delete and truncate operation will be formatted into an individual Avro message. The source trail file will contain the before and after images of the operation data. The Avro Operation Formatter takes that before and after image data and formats the data into an Avro binary representation of the operation data.

The Avro Operation Formatter formats operations from the source trail file into a format that represents the operation data. This format is more verbose than the output from the Avro Row Formatter for which the Avro messages model the row data.

6.5.1 Message Formatting Details

This section contains following topics:

- Operation Metadata
- Operation Data

6.5.1.1 Operation Metadata

Avro messages, generated by the Avro Operation Formatter, contain the following metadata fields that begin the message:

Table 6-8 Avro Messages and its Metadata

Fields	Description
`table`	*`CATALOG_NAME.SCHEMA NAME.TABLE NAME`*The fully qualified table name. The format of the fully qualified table name is the following:
`op_type`	The operation type that is the indicator of the type of database operation from the source trail file. Default values are "I" for insert, "U" for update, "D" for delete, "T" for truncate.
`op_ts`	The operation timestamp is the timestamp of the operation from the source trail file. Since this timestamp is from the source trail it is fixed. Replaying the trail file will result in the same timestamp for the same operation.
`current_ts`	The current timestamp is the current time when the formatter processed the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file will NOT result in the same timestamp for the same operation.

Table 6-8 (Cont.) Avro Messages and its Metadata

Fields	Description
pos	The trail file position with is the concatenated sequence number and rba number from the source trail file. The trail position provides traceability of the operation back to the source trail file. The sequence number is the source trail file number. The rba number is the offset in the trail file.
primary_keys	An array variable holding the column names of the primary keys of the source table.
tokens	A map variable holding the token key value pairs from the source trail file.

6.5.1.2 Operation Data

The operation data is represented as individual fields identified by the column names.

Column values for an operation from the source trail file can have one of three states: column has a value, column value is NULL, or column value is missing. Avro attributes only support two states, column has a value or column value is NULL. The Avro Operation Formatter contains an additional Boolean field for each column as a special indicator if the column value is missing or not. This special Boolean field is name `COLUMN_NAME_isMissing`. Using the combination of the `COLUMN_NAME` field, all three states can be defined.

- State 1: Column has a value

  `COLUMN_NAME` field has a value

  `COLUMN_NAME_isMissing` field is false

- State 2: Column value is NULL

  `COLUMN_NAME` field value is NULL

  `COLUMN_NAME_isMissing` field is false

- State 3: Column value is missing

  `COLUMN_NAME` field value is NULL

  `COLUMN_NAME_isMissing` field is true

The default setting of the Avro Row Formatter is to map the data types from the source trail file to the associated Avro data type. Avro supports few data types so this functionality largely results in the mapping of numeric fields from the source trail file to members typed as numbers. This data type mapping is configurable to alternatively treat all data as strings.

6.5.2 Sample Avro Operation Messages

Avro messages are binary and therefore not human readable. Sample messages the JSON representation of the messages displayed:

6.5.2.1 Sample Insert Message

```
{"table": "GG.TCUSTORD",
"op_type": "I",
"op_ts": "2013-06-02 22:14:36.000000",
"current_ts": "2015-09-18T10:17:49.570000",
"pos": "00000000000000001444",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
 {"R": "AADPkvAAEAAEqL2AAA"},
"before": null,
"after": {
"CUST_CODE": "WILL",
"CUST_CODE_isMissing": false,
"ORDER_DATE": "1994-09-30:15:33:00",
"ORDER_DATE_isMissing": false,
"PRODUCT_CODE": "CAR",
"PRODUCT_CODE_isMissing": false,
"ORDER_ID": "144", "ORDER_ID_isMissing": false,
"PRODUCT_PRICE": 17520.0,
"PRODUCT_PRICE_isMissing": false,
"PRODUCT_AMOUNT": 3.0, "PRODUCT_AMOUNT_isMissing": false,
"TRANSACTION_ID": "100",
"TRANSACTION_ID_isMissing": false}}
```

6.5.2.2 Sample Update Message

```
{"table": "GG.TCUSTORD",
"op_type": "U",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:17:49.880000",
"pos": "00000000000000002891",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
 {"R": "AADPkvAAEAAEqLzAAA"},
"before": {
"CUST_CODE": "BILL",
"CUST_CODE_isMissing": false,
"ORDER_DATE": "1995-12-31:15:00:00",
"ORDER_DATE_isMissing": false,
"PRODUCT_CODE": "CAR",
"PRODUCT_CODE_isMissing": false,
"ORDER_ID": "765",
"ORDER_ID_isMissing": false,
"PRODUCT_PRICE": 15000.0,
"PRODUCT_PRICE_isMissing": false,
"PRODUCT_AMOUNT": 3.0,
"PRODUCT_AMOUNT_isMissing": false,
"TRANSACTION_ID": "100",
"TRANSACTION_ID_isMissing": false},
"after": {
"CUST_CODE": "BILL",
"CUST_CODE_isMissing": false,
"ORDER_DATE": "1995-12-31:15:00:00",
"ORDER_DATE_isMissing": false,
"PRODUCT_CODE": "CAR",
"PRODUCT_CODE_isMissing": false,
"ORDER_ID": "765",
"ORDER_ID_isMissing": false,
"PRODUCT_PRICE": 14000.0,
"PRODUCT_PRICE_isMissing": false,
"PRODUCT_AMOUNT": 3.0,
"PRODUCT_AMOUNT_isMissing": false,
"TRANSACTION_ID": "100",
"TRANSACTION_ID_isMissing": false}}
```

6.5.2.3 Sample Delete Message

```
{"table": "GG.TCUSTORD",
"op_type": "D",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:17:49.899000",
"pos": "00000000000000004338",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
 {"L": "206080450", "6": "9.0.80330", "R": "AADPkvAAEAAEqLzAAC"}, "before": {
"CUST_CODE": "DAVE",
"CUST_CODE_isMissing": false,
"ORDER_DATE": "1993-11-03:07:51:35",
"ORDER_DATE_isMissing": false,
"PRODUCT_CODE": "PLANE",
"PRODUCT_CODE_isMissing": false,
"ORDER_ID": "600",
"ORDER_ID_isMissing": false,
"PRODUCT_PRICE": null,
"PRODUCT_PRICE_isMissing": true,
"PRODUCT_AMOUNT": null,
"PRODUCT_AMOUNT_isMissing": true,
"TRANSACTION_ID": null,
"TRANSACTION_ID_isMissing": true},
"after": null}
```

6.5.2.4 Sample Truncate Message

```
{"table": "GG.TCUSTORD",
"op_type": "T",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:17:49.900000",
"pos": "00000000000000004515",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
 {"R": "AADPkvAAEAAEqL2AAB"},
"before": null,
"after": null}
```

6.5.3 Avro Schema

Avro schemas are represented as JSONs. Avro schemas define the format of generated Avro messages and are required to serialize and deserialize Avro messages. Avro schemas are generated on a JIT (just in time) basis when the first operation for a table is encountered. Avro schemas are specific to a table definition, which means that a separate Avro schema will be generated for every table encountered for processed operations. Avro schemas are by default written to the `GoldenGate_Home/dirdef` directory although the write location is configurable. Avro schema file names adhere to the following naming convention: FullyQualifiedTableName`.avsc` directory although the write location is configurable. Avro schema file names adhere to the following naming convention: .

The following is a sample Avro schema for the Avro Operation Format for the previous references examples:

```
{
  "type" : "record",
  "name" : "TCUSTORD",
  "namespace" : "GG",
  "fields" : [ {
    "name" : "table",
    "type" : "string"
```

```
}, {
  "name" : "op_type",
  "type" : "string"
}, {
  "name" : "op_ts",
  "type" : "string"
}, {
  "name" : "current_ts",
  "type" : "string"
}, {
  "name" : "pos",
  "type" : "string"
}, {
  "name" : "primary_keys",
  "type" : {
    "type" : "array",
    "items" : "string"
  }
}, {
  "name" : "tokens",
  "type" : {
    "type" : "map",
    "values" : "string"
  },
  "default" : { }
}, {
  "name" : "before",
  "type" : [ "null", {
    "type" : "record",
    "name" : "columns",
    "fields" : [ {
      "name" : "CUST_CODE",
      "type" : [ "null", "string" ],
      "default" : null
    }, {
      "name" : "CUST_CODE_isMissing",
      "type" : "boolean"
    }, {
      "name" : "ORDER_DATE",
      "type" : [ "null", "string" ],
      "default" : null
    }, {
      "name" : "ORDER_DATE_isMissing",
      "type" : "boolean"
    }, {
      "name" : "PRODUCT_CODE",
      "type" : [ "null", "string" ],
      "default" : null
    }, {
      "name" : "PRODUCT_CODE_isMissing",
      "type" : "boolean"
    }, {
      "name" : "ORDER_ID",
      "type" : [ "null", "string" ],
      "default" : null
    }, {
      "name" : "ORDER_ID_isMissing",
      "type" : "boolean"
    }, {
      "name" : "PRODUCT_PRICE",
      "type" : [ "null", "double" ],
          "default" : null
        }, {
          "name" : "PRODUCT_PRICE_isMissing",
          "type" : "boolean"
        }, {
          "name" : "PRODUCT_AMOUNT",
          "type" : [ "null", "double" ],
          "default" : null
        }, {
          "name" : "PRODUCT_AMOUNT_isMissing",
          "type" : "boolean"
        }, {
          "name" : "TRANSACTION_ID",
          "type" : [ "null", "string" ],
          "default" : null
        }, {
          "name" : "TRANSACTION_ID_isMissing",
          "type" : "boolean"
        } ]
      } ],
      "default" : null
    }, {
      "name" : "after",
      "type" : [ "null", "columns" ],
      "default" : null
    } ]
  }
```

6.5.4 Avro Operation Formatter Configuration

Table 6-9 Configuration Options

Properties	Optional Y/N	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.insertOpKey`	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation
`gg.handler.`*`name`*`.format.updateOpKey`	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
`gg.handler.`*`name`*`.format.deleteOpKey`	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
`gg.handler.`*`name`*`.format.truncateOpKey`	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.

Table 6-9 (Cont.) Configuration Options

Properties	Optional Y/N	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.encoding`	Optional	Any legal encoding name or alias supported by Java	UTF-8 (the JSON default)	Controls the output encoding of generated Avro schema that is a JSON. JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding.
`gg.handler.`*`name`*`.format.treatAllColumnsAsStrings`	Optional	`true | false`	`false`	Controls the output typing of generated Avro messages. If set to false then the formatter will attempt to map Oracle GoldenGate types to the corresponding Avro type. If set to true then all data will be treated as Strings in the generated Avro messages and schemas.
`gg.handler.`*`name`*`.format.lineDelimiter`	Optional	Any string	no value	Optionally allows a user to insert a delimiter after each Avro message. This is not considered the best practice but in certain use cases customers may wish to parse a stream of data and extract individual Avro messages from the stream. This property allows the customer that option. Select a unique delimiter that cannot occur in any Avro message. This property supports `CDATA[]` wrapping.
`gg.handler.`*`name`*`.format.schemaDirectory`	Optional	Any legal, existing file system path.	`./dirdef`	Controls the output location of generated Avro schemas.
`gg.handler.`*`name`*`.format.wrapMessageInGenericAvroSchema`	Optional	`true|false`	`false`	Provides functionality to wrap the Avro messages for operations from the source trail file in a generic Avro wrapper message. For more information, see Generic Wrapper Functionality.

Table 6-9 (Cont.) Configuration Options

Properties	Optional Y/N	Legal Values	Default	Explanation
`gg.handler.name.format.iso8601Format`	Optional	`true` | `false`	`true`	Controls the format of the current timestamp. The default is the ISO 8601 format. Set to `false` removes the "T" between the date and time in the current timestamp, which outputs " " instead.

6.5.5 Sample Configuration

The following is a sample configuration for the Avro Operation Formatter from the Java Adapter properg.handlerties file:

```
gg.hdfs.format=avro_row
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.encoding=UTF-8
gg.handler.hafs.format.wrapMessageInGenericAvroMessage=false
```

6.5.6 Metadata Change Events

The Avro Operation Formatter is capable of taking action in the case of a metadata change event. This assumes that the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events. Metadata change events are of particular importance when formatting using Avro due to the tight dependency of Avro messages to its corresponding schema. Metadata change events are handled seamlessly by the Avro Operation Formatter and an updated Avro schema will be generated upon the first encounter of an operation of that table after the metadata change event. You should understand the impact of a metadata change event and change downstream targets to the new Avro schema. The tight dependency of Avro messages to Avro schemas may result in compatibility issues. Avro messages generated before the schema change may not be able to be deserialized with the newly generated Avro schema. Conversely, Avro messages generated after the schema change may not be able to be deserialized with the previous Avro schema. It is a best practice to use the same version of the Avro schema that was used to generate the message. Consult the Apache Avro documentation for more details.

6.5.7 Special Considerations

This section describes the special considerations:

6.5.7.1 Troubleshooting

Avro is a binary format therefore is not human readable. Since Avro messages are in binary format, it will be difficult to debug any issues. When the `log4j` Java logging level is set to `TRACE` the created Avro messages will be deserialized and displayed in the log file as a JSON object. This allows you to view the structure and contents of the created Avro messages. `TRACE` should never be enabled in a production environment as it has a substantial impact on performance.

6.5.7.2 Primary Key Updates

The Avro Operation Formatter creates messages with complete data of before and after images for update operations. Therefore, the Avro Operation Formatter requires no special treatment for primary key updates.

6.5.7.3 Generic Wrapper Message

Avro messages are not self describing, which means the receiver of the message must know the schema associated with the message before the message can be deserialized. Avro messages are binary and provide no consistent or reliable way to inspect the message contents in order to ascertain the message type. Therefore, Avro can be especially troublesome when messages are interlaced into a single stream of data like Kafka.

The Avro formatter provides a special feature to wrap the Avro message in a generic Avro message. This functionality is enabled by setting the following configuration parameter.

```
gg.handler.name.format.wrapMessageInGenericAvroMessage=true
```

The generic message is Avro message wrapping the Avro payload message that is common to all Avro messages that are output. The schema for the generic message is name `generic_wrapper.avsc` and is written to the output schema directory. This message has the following three fields.

- `table_name` - The fully qualified source table name.
- `schema_hash` - The hash code of the Avro schema generating the message.
- `payload` - The wrapped Avro message.

The following is the Avro Formatter generic wrapper schema:

```
{
  "type" : "record",
  "name" : "generic_wrapper",
  "namespace" : "oracle.goldengate",
  "fields" : [ {
    "name" : "table_name",
    "type" : "string"
  }, {
    "name" : "schema_hash",
    "type" : "int"
  }, {
    "name" : "payload",
    "type" : "bytes"
  } ]
}
]
}
```

6.6 Avro Object Container File Formatter

Oracle GoldenGate for Big Data can write to HDFS in Avro Object Container File (OCF) format. Using Avro OCF is a good choice for data formatting into HDFS because it handles schema evolution more efficiently than other formats. Compression and decompression is also supported in the Avro OCF Formatter to allow more efficient use of disk space.

The HDFS Handler integration with the Avro formatters to write files to HDFS in Avro OCF format is a specialized use case of the HDFS Handler. The Avro OCF format is required for Hive to be able to read Avro data in HDFS. The Avro OCF format is detailed in the Avro specification.

http://avro.apache.org/docs/current/spec.html#Object+Container+Files

Another important feature is that the HDFS Handler can be configured to stream data in Avro OCF format, generate table definitions in Hive, and update table definitions in Hive in the case of a metadata change event.

6.6.1 Avro OCF Formatter Configuration

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.insertOpKey`	Optional	Any string	`I`	Indicator to be inserted into the output record to indicate an insert operation.
`gg.handler.`*`name`*`.format.updateOpKey`	Optional	Any string	`U`	Indicator to be inserted into the output record to indicate an update operation.
`gg.handler.`*`name`*`.format.truncateOpKey`	Optional	Any string	`T`	Indicator to be truncated into the output record to indicate a truncate operation.
`gg.handler.`*`name`*`.format.deleteOpKey`	Optional	Any string	`D`	Indicator to be inserted into the output record to indicate a truncate operation.
`gg.handler.`*`name`*`.format.encoding`	Optional	Any legal encoding name or alias supported by Java.	`UTF-8`	Controls the output encoding of generated Avro schema, which is a JSON. JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding.

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.treatAllColumnsAsStrings`	Optional	`true | false`	`false`	Controls the output typing of generated Avro messages. If set to `false`, then the formatter attempts to map Oracle GoldenGate types to the corresponding Avro type. If set to `true`, then all data is treated as strings in the generated Avro messages and schemas.

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Controls how the formatter should handle update operations that change a primary key. Primary key operations can be problematic for the Avro Row formatter and require special consideration by you. • `abend` - indicates the process will abend • `update` - indicates the process will treat this as a normal update • `delete-insert` - indicates the process will treat this as a delete and an insert. The full before image is required for this feature to work properly. This can be achieved by using full supplemental logging in Oracle. Without full before and after row images the insert data will be incomplete.

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.generateSchema`	Optional	`true | false`	`true`	Schemas must be generated for Avro serialization so this property can be set to `false` to suppress the writing of the generated schemas to the local file system.
`gg.handler.`*`name`*`.format.schemaDirectory`	Optional	Any legal, existing file system path	`./dirdef`	Controls the output location of generated Avro schemas to the local file system. This property does not control where the Avro schema is written to in HDFS; that is controlled by an HDFS Handler property.
`gg.handler.`*`name`*`.format.iso8601Format`	Optional	`true | false`	`true`	The default format for the current timestamp is ISO8601. Set to `false` to remove the `T` between the date and time in the current timestamp and output a space instead.

Parameter	Optional / Required	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.versionSchemas`	Optional	`true | false`	`false`	If set to true, an Avro schema is created in the schema directory and versioned by a time stamp. The format of the schema is the following: *`{fully qualified table name}_{time stamp}`*`.avsc`

6.7 XML Formatter

The XML Formatter formats operation data from the source trail file into a XML documents. The XML Formatter takes that before and after image data and formats the data into an XML document representation of the operation data. The format of the XML document is effectively the same as the XML format in the previous releases of the Oracle GoldenGate Java Adapter product.

6.7.1 Message Formatting Details

The XML formatted messages contain the following information:

Table 6-10 XML formatting details

Value	Description
`table`	The fully qualified table name.
`type`	The operation type.
`current_ts`	The current timestamp is the time when the formatter processed the current operation record. This timestamp follows the ISO-8601 format and includes micro second precision. Replaying the trail file does not result in the same timestamp for the same operation.
`pos`	The position from the source trail file.
`numCols`	The total number of columns in the source table.
`col`	The col element is a repeating element that contains the before and after images of operation data.
`tokens`	The tokens element contains the token values from the source trail file.

6.7.2 Sample XML Messages

This sections provides sample XML messages.

6.7.2.1 Sample Insert Message

```
<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='I' ts='2013-06-02 22:14:36.000000'
current_ts='2015-10-06T12:21:50.100001' pos='00000000000000001444' numCols='7'>
 <col name='CUST_CODE' index='0'>
   <before missing='true'/>
   <after><![CDATA[WILL]]></after>
 </col>
 <col name='ORDER_DATE' index='1'>
   <before missing='true'/>
   <after><![CDATA[1994-09-30:15:33:00]]></after>
 </col>
 <col name='PRODUCT_CODE' index='2'>
   <before missing='true'/>
   <after><![CDATA[CAR]]></after>
 </col>
 <col name='ORDER_ID' index='3'>
   <before missing='true'/>
   <after><![CDATA[144]]></after>
 </col>
 <col name='PRODUCT_PRICE' index='4'>
   <before missing='true'/>
   <after><![CDATA[17520.00]]></after>
 </col>
 <col name='PRODUCT_AMOUNT' index='5'>
   <before missing='true'/>
   <after><![CDATA[3]]></after>
 </col>
 <col name='TRANSACTION_ID' index='6'>
   <before missing='true'/>
   <after><![CDATA[100]]></after>
 </col>
 <tokens>
   <token>
     <Name><![CDATA[R]]></Name>
     <Value><![CDATA[AADPkvAAEAAEqL2AAA]]></Value>
   </token>
 </tokens>
</operation>
```

6.7.2.2 Sample Update Message

```
<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='U' ts='2013-06-02 22:14:41.000000'
current_ts='2015-10-06T12:21:50.413000' pos='00000000000000002891' numCols='7'>
 <col name='CUST_CODE' index='0'>
   <before><![CDATA[BILL]]></before>
   <after><![CDATA[BILL]]></after>
 </col>
 <col name='ORDER_DATE' index='1'>
   <before><![CDATA[1995-12-31:15:00:00]]></before>
   <after><![CDATA[1995-12-31:15:00:00]]></after>
 </col>
 <col name='PRODUCT_CODE' index='2'>
   <before><![CDATA[CAR]]></before>
   <after><![CDATA[CAR]]></after>
 </col>
 <col name='ORDER_ID' index='3'>
   <before><![CDATA[765]]></before>
   <after><![CDATA[765]]></after>
 </col>
 <col name='PRODUCT_PRICE' index='4'>
   <before><![CDATA[15000.00]]></before>
   <after><![CDATA[14000.00]]></after>
 </col>
 <col name='PRODUCT_AMOUNT' index='5'>
   <before><![CDATA[3]]></before>
   <after><![CDATA[3]]></after>
 </col>
 <col name='TRANSACTION_ID' index='6'>
   <before><![CDATA[100]]></before>
   <after><![CDATA[100]]></after>
 </col>
 <tokens>
   <token>
     <Name><![CDATA[R]]></Name>
     <Value><![CDATA[AADPkvAAEAAEqLzAAA]]></Value>
   </token>
 </tokens>
</operation>
```

6.7.2.3 Sample Delete Message

```
<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='D' ts='2013-06-02 22:14:41.000000'
current_ts='2015-10-06T12:21:50.415000' pos='00000000000000004338' numCols='7'>
 <col name='CUST_CODE' index='0'>
   <before><![CDATA[DAVE]]></before>
   <after missing='true'/>
 </col>
 <col name='ORDER_DATE' index='1'>
   <before><![CDATA[1993-11-03:07:51:35]]></before>
   <after missing='true'/>
 </col>
 <col name='PRODUCT_CODE' index='2'>
   <before><![CDATA[PLANE]]></before>
   <after missing='true'/>
 </col>
 <col name='ORDER_ID' index='3'>
   <before><![CDATA[600]]></before>
   <after missing='true'/>
 </col>
 <col name='PRODUCT_PRICE' index='4'>
  <missing/>
 </col>
 <col name='PRODUCT_AMOUNT' index='5'>
  <missing/>
 </col>
 <col name='TRANSACTION_ID' index='6'>
  <missing/>
 </col>
 <tokens>
   <token>
     <Name><![CDATA[L]]></Name>
     <Value><![CDATA[206080450]]></Value>
   </token>
   <token>
     <Name><![CDATA[6]]></Name>
     <Value><![CDATA[9.0.80330]]></Value>
   </token>
   <token>
     <Name><![CDATA[R]]></Name>
     <Value><![CDATA[AADPkvAAEAAEqLzAAC]]></Value>
   </token>
 </tokens>
</operation>
```

6.7.2.4 Sample Truncate Message

```
<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='T' ts='2013-06-02 22:14:41.000000'
current_ts='2015-10-06T12:21:50.415001' pos='00000000000000004515' numCols='7'>
 <col name='CUST_CODE' index='0'>
   <missing/>
 </col>
 <col name='ORDER_DATE' index='1'>
   <missing/>
 </col>
 <col name='PRODUCT_CODE' index='2'>
   <missing/>
 </col>
 <col name='ORDER_ID' index='3'>
   <missing/>
 </col>
 <col name='PRODUCT_PRICE' index='4'>
  <missing/>
 </col>
 <col name='PRODUCT_AMOUNT' index='5'>
  <missing/>
 </col>
 <col name='TRANSACTION_ID' index='6'>
  <missing/>
 </col>
 <tokens>
   <token>
     <Name><![CDATA[R]]></Name>
     <Value><![CDATA[AADPkvAAEAAEqL2AAB]]></Value>
   </token>
 </tokens>
</operation>
```

6.7.3 XML Schema

An XML schema (XSD) is not generated as part of the XML Formatter functionality. The XSD is generic to any and all messages generated by the XML Formatter. An XSD defining the structure of output XML documents is defined as follows:

```
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="operation">
     <xs:complexType>
       <xs:sequence>
          <xs:element name="col" maxOccurs="unbounded" minOccurs="0">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="before" minOccurs="0">
                 <xs:complexType>
```

```
                   <xs:simpleContent>
                     <xs:extension base="xs:string">
                       <xs:attribute type="xs:string" name="missing"
use="optional"/>
                     </xs:extension>
                   </xs:simpleContent>
                 </xs:complexType>
               </xs:element>
               <xs:element name="after" minOccurs="0">
                 <xs:complexType>
                   <xs:simpleContent>
                     <xs:extension base="xs:string">
                       <xs:attribute type="xs:string" name="missing"
use="optional"/>
                     </xs:extension>
                   </xs:simpleContent>
                 </xs:complexType>
               </xs:element>
               <xs:element type="xs:string" name="missing" minOccurs="0"/>
             </xs:sequence>
             <xs:attribute type="xs:string" name="name"/>
             <xs:attribute type="xs:short" name="index"/>
           </xs:complexType>
         </xs:element>
         <xs:element name="tokens" minOccurs="0">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="token" maxOccurs="unbounded" minOccurs="0">
                 <xs:complexType>
                   <xs:sequence>
                     <xs:element type="xs:string" name="Name"/>
                     <xs:element type="xs:string" name="Value"/>
                   </xs:sequence>
                 </xs:complexType>
               </xs:element>
             </xs:sequence>
           </xs:complexType>
         </xs:element>
       </xs:sequence>
       <xs:attribute type="xs:string" name="table"/>
       <xs:attribute type="xs:string" name="type"/>
       <xs:attribute type="xs:string" name="ts"/>
       <xs:attribute type="xs:dateTime" name="current_ts"/>
       <xs:attribute type="xs:long" name="pos"/>
       <xs:attribute type="xs:short" name="numCols"/>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

6.7.4 XML Configuration

Table 6-11 Configuration Options

Table 6-11 (Cont.) Configuration Options

Parameter	Optional Y/N	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.insertOpKey`	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.
`gg.handler.`*`name`*`.format.updateOpKey`	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
`gg.handler.`*`name`*`.format.deleteOpKey`	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
`gg.handler.`*`name`*`.format.truncateOpKey`	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.
`gg.handler.`*`name`*`.format.encoding`	Optional	Any legal encoding name or alias supported by Java.	UTF-8 (the XML default)	Controls the output encoding of generated XML documents.
`gg.handler.`*`name`*`.format.includeProlog`	Optional	true | false	false	Controls the output of an XML prolog on generated XML documents. The XML prolog is optional for well formed XML. Sample XML prolog looks like `<?xml version='1.0' encoding='UTF-8' ?>`
`gg.handler.`*`name`*`.format.iso8601Format`	Optional	`true | false`	`true`	Controls the format of the current timestamp in the XML message. Set to `false` to suppress the `"T"` between the date and time and instead include blank space.

6.7.5 Sample Configuration

The following is sample configuration for the XML Formatter from the Java Adapter properties file:

```
gg.handler.hdfs.format=xml
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.encoding=ISO-8859-1
gg.handler.hdfs.format.includeProlog=false
```

6.7.6 Metadata Change Events

The XML Formatter will seamlessly handle metadata change events. The format of the XML document is such that a metadata change event does not even result in a change to the XML schema. The XML schema is designed to be generic so that the same schema represents the data of any operation from any table.

The XML Formatter is capable of taking action in the case of a metadata change event. This assumes that the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events. The format of the XML document is such that a metadata change event does not result in a change to the XML schema. The XML schema is generic so that the same schema represents the data of any operation form any table. The resulting changes in the metadata will be reflected in messages after the metadata change event. For example in the case of adding a column, the new column and column data will begin showing up in XML messages for that table after the metadata change event.

6.7.7 Primary Key Updates

Updates to a primary key require no special handling by the XML formatter. The XML formatter creates messages that model the database operations. For update operations this includes before and after images of column values. Primary key changes are simply represented in this format as a change to a column value just like a change to any other column value

7 Using the Metadata Provider

This chapter explains the Metadata Provider functionality, different types of Metadata Providers and examples that can be used to understand the functionality.

This chapter contains the following:

- Avro Metadata Provider
- Hive Metadata Provider

Valid only if handlers are configured to run with Replicat process.

The Replicat process provides functionality to perform source table to target table and source column to target column mapping using syntax in the Replicat configuration file. This mapping syntax is documented as part of the Oracle GoldenGate Replicat documentation. The source metadata definitions are included in the Oracle GoldenGate trail file (or by source definitions files for Oracle GoldenGate versions older than 12.2). When the replication target is a database, the Replicat process obtains the target metadata definitions from the target database. However, this is a shortcoming when pushing data to Big Data applications or Java Delivery in general. Big Data applications generally provide no target metadata therefore the Replicat mapping is not possible. The Metadata Provider exists to address this deficiency. The Metadata Provider can be used to define target metadata using either Avro or Hive which in turn enables source table to target table and source column to target column Replicat mapping. The use of the Metadata Provider is not required. If the metadata included in the source Oracle GoldenGate trail file is acceptable for the output, then do not use the Metadata Provider. The Metadata Provider should be used in the following cases:

- The requirement is for mapping source table names into target table names that do not match.
- The requirement is for mapping of source column names into target column name that do not match.
- The requirement is for the inclusion of certain columns from the source trail file and omitting other columns.

Replicat mapping has a general limitation in that the mapping defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 provides functionality for DDL propagation when using an Oracle Database as the source. Therefore the proper handling of schema evolution can be problematic when the Metadata Provider and Replicat mapping are being used. Users will need to consider their particular use cases for schema evolution and plan for how they will update the Metadata Provider and the Replicat mapping syntax for required changes.

For every table mapped in `REPLICAT` using `COLMAP`, the metadata will be retrieved from a configured metadata provider and retrieved metadata will then be used by `REPLICAT` for column mapping functionality.

Users have the choice of configuring one Metadata Provider implementation. Currently Hive and Avro Metadata Providers are supported. Details on configuring Hive and Avro Metadata Providers will be discussed in the following sections.

The Metadata Provider is optional and is enabled if the `gg.mdp.type` property is specified in the Java Adapter Properties file.

Scenarios - When to use Metadata Provider

1. The following scenarios do *not* require the Metadata Provider to be configured:

   The mapping of schema name whereby the source schema named `GG` is mapped to the target schema named `GGADP`.*

   The mapping of schema and table name whereby the schema `GG.TCUSTMER` is mapped to the table name `GGADP.TCUSTMER_NEW`

   ```
   MAP GG.*, TARGET GGADP.*;
   (OR)
   MAP GG.TCUSTMER, TARGET GG_ADP.TCUSTMER_NEW;
   ```

2. The following scenario requires Metadata Provider to be configured:

   The mapping of column names whereby the source column name does not match the target column name. For example source column `CUST_CODE` mapped to target column `CUST_CODE_NEW`

   ```
   MAP GG.TCUSTMER, TARGET GG_ADP.TCUSTMER_NEW, COLMAP(USEDEFAULTS,
   CUST_CODE_NEW=CUST_CODE, CITY2=CITY);
   ```

7.1 Avro Metadata Provider

The Avro Metadata Provider is used to retrieve the table metadata from Avro Schema files. For every table mapped in `REPLICAT` using `COLMAP`, the metadata will be retrieved from Avro Schema and retrieved metadata will then be used by `REPLICAT` for column mapping.

This section contains the following:

- Detailed Functionality
- Runtime Prerequisites
- Classpath Configuration
- Avro Metadata Provider Configuration
- Sample Configuration
- Metadata Change Event
- Limitations
- Troubleshooting

7.1.1 Detailed Functionality

The Avro Metadata Provider uses Avro schema definition files to retrieve metadata. The Avro schemas are defined using the `JSON`. For each table mapped in `Replicat.prm` file, a corresponding The Avro schema definition file should be created. More information on defining Avro schemas can be found at:

http://avro.apache.org/docs/current/gettingstartedjava.html#Defining+a+schema

Avro Metadata Provider Schema definition syntax:

```
{"namespace": "[$catalogname.]$schemaname",
"type": "record",
"name": "$tablename",
"fields": [
     {"name": "$col1", "type": "$datatype"},
     {"name": "$col2 ",  "type": "$datatype ", "primary_key":true},
     {"name": "$col3", "type": "$datatype ", "primary_key":true},
     {"name": "$col4", "type": ["$datatype","null"]}
   ]
}

namespace            - name of catalog/schema being mapped
name                 - name of the table being mapped
fields.name          - array of column names
fields.type          - datatype of the column
fields.primary_key   - indicates the column is part of primary key.

Representing nullable and not nullable columns:

"type":"$datatype" - indicates the column is not nullable, where "$datatype" is the
actual datatype.
"type": ["$datatype","null"] - indicates the column is nullable, where "$datatype"
is the actual datatype
```

The file naming convention for Avro schema files accessed by the Avro Metadata Provider must be in the following format:

```
[$catalogname.]$schemaname.$tablename.mdp.avsc

$catalogname    - name of the catalog if exists
$schemaname   - name of the schema
$tablename        - name of the table
.mdp.avsc          -  constant, which should be appended always
```

Supported Avro Data Types:

- boolean
- bytes
- double
- float
- int
- long
- string

For more information on Avro data types, see https://avro.apache.org/docs/1.7.5/spec.html#schema_primitive.

7.1.2 Runtime Prerequisites

The Avro schema definitions should be created for all tables mapped in Replicat's parameter file before starting the Replicat process.

7.1.3 Classpath Configuration

There is no additional classpath setting required for Avro Metadata Provider.

7.1.4 Avro Metadata Provider Configuration

The configuration properties of Oracle GoldenGate Avro Metadata Provider are detailed in this section

Property	Required/ Optional	Legal Values	Default	Explanation
`gg.mdp.type`	Required	`avro`	-	Selects Avro Metadata Provider
`gg.mdp.schemaFilesPath`	Required	Example for a legal value could be `/home/user/ggadp/avroschema/`	-	Path to Avro schema files directory
`gg.mdp.charset`	Optional	Valid character set	UTF-8	Specifies the character set of the column with character data type. This is used to convert the source data from trail file to the correct target charset.
`gg.mdp.nationalCharset`	Optional	Valid character set	UTF-8	Specifies the character set of the column with character data type. This is used to convert the source data from trail file to the correct target charset. Example: Indicates character set of columns like `NCHAR`, `NVARCHAR` in case of Oracle Database.

7.1.5 Sample Configuration

This section provides an example for configuring the Avro Metadata Provider. Consider a source with following table:

```
TABLE GG.TCUSTMER {
     CUST_CODE VARCHAR(4) PRIMARY KEY,
     NAME VARCHAR(100),
     CITY VARCHAR(200),
     STATE VARCHAR(200)
}
```

Map column `CUST_CODE (GG.TCUSTMER)` in source to `CUST_CODE2 (GG_AVRO.TCUSTMER_AVRO)` on target and column `CITY (GG.TCUSTMER)` in source to `CITY2 (GG_AVRO.TCUSTMER_AVRO)` on target.

Mapping in `Replicat .prm` file:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS,
CUST_CODE2=CUST_CODE, CITY2=CITY);

Mapping definition in this example:
1.Source schema GG is mapped to target schema GG_AVRO
2.Source column CUST_CODE is mapped to target column CUST_CODE2
3.Source column CITY is mapped to target column CITY2
4.USEDEFAULTS specifies that rest of the columns names are same on both source and
target (NAME and STATE columns).
```

The Avro schema definition file for the preceding example:

File path: `/home/ggadp/avromdpGG_AVRO.TCUSTMER_AVRO.mdp.avsc`

```
{"namespace": "GG_AVRO",
"type": "record",
"name": "TCUSTMER_AVRO",
"fields": [
     {"name": "NAME", "type": "string"},
    {"name": "CUST_CODE2",  "type": "string", "primary_key":true},
     {"name": "CITY2", "type": "string"},
     {"name": "STATE", "type": ["string","null"]}
]
}
```

The configuration in the Java Adapter Properties file will include the following:

```
gg.mdp.type = avro
gg.mdp.schemaFilesPath = /home/ggadp/avromdp
```

Following is the sample output using delimited text formatter with "`;`" as delimiter for the preceding example.

```
I;GG_AVRO.TCUSTMER_AVRO;2013-06-02 22:14:36.000000;NAME;BG SOFTWARE
CO;CUST_CODE2;WILL;CITY2;SEATTLE;STATE;WA
```

The Oracle GoldenGate for Big Data installation include a sample Replicat configuration file, a sample Java Adapter properties file, and sample Avro schemas at:

`GoldenGate_install_directory``/AdapterExamples/big-data/metadata_provider/avro`

7.1.6 Metadata Change Event

The Avro schema definitions and the mappings in the Replicat configuration file may need to be modified if there is a DDL change in the source database tables. You may wish to abort or suspend the Replicat process in the case of a metadata change event. The Replicat process can be aborted by adding the following to the Replicat configuration file (`.prm file`).

```
DDL INCLUDE ALL, EVENTACTIONS (ABORT)
```

Alternatively, the Replicat process can be suspended by adding the following to the Replication configuration file (`.prm file`).

```
DDL INCLUDE ALL, EVENTACTIONS (SUSPEND)
```

7.1.7 Limitations

Avro bytes data type cannot be used as primary key.

The source to target mapping defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 supports DDL propagation and source schema evolution for Oracle Databases as the replication source. However, evolution of the source schemas may be problematic the static mapping configured in the Replicat configuration file.

7.1.8 Troubleshooting

This section contains the following:

- Invalid Schema Files Location
- Invalid Schema File Name
- Invalid Namespace in Schema File
- Invalid Table Name in Schema File

7.1.8.1 Invalid Schema Files Location

The Avro schema files directory location specified by the configuration property `gg.mdp.schemaFilesPath` should be a valid directory. Failure to configure a valid directory in `gg.mdp.schemaFilesPath` property leads to following exception:

```
oracle.goldengate.util.ConfigException: Error initializing Avro metadata provider
Specified schema location does not exist. {/path/to/schema/files/dir}
```

7.1.8.2 Invalid Schema File Name

For every table mapped in `Replicat.prm` file, a corresponding Avro schema file must be created in the directory specified in `gg.mdp.schemaFilesPath`.

For example, consider the following scenario:

Mapping:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS,
cust_code2=cust_code, CITY2 = CITY);
```

Property:

```
gg.mdp.schemaFilesPath=/home/usr/avro/
```

A file called `GG_AVRO.TCUSTMER_AVRO.mdp.avsc` must be created in the `/home/usr/avro/` directory. that is, `/home/usr/avro/GG_AVRO.TCUSTMER_AVRO.mdp.avsc`

Failing to create the `/home/usr/avro/GG_AVRO.TCUSTMER_AVRO.mdp.avsc` file results in the following exception:

```
java.io.FileNotFoundException: /home/usr/avro/GG_AVRO.TCUSTMER_AVRO.mdp.avsc
```

7.1.8.3 Invalid Namespace in Schema File

The target schema name specified in `REPLICAT` mapping must be same as namespace in the Avro schema definition file.

For example, consider the following scenario:

Mapping:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS, cust_code2 =
cust_code, CITY2 = CITY);

Avro Schema Definition:

{
"namespace": "GG_AVRO",
..
}
```

In this scenario, `REPLICAT` abends with following exception if the target schema name specified in Replicat mapping does not match with Avro schema namespace:

```
Unable to retrieve table matadata. Table : GG_AVRO.TCUSTMER_AVRO
Mapped [catalogname.]schemaname (GG_AVRO) does not match with the schema namespace
{schema namespace}
```

7.1.8.4 Invalid Table Name in Schema File

The target table name specified in Replicat mapping must be same as name in the Avro schema definition file.

For example, consider the following scenario:

Mapping:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS, cust_code2 =
cust_code, CITY2 = CITY);

Avro Schema Definition:

{
"namespace": "GG_AVRO",
"name": "TCUSTMER_AVRO",
..
}
```

In this scenario, `REPLICAT` abends with following exception if the target table name specified in Replicat mapping does not match with Avro schema name.

```
Unable to retrieve table matadata. Table : GG_AVRO.TCUSTMER_AVRO
Mapped table name (TCUSTMER_AVRO) does not match with the schema table name {table
name}
```

7.2 Hive Metadata Provider

The Hive Metadata Provider is used to retrieve the table metadata from a Hive metastore. The metadata will be retrieved from Hive for every target table mapped in the Replicat properties file using the `COLMAP` syntax. The retrieved target metadata will be used by Replicat for the column mapping functionality.

This sections contains the following:

7.2.1 Detailed Functionality

The Hive Metadata Provider uses both Hive JDBC and HCatalog interfaces to retrieve metadata from the Hive metastore. For each table mapped in `Replicat.prm` file, a corresponding table should be created in Hive.

The default Hive configuration starts an embedded/local metastore Derby database. Apache Derby is designed to be an embedded database and only allows a single connection. The single connection limitation of the Derby Database as the Hive Metastore implementation means that it will not function when working with the Oracle GoldenGate for Big Data Hive Metadata Provider. To overcome this limitation Hive should be configured with a remote metastore database. More information on configuring Hive with remote metastore database can found at:

`https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-RemoteMetastoreDatabase`

Hive does not support Primary Key semantics, so the metadata retrieved from Hive metastore will not include any Primary Key definition. Replicat's `KEYCOLS` configuration syntax should instead be used to define primary keys when the Hive Metadata Provider is used.

KEYCOLS

The Replicat mapping configuration syntax `KEYCOLS` must be used to define primary keys in the target schema. The Oracle GoldenGate HBase Handler requires primary keys. Therefore, setting primary keys in the target schema is required when Replicat mapping is employed with HBase as the target. Additionally, the output of the Avro Formatters includes an Array field to hold the primary column names. Therefore, if Replicat mapping is employed with the Avro Formatters users should consider using `KEYCOLS` to identify the primary key columns.

Examples of configuring `KEYCOLS` will be discussed in Sample Configuration.

Supported Hive Data types:

- BIGINT
- BINARY

- BOOLEAN
- CHAR
- DATE
- DECIMAL
- DOUBLE
- FLOAT
- INT
- SMALLINT
- STRING
- TIMESTAMP
- TINYINT
- VARCHAR

For more information on Hive data types, see `https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types`.

7.2.2 Runtime Prerequisites

1. Configuring Hive with a Remote Metastore Database

   A list of supported databases that can be used to configure remote Hive metastore can be found at `https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-SupportedBackendDatabasesforMetastore`. In the following example, a MySQL database is configured as the Hive metastore. Configure the following properties in the `${HIVE_HOME}/conf/hive-site.xml` Hive configuration file.

   Note:

   The `ConnectionURL` and driver class used in this example are specific to MySQL database. Change the values appropriately if any database other than MySQL is chosen.

```
<property>
         <name>javax.jdo.option.ConnectionURL</name>
         <value>jdbc:mysql://MYSQL_DB_IP:MYSQL_DB_PORT/DB_NAME?
createDatabaseIfNotExist=false</value>
 </property>

 <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
 </property>

 <property>
          <name>javax.jdo.option.ConnectionUserName</name>
     <value>MYSQL_CONNECTION_USERNAME</value>
```

```
</property>

<property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>MYSQL_CONNECTION_PASSWORD</value>
</property>
```

The list of parameters to be configured in the `hive-site.xml` file for a remote metastore can be found at `https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-RemoteMetastoreDatabase`.

Note:

MySQL jdbc connector jar should be added in the Hive classpath that is,

a. In *`HIVE_HOME`*`/lib/` directory. *`DB_NAME`* should be replaced by a valid database name created in MySQL.

b. Start the Hive Server:

   `HIVE_HOME``/bin/hiveserver2/bin/hiveserver2`

c. Start the Hive Remote Metastore Server:

   `HIVE_HOME``/bin/hive --service metastore`

7.2.3 Classpath Configuration

You must configure two things in the `gg.classpath` configuration variable in order for the Oracle GoldenGate for Big Data Hive Metadata Provider to connect to Hive and run. The first is the `hive-site.xml` file and the second are the Hive and HDFS client jars. The client jars must match the version of Hive that the Oracle GoldenGate for Big Data Hive Metadata Provider is connecting.

1. Create `hive-site.xml` file with the following properties:

```
<configuration>
<!-- Mandatory Property -->
<property>
<name>hive.metastore.uris</name>
<value>thrift://HIVE_SERVER_HOST_IP:9083</value>
<property>

<!-- Optional Property. Default value is 5 -->
<property>
<name>hive.metastore.connect.retries</name>
<value>3</value>
</property>

<!-- Optional Property. Default value is 1 -->
<property>
<name>hive.metastore.client.connect.retry.delay</name>
<value>10</value>
</property>

<!-- Optional Property. Default value is 600 seconds -->
<property>
```

```
<name>hive.metastore.client.socket.timeout</name>
<value>50</value>
</property>

 </configuration>
```

> **Note:**
>
> For example, if the `hive-site.xml` file is created in `/home/user/oggadp/dirprm` directory, then `gg.classpath` entry will look like `gg.classpath=/home/user/oggadp/dirprm/`

2. The default location of the Hive and HDFS client jars are the following directories:

```
HIVE_HOME/hcatalog/share/hcatalog/*
HIVE_HOME/lib/*
HIVE_HOME/hcatalog/share/webhcat/java-client/*
HADOOP_HOME/share/hadoop/common/*
HADOOP_HOME/share/hadoop/common/lib/*
HADOOP_HOME/share/hadoop/mapreduce/*
```

Configure the `gg.classpath` exactly as shown in the preceding example. Creating a path to the `hive-site.xml` should simply contain the path with no wild card appended. The inclusion of the * wildcard in the path to the `hive-site.xml` file will cause it not to be picked up. Conversely, creating a path to the dependency jars should include the * wild card character in order to include all of the jar files in that directory in the associated classpath. Do not use `*.jar.`

7.2.4 Hive Metadata Provider Configuration

The configuration properties of the Hive Metadata Provider are detailed in this section.

Property	Required/ Optional	Legal Values	Default	Explanation
`gg.mdp.type`	Required	`hive`	-	Selects Hive Metadata Provider
`gg.mdp.connectionUrl`	Required	Format without Kerberos Authentication: `jdbc:hive2://`*`HIVE_SERVER_IP:HIVE_JDBC_PORT/HIVE_DB`* Format with Kerberos Authentication: `jdbc:hive2://`*`HIVE_SERVER_IP:HIVE_JDBC_PORT/HIVE_DB;`* `principal=user/FQDN@MY.REALM`	-	JDBC Connection URL of Hive Server
`gg.mdp.driverClassName`	Required	`org.apache.hive.jdbc.HiveDriver`	-	Fully qualified Hive JDBC Driver class name.

Property	Required/ Optional	Legal Values	Default	Explanation
`gg.mdp.userName`	Optional	Valid username	""	Username to connect to Hive Database. `userName` property is not required when Kerberos Authentication is used. The Kerberos principal should be specified in the connection url as specified in `connectionUrl` property's legal values.
`gg.mdp.password`	Optional	Valid Password	""	Password to connect to Hive Database
`gg.mdp.charset`	Optional	Valid character set	UTF-8	Specifies the character set of the column with character data type. This is used to convert the source data from trail file to the correct target charset.
`gg.mdp.nationalCharset`	Optional	Valid character set	UTF-8	Specifies the character set of the column with character data type. This is used to convert the source data from trail file to the correct target charset. For example, Indicates character set of columns like `NCHAR`, `NVARCHAR` in case of Oracle Database.
`gg.mdp.authType`	Optional	kerberos	none	
`gg.mdp.kerberosKeytabFile`	Optional (Required if `authType=kerberos`)	Relative or absolute path to a Kerberos keytab file.	-	The keytab file allows Hive to access a password to perform kinit operation for Kerberos security.
`gg.mdp.kerberosPrincipal`	Optional (Required if `authType=kerberos`)	A legal Kerberos principal name(`user/FQDN@MY.REALM`)	-	The Kerberos principal name for Kerberos authentication.

7.2.5 Sample Configuration

This section provides an example for configuring the Hive Metadata Provider.

Consider a source with following table:

```
TABLE GG.TCUSTMER {
     CUST_CODE VARCHAR(4)   PRIMARY KEY,
     NAME VARCHAR(100),
     CITY VARCHAR(200),
     STATE VARCHAR(200)}
```

The example maps the column `CUST_CODE (GG.TCUSTMER)` in the source to `CUST_CODE2 (GG_HIVE.TCUSTMER_HIVE)` on the target and column `CITY` (`GG.TCUSTMER`) in the source to `CITY2` `(GG_HIVE.TCUSTMER_HIVE)`on the target.

Mapping configuration in the `Replicat.prm` file:

```
MAP GG.TCUSTMER, TARGET GG_HIVE.TCUSTMER_HIVE, COLMAP(USEDEFAULTS,
CUST_CODE2=CUST_CODE, CITY2=CITY) KEYCOLS(CUST_CODE2);
```

The mapping definition for this example is:

- Source schema `GG` is mapped to target schema `GG_HIVE`
- Source column `CUST_CODE` is mapped to target column `CUST_CODE2`
- Source column `CITY` is mapped to target column `CITY2`
- `USEDEFAULTS` specifies that rest of the columns names are same on both source and target (`NAME` and `STATE` columns).
- `KEYCOLS` is used to specify that `CUST_CODE2` should be treated as primary key.

Since primary keys cannot be specified in Hive DDL, `KEYCOLS` is used to specify the primary keys.

Create schema and tables in Hive for the preceding example:

Note:

You can choose any schema name and are not restricted to the `gg_hive schema` name. The Hive schema can be pre-existing or newly created. You do this by modifying the connection URL (`gg.mdp.connectionUrl`) in the Java Adapter properties file and the mapping configuration in the Replicat `.prm` file. Once the schema name is changed, the connection URL (`gg.mdp.connectionUrl`) and mapping in Replicat `.prm` file should be updated.

To start the Hive CLI type the following command:

`HIVE_HOME``/bin/hive`

To create a schema, `GG_HIVE`, in Hive, use the following command:

```
hive> create schema gg_hive;
OK
Time taken: 0.02 seconds
```

To create a table `TCUSTMER_HIVE` in `GG_HIVE` database type the following command:

```
hive> CREATE EXTERNAL TABLE `TCUSTMER_HIVE`(
    >   "CUST_CODE2" VARCHAR(4),
    >   "NAME" VARCHAR(30),
    >   "CITY2" VARCHAR(20),
    >   "STATE" STRING);
OK
Time taken: 0.056 seconds
```

Configuration in `.properties` file can be like the following:

```
gg.mdp.type=hive
gg.mdp.connectionUrl=jdbc:hive2://<HIVE_SERVER_IP>:10000/gg_hive
gg.mdp.driverClassName=org.apache.hive.jdbc.HiveDriver
```

Following is the sample output using delimited text formatter with ";" as delimiter for the preceding example.

```
I;GG_HIVE.TCUSTMER_HIVE;2015-10-07T04:50:47.519000;cust_code2;WILL;name;BG SOFTWARE
CO;city2;SEATTLE;state;WA
```

A sample Replicat configuration file, Java Adapter properties file, and a Hive create table SQL script are included with the installation, and located at:

`GoldenGate_install_directory/``AdapterExamples/big-data/metadata_provider/hive`

7.2.6 Security

The Hive server can be secured using Kerberos Authentication. Refer to the Hive documentation for your specific Hive release for instructions on how to secure the Hive server. The Oracle GoldenGate for Big Data Hive Metadata Provider can connect to Kerberos secured Hive server.

The HDFS `core-site.xml` and `hive-site.xml` should be in handler's classpath.

Following properties should be enabled in `core-site.xml`:

```
<property>
<name>hadoop.security.authentication</name>
<value>kerberos</value>
</property>

<property>
<name>hadoop.security.authorization</name>
<value>true</value>
</property>
```

Following properties should be enabled in `hive-site.xml`

```
<property>
<name>hive.metastore.sasl.enabled</name>
<value>true</value>
</property>

<property>
<name>hive.metastore.kerberos.keytab.file</name>
<value>/path/to/keytab</value> <!-- Change this value -->
</property>

<property>
<name>hive.metastore.kerberos.principal</name>
<value>Kerberos Principal</value> <!-- Change this value -->
```

```
</property>

<property>
   <name>hive.server2.authentication</name>
    <value>KERBEROS</value>
</property>

<property>
   <name>hive.server2.authentication.kerberos.principal</name>
    <value>Kerberos Principal</value> <!-- Change this value -->
</property>

<property>
    <name>hive.server2.authentication.kerberos.keytab</name>
    <value>/path/to/keytab</value> <!-- Change this value -->
</property>
```

7.2.7 Metadata Change Event

Tables in Hive metastore should be updated/altered/created manually if there is a change in source database tables. You may wish to abort or suspend the Replicat process in the case of a metadata change event. The Replicat process can be aborted by adding the following to the Replicat configuration file (`.prm` file).

```
DDL INCLUDE ALL, EVENTACTIONS (ABORT)
```

Alternatively, the Replicat process can be suspended by adding the following to the Replication configuration file (`.prm` file).

```
DDL INCLUDE ALL, EVENTACTIONS (SUSPEND)
```

7.2.8 Limitations

Columns with binary data type cannot be used as primary key.

The source to target mapping defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 supports DDL propagation and source schema evolution for Oracle Databases as the replication source. However, evolution of the source schemas may be problematic the static mapping configured in the Replicat configuration file.

7.2.9 Common Pitfalls

The most common problems encountered are the Java classpath issues. Oracle Hive Metadata Provider requires certain Hive and HDFS client libraries to be resolved in its classpath as a prerequisite.

The required client jar directories are listed in Classpath Configuration. Hive and HDFS client jars do not ship with Oracle GoldenGate for Big Data product. The client jars should be the same version as the Hive version to which Hive Metadata Provider is connecting.

In order to establish a connection to the Hive server, the `hive-site.xml` file must be in the classpath.

7.2.10 Troubleshooting

The Replicat process will abend with a *"Table metadata resolution exception"* if the mapped target table does not exist in Hive.

For example, consider the following mapping:

```
MAP GG.TCUSTMER, TARGET GG_HIVE.TCUSTMER_HIVE, COLMAP(USEDEFAULTS,
CUST_CODE2=CUST_CODE, CITY2=CITY) KEYCOLS(CUST_CODE2);
```

The preceding mapping requires a table called `TCUSTMER_HIVE` to be created in schema `GG_HIVE` in the Hive metastore. Failure to create the `GG_HIVE.TCUSTMER_HIVE` table in Hive will result in following exception:

```
ERROR [main) - Table Metadata Resolution Exception
Unable to retrieve table matadata. Table : GG_HIVE.TCUSTMER_HIVE
NoSuchObjectException(message:GG_HIVE.TCUSTMER_HIVE table not found)
```

A HBase Handler Client Dependencies

This appendix lists the HBase client dependencies for Apache HBase. The `hbase-client-x.x.x.jar` is not distributed with Apache HBase nor is it mandatory to be in the classpath. The `hbase-client-x.x.x.jar` is an empty maven project with the purpose of aggregating all of the HBase client dependencies.

- **Maven groupId**: `org.apache.hbase`
- **Maven atifactId**: `hbase-client`
- **Maven version**: the HBase version numbers listed for each section

A.1 HBase Client Dependencies

This section lists the Hadoop client dependencies for each HBase version.

- HBase 1.1.1
- HBase 1.0.1.1

A.1.1 HBase 1.1.1

HBase 1.1.1 (HBase 1.1.0.1 is effectively the same, simply substitute 1.1.0.1 on the libraries versioned as 1.1.1)

```
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.9.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-el-1.0.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.2.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
findbugs-annotations-1.3.9-1.jar
guava-12.0.1.jar
hadoop-annotations-2.5.1.jar
```

```
hadoop-auth-2.5.1.jar
hadoop-common-2.5.1.jar
hadoop-mapreduce-client-core-2.5.1.jar
hadoop-yarn-api-2.5.1.jar
hadoop-yarn-common-2.5.1.jar
hamcrest-core-1.3.jar
hbase-annotations-1.1.1.jar
hbase-client-1.1.1.jar
hbase-common-1.1.1.jar
hbase-protocol-1.1.1.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jaxb-api-2.2.2.jar
jcodings-1.0.8.jar
jdk.tools-1.7.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.42.jar
jsr305-1.3.9.jar
junit-4.11.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
netty-all-4.0.23.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.6.1.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

A.1.2 HBase 1.0.1.1

```
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.9.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-el-1.0.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.2.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
findbugs-annotations-1.3.9-1.jar
guava-12.0.1.jar
```

```
hadoop-annotations-2.5.1.jar
hadoop-auth-2.5.1.jar
hadoop-common-2.5.1.jar
hadoop-mapreduce-client-core-2.5.1.jar
hadoop-yarn-api-2.5.1.jar
hadoop-yarn-common-2.5.1.jar
hamcrest-core-1.3.jar
hbase-annotations-1.0.1.1.jar
hbase-client-1.0.1.1.jar
hbase-common-1.0.1.1.jar
hbase-protocol-1.0.1.1.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jaxb-api-2.2.2.jar
jcodings-1.0.8.jar
jdk.tools-1.7.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.42.jar
jsr305-1.3.9.jar
junit-4.11.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
netty-all-4.0.23.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.6.1.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

B HDFS Handler Client Dependencies

This appendix lists the HDFS client dependencies for Apache Hadoop. The `hadoop-client-x.x.x.jar` is not distributed with Apache Hadoop nor is it mandatory to be in the classpath. The `hadoop-client-x.x.x.jar` is an empty maven project with the purpose of aggregating all of the Hadoop client dependencies.

Maven groupId: `org.apache.hadoop`

Maven atifactId: `hadoop-client`

Maven version: the HDFS version numbers listed for each section

B.1 Hadoop Client Dependencies

This section lists the Hadoop client dependencies for each HDFS version.

- HDFS 2.7.1
- HDFS 2.6.0
- HDFS 2.5.2
- HDFS 2.4.1
- HDFS 2.3.0
- HDFS 2.2.0

B.1.1 HDFS 2.7.1

HDFS 2.7.1 (HDFS 2.7.0 is effectively the same, simply substitute 2.7.0 on the libraries versioned as 2.7.1)

```
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
```

```
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.7.1.jar
hadoop-auth-2.7.1.jar
hadoop-client-2.7.1.jar
hadoop-common-2.7.1.jar
hadoop-hdfs-2.7.1.jar
hadoop-mapreduce-client-app-2.7.1.jar
hadoop-mapreduce-client-common-2.7.1.jar
hadoop-mapreduce-client-core-2.7.1.jar
hadoop-mapreduce-client-jobclient-2.7.1.jar
hadoop-mapreduce-client-shuffle-2.7.1.jar
hadoop-yarn-api-2.7.1.jar
hadoop-yarn-client-2.7.1.jar
hadoop-yarn-common-2.7.1.jar
hadoop-yarn-server-common-2.7.1.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsp-api-2.1.jar
jsr305-3.0.0.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
netty-3.7.0.Final.jar
netty-all-4.0.23.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.10.jar
slf4j-log4j12-1.7.10.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xercesImpl-2.9.1.jar
xml-apis-1.3.04.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

B.1.2 HDFS 2.6.0

```
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
```

```
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.6.0.jar
curator-framework-2.6.0.jar
curator-recipes-2.6.0.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.6.0.jar
hadoop-auth-2.6.0.jar
hadoop-client-2.6.0.jar
hadoop-common-2.6.0.jar
hadoop-hdfs-2.6.0.jar
hadoop-mapreduce-client-app-2.6.0.jar
hadoop-mapreduce-client-common-2.6.0.jar
hadoop-mapreduce-client-core-2.6.0.jar
hadoop-mapreduce-client-jobclient-2.6.0.jar
hadoop-mapreduce-client-shuffle-2.6.0.jar
hadoop-yarn-api-2.6.0.jar
hadoop-yarn-client-2.6.0.jar
hadoop-yarn-common-2.6.0.jar
hadoop-yarn-server-common-2.6.0.jar
htrace-core-3.0.4.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xercesImpl-2.9.1.jar
xml-apis-1.3.04.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

B.1.3 HDFS 2.5.2

HDFS 2.5.2 (HDFS 2.5.1 and 2.5.0 are effectively the same, simply substitute 2.5.1 or 2.5.0 on the libraries versioned as 2.5.2)

```
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
guava-11.0.2.jar
hadoop-annotations-2.5.2.jar
adoop-auth-2.5.2.jar
hadoop-client-2.5.2.jar
hadoop-common-2.5.2.jar
hadoop-hdfs-2.5.2.jar
hadoop-mapreduce-client-app-2.5.2.jar
hadoop-mapreduce-client-common-2.5.2.jar
hadoop-mapreduce-client-core-2.5.2.jar
hadoop-mapreduce-client-jobclient-2.5.2.jar
hadoop-mapreduce-client-shuffle-2.5.2.jar
hadoop-yarn-api-2.5.2.jar
hadoop-yarn-client-2.5.2.jar
hadoop-yarn-common-2.5.2.jar
hadoop-yarn-server-common-2.5.2.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
```

```
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

B.1.4 HDFS 2.4.1

HDFS 2.4.1 (HDFS 2.4.0 is effectively the same, simply substitute 2.4.0 on the libraries versioned as 2.4.1)

```
activation-1.1.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
guava-11.0.2.jar
hadoop-annotations-2.4.1.jar
hadoop-auth-2.4.1.jar
hadoop-client-2.4.1.jar
hadoop-hdfs-2.4.1.jar
hadoop-mapreduce-client-app-2.4.1.jar
hadoop-mapreduce-client-common-2.4.1.jar
hadoop-mapreduce-client-core-2.4.1.jar
hadoop-mapreduce-client-jobclient-2.4.1.jar
hadoop-mapreduce-client-shuffle-2.4.1.jar
hadoop-yarn-api-2.4.1.jar
hadoop-yarn-client-2.4.1.jar
hadoop-yarn-common-2.4.1.jar
hadoop-yarn-server-common-2.4.1.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
log4j-1.2.17.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.5.jar
hadoop-common-2.4.1.jar
```

B.1.5 HDFS 2.3.0

```
activation-1.1.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
guava-11.0.2.jar
hadoop-annotations-2.3.0.jar
hadoop-auth-2.3.0.jar
hadoop-client-2.3.0.jar
hadoop-common-2.3.0.jar
hadoop-hdfs-2.3.0.jar
hadoop-mapreduce-client-app-2.3.0.jar
hadoop-mapreduce-client-common-2.3.0.jar
hadoop-mapreduce-client-core-2.3.0.jar
hadoop-mapreduce-client-jobclient-2.3.0.jar
hadoop-mapreduce-client-shuffle-2.3.0.jar
hadoop-yarn-api-2.3.0.jar
hadoop-yarn-client-2.3.0.jar
hadoop-yarn-common-2.3.0.jar
hadoop-yarn-server-common-2.3.0.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jaxb-api-2.2.2.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
log4j-1.2.17.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.5.jar
```

B.1.6 HDFS 2.2.0

```
activation-1.1.jar
aopalliance-1.0.jar
asm-3.1.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
```

```
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.1.jar
commons-lang-2.5.jar
commons-logging-1.1.1.jar
commons-math-2.1.jar
commons-net-3.1.jar
gmbal-api-only-3.0.0-b023.jar
grizzly-framework-2.1.2.jar
grizzly-http-2.1.2.jar
grizzly-http-server-2.1.2.jar
grizzly-http-servlet-2.1.2.jar
grizzly-rcm-2.1.2.jar
guava-11.0.2.jar
guice-3.0.jar
hadoop-annotations-2.2.0.jar
hadoop-auth-2.2.0.jar
hadoop-client-2.2.0.jar
hadoop-common-2.2.0.jar
hadoop-hdfs-2.2.0.jar
hadoop-mapreduce-client-app-2.2.0.jar
hadoop-mapreduce-client-common-2.2.0.jar
hadoop-mapreduce-client-core-2.2.0.jar
hadoop-mapreduce-client-jobclient-2.2.0.jar
hadoop-mapreduce-client-shuffle-2.2.0.jar
hadoop-yarn-api-2.2.0.jar
hadoop-yarn-client-2.2.0.jar
hadoop-yarn-common-2.2.0.jar
hadoop-yarn-server-common-2.2.0.jar
jackson-core-asl-1.8.8.jar
jackson-jaxrs-1.8.3.jar
jackson-mapper-asl-1.8.8.jar
jackson-xc-1.8.3.jar
javax.inject-1.jar
javax.servlet-3.1.jar
javax.servlet-api-3.0.1.jar
jaxb-api-2.2.2.jar
jaxb-impl-2.2.3-1.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jersey-grizzly2-1.9.jar
jersey-guice-1.9.jar
jersey-json-1.9.jar
jersey-server-1.9.jar
jersey-test-framework-core-1.9.jar
jersey-test-framework-grizzly2-1.9.jar
jettison-1.1.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
log4j-1.2.17.jar
management-api-3.0.0-b012.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
```

```
snappy-java-1.0.4.1.jar
stax-api-1.0.1.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.5.jar
```

C Flume Handler Client Dependencies

This appendix lists the Flume client dependencies for Apache Flume.

Maven groupId: `org.apache.flume`

Maven atifactId: `hadoop-ng-skd`

Maven version: the Flume version numbers listed for each section

C.1 Flume Client Dependencies

This section lists the Flume client dependencies for each Flume version.

- Flume 1.6.0
- Flume 1.5.2
- "Flume 1.4.0"

C.1.1 Flume 1.6.0

```
avro-1.7.4.jar
avro-ipc-1.7.4.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-lang-2.5.jar
commons-logging-1.1.1.jar
flume-ng-sdk-1.6.0.jar
httpclient-4.1.3.jar
httpcore-4.1.3.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jetty-6.1.26.jar
jetty-util-6.1.26.jar
libthrift-0.9.0.jar
netty-3.5.12.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
snappy-java-1.0.4.1.jar
velocity-1.7.jar
xz-1.0.jar
```

C.1.2 Flume 1.5.2

```
avro-1.7.3.jar
avro-ipc-1.7.3.jar
commons-codec-1.3.jar
commons-collections-3.2.1.jar
commons-lang-2.5.jar
```

```
commons-logging-1.1.1.jar
flume-ng-sdk-1.5.2.jar
httpclient-4.0.1.jar
httpcore-4.0.1.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jetty-6.1.26.jar
jetty-util-6.1.26.jar
libthrift-0.7.0.jar
netty-3.5.12.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
snappy-java-1.0.4.1.jar
velocity-1.7.jar
```

C.1.3 Flume 1.4.0

```
avro-1.7.3.jar
avro-ipc-1.7.3.jar
commons-codec-1.3.jar
commons-collections-3.2.1.jar
commons-lang-2.5.jar
commons-logging-1.1.1.jar
flume-ng-sdk-1.4.0.jar
httpclient-4.0.1.jar
httpcore-4.0.1.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jetty-6.1.26.jar
jetty-util-6.1.26.jar
libthrift-0.7.0.jar
netty-3.4.0.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
snappy-java-1.0.4.1.jar
velocity-1.7.jar
```

D Kafka Handler Client Dependencies

This appendix lists the Kafka client dependencies for Apache Kafka.

Maven groupId: `org.apache.kafka`

Maven atifactId: `kafka-clients`

Maven version: the Kafka version numbers listed for each section

D.1 Kafka Client Dependencies

This section lists the Kafka client dependencies for each Kafka version.

- Kafka 0.8.2.1

D.1.1 Kafka 0.8.2.1

```
kafka-clients-0.8.2.1.jar
lz4-1.2.0.jar
slf4j-api-1.7.6.jar
snappy-java-1.1.1.6.jar
```