Oracle® Fusion Middleware

Using Oracle GoldenGate for Big Data

Release 12.3.2.1
E89845-03
September 2018

Oracle Fusion Middleware Using Oracle GoldenGate for Big Data, Release 12.3.2.1

E89845-03

Copyright © 2015, 2018, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience xviii
Documentation Accessibility xviii
Related Information xviii
Conventions xix

1 Introducing Oracle GoldenGate for Big Data

1.1 Understanding What's Supported 1-1
1.1.1 Verifying Certification and System Requirements 1-1
1.1.2 What are the Additional Support Considerations? 1-2
1.2 Setting Up Oracle GoldenGate for Big Data 1-4
1.2.1 About Oracle GoldenGate Properties Files 1-4
1.2.2 Setting Up the Java Runtime Environment 1-5
1.2.3 Configuring Java Virtual Machine Memory 1-5
1.2.4 Grouping Transactions 1-6
1.3 Configuring GoldenGate for Big Data 1-6
1.3.1 Running with Replicat 1-7
1.3.1.1 Configuring Replicat 1-7
1.3.1.2 Adding the Replicat Process 1-7
1.3.1.3 Replicat Grouping 1-8
1.3.1.4 About Replicat Checkpointing 1-8
1.3.1.5 About Initial Load Support 1-8
1.3.1.6 About the Unsupported Replicat Features 1-8
1.3.1.7 How the Mapping Functionality Works 1-8
1.3.2 Overview of Logging 1-8
1.3.2.1 About Replicat Process Logging 1-9
1.3.2.2 About Java Layer Logging 1-9
1.3.3 About Schema Evolution and Metadata Change Events 1-10
1.3.4 About Configuration Property CDATA[] Wrapping 1-10
1.3.5 Using Regular Expression Search and Replace 1-11
1.3.5.1 Using Schema Data Replace 1-11
1.3.5.2 Using Content Data Replace 1-12

1.3.6 Scaling Oracle GoldenGate for Big Data Delivery 1-13
1.3.7 Using Identities in Oracle GoldenGate Credential Store 1-15
1.3.7.1 Creating a Credential Store 1-16
1.3.7.2 Adding Users to a Credential Store 1-16
1.3.7.3 Configuring Properties to Access the Credential Store 1-17

2 Using the BigQuery Handler

2.1 Detailing the Functionality 2-1
2.1.1 Data Types 2-1
2.1.2 Operation Modes 2-1
2.1.3 Operation Processing Support 2-2
2.2 Setting Up and Running the BigQuery Handler 2-3
2.2.1 Understanding the BigQuery Handler Configuration 2-4
2.2.2 Review a Sample Configuration 2-6
2.2.3 Proxy Settings 2-6
2.2.4 Configuring Handler Authentication 2-6

3 Using the Cassandra Handler

3.1 Overview 3-1
3.2 Detailing the Functionality 3-1
3.2.1 About the Cassandra Data Types 3-2
3.2.2 About Catalog, Schema, Table, and Column Name Mapping 3-3
3.2.3 About DDL Functionality 3-3
3.2.3.1 About the Keyspaces 3-4
3.2.3.2 About the Tables 3-4
3.2.3.3 Addng Column Functionality 3-5
3.2.3.4 Dropping Column Functionality 3-5
3.2.4 How Operations are Processed 3-5
3.2.5 About Compressed Updates vs. Full Image Updates 3-6
3.2.6 About Primary Key Updates 3-7
3.3 Setting Up and Running the Cassandra Handler 3-7
3.3.1 Understanding the Cassandra Handler Configuration 3-8
3.3.2 Review a Sample Configuration 3-10
3.3.3 Configuring Security 3-11
3.4 About Automated DDL Handling 3-11
3.4.1 About the Table Check and Reconciliation Process 3-11
3.4.2 Capturing New Change Data 3-12
3.5 Performance Considerations 3-12
3.6 Additional Considerations 3-12

3.7 Troubleshooting 3-13
3.7.1 Java Classpath 3-14
3.7.2 Logging 3-14
3.7.3 Write Timeout Exception 3-14
3.7.4 Logging 3-15
3.7.5 Datastax Driver Error 3-15

4 Using the Elasticsearch Handler

4.1 Overview 4-1
4.2 Detailing the Functionality 4-1
4.2.1 About the Elasticsearch Version Property 4-2
4.2.2 About the Index and Type 4-2
4.2.3 About the Document 4-2
4.2.4 About the Primary Key Update 4-2
4.2.5 About the Data Types 4-3
4.2.6 Operation Mode 4-3
4.2.7 Operation Processing Support 4-3
4.2.8 About the Connection 4-3
4.3 Setting Up and Running the Elasticsearch Handler 4-4
4.3.1 Configuring the Elasticsearch Handler 4-4
4.3.2 About the Transport Client Settings Properties File 4-7
4.4 Performance Consideration 4-7
4.5 About the Shield Plug-In Support 4-8
4.6 About DDL Handling 4-8
4.7 Troubleshooting 4-8
4.7.1 Incorrect Java Classpath 4-8
4.7.2 Elasticsearch Version Mismatch 4-9
4.7.3 Transport Client Properties File Not Found 4-9
4.7.4 Cluster Connection Problem 4-9
4.7.5 Unsupported Truncate Operation 4-9
4.7.6 Bulk Execute Errors 4-10
4.8 Logging 4-10
4.9 Known Issues in the Elasticsearch Handler 4-11

5 Using the File Writer Handler

5.1 Overview 5-1
5.1.1 Detailing the Functionality 5-2
5.1.1.1 Using File Roll Events 5-2
5.1.1.2 Automatic Directory Creation 5-4

5.1.1.3 About the Active Write Suffix 5-4
5.1.1.4 Maintenance of State 5-4
5.1.1.5 Using Templated Strings 5-5
5.1.2 Configuring the File Writer Handler 5-6
5.1.3 Review a Sample Configuration 5-14
5.2 Using the HDFS Event Handler 5-15
5.2.1 Detailing the Functionality 5-15
5.2.1.1 Configuring the Handler 5-15
5.2.1.2 Using Templated Strings 5-16
5.2.1.3 Configuring the HDFS Event Handler 5-17
5.3 Using the Optimized Row Columnar Event Handler 5-19
5.3.1 Overview 5-19
5.3.2 Detailing the Functionality 5-19
5.3.2.1 About the Upstream Data Format 5-19
5.3.2.2 About the Library Dependencies 5-20
5.3.2.3 Requirements 5-20
5.3.2.4 Using Templated Strings 5-20
5.3.3 Configuring the ORC Event Handler 5-22
5.4 Using the Oracle Cloud Infrastructure Event Handler 5-24
5.4.1 Overview 5-24
5.4.2 Detailing the Functionality 5-24
5.4.3 Configuring the Oracle Cloud Infrastructure Event Handler 5-24
5.4.4 Configuring Credentials for Oracle Cloud Infrastructure 5-27
5.4.5 Using Templated Strings 5-28
5.4.6 Troubleshooting 5-30
5.5 Using the Oracle Cloud Infrastructure Classic Event Handler 5-30
5.5.1 Overview 5-31
5.5.2 Detailing the Functionality 5-31
5.5.3 Configuring the Oracle Cloud Infrastructure Classic Event Handler 5-31
5.5.4 Using Templated Strings 5-34
5.5.5 Troubleshooting 5-35
5.6 Using the Parquet Event Handler 5-36
5.6.1 Overview 5-36
5.6.2 Detailing the Functionality 5-36
5.6.2.1 Configuring the Parquet Event Handler to Write to HDFS 5-36
5.6.2.2 About the Upstream Data Format 5-37
5.6.2.3 Using Templated Strings 5-37
5.6.3 Configuring the Parquet Event Handler 5-38
5.7 Using the S3 Event Handler 5-41
5.7.1 Overview 5-41
5.7.2 Detailing Functionality 5-41

5.7.2.1 Configuring the Client ID and Secret 5-42
5.7.2.2 About the AWS S3 Buckets 5-42
5.7.2.3 Using Templated Strings 5-42
5.7.2.4 Troubleshooting 5-44
5.7.3 Configuring the S3 Event Handler 5-44

6 Using the Flume Handler

6.1 Overview 6-1
6.2 Setting Up and Running the Flume Handler 6-1
6.2.1 Classpath Configuration 6-2
6.2.2 Flume Handler Configuration 6-2
6.2.3 Review a Sample Configuration 6-3
6.3 Data Mapping of Operations to Flume Events 6-3
6.3.1 Operation Mode 6-4
6.3.2 Transaction Mode and EventMapsTo Operation 6-4
6.3.3 Transaction Mode and EventMapsTo Transaction 6-4
6.4 Performance Considerations 6-5
6.5 Metadata Change Events 6-5
6.6 Example Flume Source Configuration 6-5
6.6.1 Avro Flume Source 6-5
6.6.2 Thrift Flume Source 6-6
6.7 Advanced Features 6-6
6.7.1 Schema Propagation 6-6
6.7.2 Security 6-7
6.7.3 Fail Over Functionality 6-7
6.7.4 Load Balancing Functionality 6-7
6.8 Troubleshooting the Flume Handler 6-8
6.8.1 Java Classpath 6-8
6.8.2 Flume Flow Control Issues 6-8
6.8.3 Flume Agent Configuration File Not Found 6-8
6.8.4 Flume Connection Exception 6-9
6.8.5 Other Failures 6-9

7 Using the HBase Handler

7.1 Overview 7-1
7.2 Detailed Functionality 7-1
7.3 Setting Up and Running the HBase Handler 7-2
7.3.1 Classpath Configuration 7-2
7.3.2 HBase Handler Configuration 7-3

7.3.3 Sample Configuration 7-5
7.3.4 Performance Considerations 7-6
7.4 Security 7-6
7.5 Metadata Change Events 7-6
7.6 Additional Considerations 7-7
7.7 Troubleshooting the HBase Handler 7-7
7.7.1 Java Classpath 7-7
7.7.2 HBase Connection Properties 7-8
7.7.3 Logging of Handler Configuration 7-8
7.7.4 HBase Handler Delete-Insert Problem 7-8
7.7.5 Cloudera CDH HBase Compatibility 7-9

8 Using the HDFS Handler

8.1 Overview 8-1
8.2 Writing into HDFS in SequenceFile Format 8-1
8.2.1 Integrating with Hive 8-1
8.2.2 Understanding the Data Format 8-2
8.3 Setting Up and Running the HDFS Handler 8-2
8.3.1 Classpath Configuration 8-3
8.3.2 HDFS Handler Configuration 8-3
8.3.3 Review a Sample Configuration 8-9
8.3.4 Performance Considerations 8-10
8.3.5 Security 8-10
8.4 Writing in HDFS in Avro Object Container File Format 8-11
8.5 Generating HDFS File Names Using Template Strings 8-11
8.6 Metadata Change Events 8-12
8.7 Partitioning 8-13
8.8 HDFS Additional Considerations 8-13
8.9 Best Practices 8-14
8.10 Troubleshooting the HDFS Handler 8-15
8.10.1 Java Classpath 8-15
8.10.2 HDFS Connection Properties 8-15
8.10.3 Handler and Formatter Configuration 8-15

9 Using the Java Database Connectivity Handler

9.1 Overview 9-1
9.2 Detailed Functionality 9-1
9.2.1 Single Operation Mode 9-2
9.2.2 Oracle Database Data Types 9-2

9.2.3 MySQL Database Data Types 9-2
9.2.4 Netezza Database Data Types 9-3
9.2.5 Redshift Database Data Types 9-3
9.3 Setting Up and Running the JDBC Handler 9-3
9.3.1 Java Classpath 9-4
9.3.2 Handler Configuration 9-4
9.3.3 Statement Caching 9-5
9.3.4 Setting Up Error Handling 9-6
9.4 Sample Configurations 9-7
9.4.1 Sample Oracle Database Target 9-7
9.4.2 Sample Oracle Database Target with JDBC Metadata Provider 9-7
9.4.3 Sample MySQL Database Target 9-8
9.4.4 Sample MySQL Database Target with JDBC Metadata Provider 9-8

10 Using the Kafka Handler

10.1 Overview 10-1
10.2 Detailed Functionality 10-1
10.3 Setting Up and Running the Kafka Handler 10-3
10.3.1 Classpath Configuration 10-4
10.3.2 Kafka Handler Configuration 10-4
10.3.3 Java Adapter Properties File 10-6
10.3.4 Kafka Producer Configuration File 10-7
10.3.5 Using Templates to Resolve the Topic Name and Message Key 10-7
10.3.6 Kafka Configuring with Kerberos on a Hadoop Platform 10-9
10.4 Schema Propagation 10-13
10.5 Performance Considerations 10-13
10.6 About Security 10-14
10.7 Metadata Change Events 10-14
10.8 Snappy Considerations 10-15
10.9 Troubleshooting 10-15
10.9.1 Verify the Kafka Setup 10-15
10.9.2 Classpath Issues 10-15
10.9.3 Invalid Kafka Version 10-15
10.9.4 Kafka Producer Properties File Not Found 10-15
10.9.5 Kafka Connection Problem 10-16

11 Using the Kafka Connect Handler

11.1 Overview 11-1
11.2 Detailed Functionality 11-1

11.3 Setting Up and Running the Kafka Connect Handler 11-3
11.3.1 Kafka Connect Handler Configuration 11-4
11.3.2 Using Templates to Resolve the Topic Name and Message Key 11-9
11.3.3 Configuring Security in the Kafka Connect Handler 11-10
11.4 Kafka Connect Handler Performance Considerations 11-11
11.5 Troubleshooting the Kafka Connect Handler 11-11
11.5.1 Java Classpath for Kafka Connect Handler 11-12
11.5.2 Invalid Kafka Version 11-12
11.5.3 Kafka Producer Properties File Not Found 11-12
11.5.4 Kafka Connection Problem 11-12

12 Using the Kafka REST Proxy Handler

12.1 Overview 12-1
12.2 Setting Up and Starting the Kafka REST Proxy Handler Services 12-1
12.2.1 Using the Kafka REST Proxy Handler 12-2
12.2.2 Kafka REST Proxy Handler Configuration 12-2
12.2.3 Security 12-4
12.2.4 Generating a Keystore 12-5
12.2.5 Using Templates to Resolve the Topic Name and Message Key 12-5
12.2.6 Kafka REST Proxy Handler Formatter Properties 12-8
12.2.7 Setting Metacolumn Output 12-11
12.3 Consuming the Records 12-14
12.4 Performance Considerations 12-15
12.5 Kafka REST Proxy Handler Metacolumns Template Property 12-15

13 Using the Kinesis Streams Handler

13.1 Overview 13-1
13.2 Detailed Functionality 13-1
13.2.1 Amazon Kinesis Java SDK 13-1
13.2.2 Kinesis Streams Input Limits 13-2
13.3 Setting Up and Running the Kinesis Streams Handler 13-2
13.3.1 Set the Classpath in Kinesis Streams Handler 13-3
13.3.2 Kinesis Streams Handler Configuration 13-3
13.3.3 Using Templates to Resolve the Stream Name and Partition Name 13-8
13.3.4 Configuring the Client ID and Secret in Kinesis Handler 13-10
13.3.5 Configuring the Proxy Server for Kinesis Streams Handler 13-11
13.3.6 Configuring Security in Kinesis Streams Handler 13-11
13.4 Kinesis Handler Performance Considerations 13-12
13.4.1 Kinesis Streams Input Limitations 13-12

13.4.2 Transaction Batching 13-12
13.4.3 Deferring Flush at Transaction Commit 13-13
13.5 Troubleshooting 13-13
13.5.1 Java Classpath 13-13
13.5.2 Kinesis Handler Connectivity Issues 13-14
13.5.3 Logging 13-14

14 Using the MongoDB Handler

14.1 Overview 14-1
14.2 Detailed Functionality 14-1
14.2.1 Document Key Column 14-1
14.2.2 Primary Key Update Operation 14-2
14.2.3 MongoDB Trail Data Types 14-2
14.3 Setting Up and Running the MongoDB Handler 14-2
14.3.1 Classpath Configuration 14-3
14.3.2 MongoDB Handler Configuration 14-3
14.3.3 Connecting and Authenticating 14-5
14.3.4 Using Bulk Write 14-6
14.3.5 Using Write Concern 14-6
14.3.6 Using Three-Part Table Names 14-6
14.3.7 Using Undo Handling 14-7
14.4 Review a Sample Configuration 14-7

15 Using the Metadata Providers

15.1 About the Metadata Providers 15-1
15.2 Avro Metadata Provider 15-2
15.2.1 Detailed Functionality 15-2
15.2.2 Runtime Prerequisites 15-3
15.2.3 Classpath Configuration 15-4
15.2.4 Avro Metadata Provider Configuration 15-4
15.2.5 Review a Sample Configuration 15-4
15.2.6 Metadata Change Events 15-5
15.2.7 Limitations 15-6
15.2.8 Troubleshooting 15-6
15.2.8.1 Invalid Schema Files Location 15-6
15.2.8.2 Invalid Schema File Name 15-6
15.2.8.3 Invalid Namespace in Schema File 15-7
15.2.8.4 Invalid Table Name in Schema File 15-7
15.3 Java Database Connectivity Metadata Provider 15-7

15.3.1 JDBC Detailed Functionality 15-8
15.3.2 Java Classpath 15-8
15.3.3 JDBC Metadata Provider Configuration 15-9
15.3.4 Review a Sample Configuration 15-9
15.4 Hive Metadata Provider 15-10
15.4.1 Detailed Functionality 15-11
15.4.2 Configuring Hive with a Remote Metastore Database 15-12
15.4.3 Classpath Configuration 15-13
15.4.4 Hive Metadata Provider Configuration Properties 15-14
15.4.5 Review a Sample Configuration 15-15
15.4.6 Security 15-17
15.4.7 Metadata Change Event 15-17
15.4.8 Limitations 15-18
15.4.9 Additional Considerations 15-18
15.4.10 Troubleshooting 15-18

16 Using the Oracle NoSQL Handler

16.1 Overview 16-1
16.2 Detailed Functionality 16-1
16.2.1 Oracle NoSQL Data Types 16-2
16.2.2 Performance Considerations 16-2
16.2.3 Operation Processing Support 16-2
16.2.4 Column Processing 16-3
16.2.5 Table Check and Reconciliation Process 16-3
16.2.6 Security 16-4
16.3 Oracle NoSQL Handler Configuration 16-5
16.4 Review a Sample Configuration 16-7
16.5 Performance Considerations 16-8
16.6 Full Image Data Requirements 16-8

17 Using the Pluggable Formatters

17.1 Using the Avro Formatter 17-1
17.1.1 Avro Row Formatter 17-1
17.1.1.1 Operation Metadata Formatting Details 17-2
17.1.1.2 Operation Data Formatting Details 17-2
17.1.1.3 Sample Avro Row Messages 17-3
17.1.1.4 Avro Schemas 17-4
17.1.1.5 Avro Row Configuration Properties 17-5
17.1.1.6 Review a Sample Configuration 17-7

17.1.1.7 Metadata Change Events 17-8
17.1.1.8 Special Considerations 17-8
17.1.2 The Avro Operation Formatter 17-10
17.1.2.1 Operation Metadata Formatting Details 17-10
17.1.2.2 Operation Data Formatting Details 17-11
17.1.2.3 Sample Avro Operation Messages 17-11
17.1.2.4 Avro Schema 17-13
17.1.2.5 Avro Operation Formatter Configuration Properties 17-15
17.1.2.6 Review a Sample Configuration 17-16
17.1.2.7 Metadata Change Events 17-17
17.1.2.8 Special Considerations 17-17
17.1.3 Avro Object Container File Formatter 17-18
17.1.3.1 Avro OCF Formatter Configuration Properties 17-19
17.1.4 Setting Metacolumn Output 17-23
17.2 Using the Delimited Text Formatter 17-25
17.2.1 Message Formatting Details 17-25
17.2.2 Sample Formatted Messages 17-26
17.2.2.1 Sample Insert Message 17-26
17.2.2.2 Sample Update Message 17-26
17.2.2.3 Sample Delete Message 17-27
17.2.2.4 Sample Truncate Message 17-27
17.2.3 Output Format Summary Log 17-27
17.2.4 Delimited Text Formatter Configuration Properties 17-27
17.2.5 Review a Sample Configuration 17-29
17.2.6 Metadata Change Events 17-30
17.2.7 Setting Metacolumn Output 17-30
17.2.8 Additional Considerations 17-32
17.2.8.1 Primary Key Updates 17-33
17.2.8.2 Data Consolidation 17-34
17.3 Using the JSON Formatter 17-34
17.3.1 Operation Metadata Formatting Details 17-34
17.3.2 Operation Data Formatting Details 17-35
17.3.3 Row Data Formatting Details 17-36
17.3.4 Sample JSON Messages 17-36
17.3.4.1 Sample Operation Modeled JSON Messages 17-37
17.3.4.2 Sample Flattened Operation Modeled JSON Messages 17-38
17.3.4.3 Sample Row Modeled JSON Messages 17-39
17.3.4.4 Sample Primary Key Output JSON Message 17-40
17.3.5 JSON Schemas 17-40
17.3.6 JSON Formatter Configuration Properties 17-47
17.3.7 Review a Sample Configuration 17-49

17.3.8 Metadata Change Events 17-50
17.3.9 Setting Metacolumn Output 17-50
17.3.10 JSON Primary Key Updates 17-52
17.3.11 Integrating Oracle Stream Analytics 17-52
17.4 Using the Length Delimited Value Formatter 17-52
17.4.1 Formatting Message Details 17-53
17.4.2 Sample Formatted Messages 17-53
17.4.3 LDV Formatter Configuration Properties 17-54
17.4.4 Additional Considerations 17-57
17.5 Using Operation-Based versus Row-Based Formatting 17-57
17.5.1 Operation Formatters 17-58
17.5.2 Row Formatters 17-58
17.5.3 Table Row or Column Value States 17-58
17.6 Using the XML Formatter 17-59
17.6.1 Message Formatting Details 17-59
17.6.2 Sample XML Messages 17-60
17.6.2.1 Sample Insert Message 17-60
17.6.2.2 Sample Update Message 17-61
17.6.2.3 Sample Delete Message 17-61
17.6.2.4 Sample Truncate Message 17-62
17.6.3 XML Schema 17-63
17.6.4 XML Formatter Configuration Properties 17-64
17.6.5 Review a Sample Configuration 17-65
17.6.6 Metadata Change Events 17-65
17.6.7 Setting Metacolumn Output 17-65
17.6.8 Primary Key Updates 17-67

18 Using Oracle GoldenGate Capture for Cassandra

18.1 Overview 18-1
18.2 Setting Up Cassandra Change Data Capture 18-2
18.2.1 Data Types 18-2
18.2.2 Cassandra Database Operations 18-3
18.3 Deduplication 18-3
18.4 Topology Changes 18-4
18.5 Data Availability in the CDC Logs 18-4
18.6 Using Extract Initial Load 18-4
18.7 Using Change Data Capture Extract 18-5
18.8 Replicating to RDMBS Targets 18-7
18.9 Partition Update or Insert of Static Columns 18-7
18.10 Partition Delete 18-8

18.11 Security and Authentication 18-8
18.11.1 Configuring SSL 18-8
18.12 Multiple Extract Support 18-9
18.13 CDC Configuration Reference 18-9
18.14 Troubleshooting 18-15

19 Connecting to Microsoft Azure Data Lake

A Cassandra Handler Client Dependencies

A.1 Cassandra Datastax Java Driver 3.1.0 A-1

B Cassandra Capture Client Dependencies

C Elasticsearch Handler Client Dependencies

C.1 Elasticsearch 2.4.4 and Shield Plugin 2.2.2 C-1
C.2 Elasticsearch 5.1.2 with X-Pack 5.1.2 C-2

D Flume Handler Client Dependencies

D.1 Flume 1.7.0 D-1
D.2 Flume 1.6.0 D-1
D.3 Flume 1.5.2 D-2
D.4 Flume 1.4.0 D-2

E HBase Handler Client Dependencies

E.1 HBase 1.2.5 E-1
E.2 HBase 1.1.1 E-2
E.3 HBase 1.0.1.1 E-3

F HDFS Handler Client Dependencies

F.1 Hadoop Client Dependencies F-1
F.1.1 HDFS 2.8.0 F-1
F.1.2 HDFS 2.7.1 F-2
F.1.3 HDFS 2.6.0 F-4
F.1.4 HDFS 2.5.2 F-5

F.1.5 HDFS 2.4.1 F-6
F.1.6 HDFS 2.3.0 F-7
F.1.7 HDFS 2.2.0 F-8

G Kafka Handler Client Dependencies

G.1 Kafka 1.1.0 G-1
G.2 Kafka 1.0.0 G-1
G.3 Kafka 0.11.0.0 G-1
G.4 Kafka 0.10.2.0 G-2
G.5 Kafka 0.10.1.1 G-2
G.6 Kafka 0.10.0.1 G-2
G.7 Kafka 0.9.0.1 G-2

H Kafka Connect Handler Client Dependencies

H.1 Kafka 0.11.0.0 H-1
H.2 Kafka 0.10.2.0 H-2
H.3 Kafka 0.10.2.0 H-2
H.4 Kafka 0.10.0.0 H-2
H.5 Kafka 0.9.0.1 H-3
H.6 Confluent 4.1.2 H-3
H.7 Confluent 4.0.0 H-4
H.8 Confluent 3.2.1 H-4
H.9 Confluent 3.2.0 H-5
H.10 Confluent 3.2.1 H-5
H.11 Confluent 3.1.1 H-5
H.12 Confluent 3.0.1 H-6
H.13 Confluent 2.0.1 H-6
H.14 Confluent 2.0.1 H-7

I MongoDB Handler Client Dependencies

I.1 MongoDB Java Driver 3.4.3 I-1

J Optimized Row Columnar Event Handler Client Dependencies

J.1 ORC Client Dependencies J-1

K Parquet Event Handler Client Dependencies

K.1 Parquet Client Dependencies K-1

Preface

This guide contains information about configuring, and running Oracle GoldenGate for Big Data to extend the capabilities of Oracle GoldenGate instances.

- Audience (page xviii)
- Documentation Accessibility (page xviii)
- Related Information (page xviii)
- Conventions (page xix)

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`.

Accessible Access to Oracle Support

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

Related Information

The Oracle GoldenGate Product Documentation Libraries are found at

Oracle GoldenGate

Oracle GoldenGate Application Adapters

Oracle GoldenGate for Big Data

Oracle GoldenGate Plug-in for EMCC

Oracle GoldenGate Monitor

Oracle GoldenGate for HP NonStop (Guardian)

Oracle GoldenGate Veridata

Oracle GoldenGate Studio

Additional Oracle GoldenGate information, including best practices, articles, and solutions, is found at:

Oracle GoldenGate A-Team Chronicles

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

Introducing Oracle GoldenGate for Big Data

Learn about Oracle GoldenGate for Big Data concepts and features, including how to setup and configure the environment.
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 description in Oracle GoldenGate Java Delivery.

Topics:

- Understanding What's Supported (page 1-1)
- Setting Up Oracle GoldenGate for Big Data (page 1-4)
- Configuring GoldenGate for Big Data (page 1-6)

1.1 Understanding What's Supported

Oracle GoldenGate for Big Data supports specific configurations: the handlers, which are compatible with clearly defined software versions, and there are many support topics. This section provides the relevant support information.

Topics:

- Verifying Certification and System Requirements (page 1-1)
- What are the Additional Support Considerations? (page 1-2)

1.1.1 Verifying Certification and System Requirements

Make sure that you install your product on a supported hardware or software configuration. For more information, see the certification document for your release on the *Oracle Fusion Middleware Supported System Configurations* page.

Oracle has tested and verified the performance of your product on all certified systems and environments; whenever new certifications occur, they are added to the proper certification document right away. New certifications can occur at any time, and for this reason the certification documents are kept outside of the documentation libraries on the Oracle Technology Network.

The compatibility of the Oracle GoldenGate for Big Data Handlers with the various data collections, including distributions, database releases, and drivers is included in the certification document.

1.1.2 What are the Additional Support Considerations?

This section describes additional Oracle GoldenGate for Big Data Handlers additional support considerations.

Pluggable Formatters—Support
The handlers support the Pluggable Formatters as described in Using the Pluggable Formatters (page 17-1) as follows:

- The HDFS Handler supports all of the pluggable handlers .
- Pluggable formatters are not applicable to the HBase Handler. Data is streamed to HBase using the proprietary HBase client interface.
- The Flume Handler supports all of the pluggable handlers described in Using the Pluggable Formatters (page 17-1).
- The Kafka Handler supports all of the pluggable handlers described in Using the Pluggable Formatters (page 17-1).
- The Kafka Connect Handler does *not* support pluggable formatters. You can convert data to JSON or Avro using Kafka Connect data converters.
- The Kinesis Streams Handler supports all of the pluggable handlers described in Using the Pluggable Formatters (page 17-1).
- The Cassandra, MongoDB, and JDBC Handlers do *not* use a pluggable formatter.

Avro Formatter—Improved Support for Binary Source Data
In previous releases, the Avro Formatter did not support the Avro bytes data type. Binary data was instead converted to Base64 and persisted in Avro messages as a field with a string data type. This required an additional conversion step to convert the data from Base64 back to binary.
The Avro Formatter now can identify binary source fields that will be mapped into an Avro bytes field and the original byte stream from the source trail file will be propagated to the corresponding Avro messages without conversion to Base64.

Avro Formatter—Generic Wrapper
The `schema_hash` field was changed to the `schema_fingerprint` field. The `schema_fingerprint` is a long and is generated using the `parsingFingerprint64(Schemas)` method on the `org.apache.avro.SchemaNormalization` class. This identifier provides better traceability from the Generic Wrapper Message back to the Avro schema that is used to generate the Avro payload message contained in the Generic Wrapper Message.

JSON Formatter—Row Modeled Data
The JSON formatter supports row modeled data in addition to operation modeled data.. Row modeled data includes the after image data for insert operations, the after image data for update operations, the before image data for delete operations, and special handling for primary key updates.

Java Delivery Using Extract
Java Delivery using Extract is *not* supported and was deprecated in this release. Support for Java Delivery is only supported using the Replicat process. Replicat provides better performance, better support for checkpointing, and better control of transaction grouping.

Kafka Handler—Versions
Support for Kafka versions 0.8.2.2, 0.8.2.1, and 0.8.2.0 was discontinued. This allowed the implementation of the flush call on the Kafka producer, which provides better support for flow control and checkpointing.

HDFS Handler—File Creation
A new feature was added to the HDFS Handler so that you can use Extract, Load, Transform (ELT). The new `gg.handler.name.openNextFileAtRoll=true` property was added to create new files immediately when the previous file is closed. The new file appears in the HDFS directory immediately after the previous file stream is closed. This feature does not work when writing HDFS files in Avro Object Container File (OCF) format or sequence file format.

MongoDB Handler—Support

- The handler can only replicate unique rows from source table. If a source table has no primary key defined and has duplicate rows, replicating the duplicate rows to the MongoDB target results in a duplicate key error and the Replicat process abends.
- Missed updates and deletes are undetected so are ignored.
- Untested with sharded collections.
- Only supports date and time data types with millisecond precision. These values from a trail with microseconds or nanoseconds precision are truncated to millisecond precision.
- The `datetime` data type with `timezone` in the trail is not supported.
- A maximum BSON document size of 16 MB. If the trail record size exceeds this limit, the handler cannot replicate the record.
- No DDL propagation.
- No truncate operation.

JDBC Handler—Support

- The JDBC handler uses the generic JDBC API, which means any target database with a JDBC driver implementation should be able to use this handler. There are a myriad of different databases that support the JDBC API and Oracle cannot certify the JDBC Handler for all targets. Oracle has certified the JDBC Handler for the following RDBMS targets:

  Oracle
  MySQL
  Netezza
  Redshift
  Greenplum

- The handler supports Replicat using the `REPERROR` and `HANDLECOLLISIONS` parameters, see *Reference for Oracle GoldenGate*.
- The database metadata retrieved through the Redshift JDBC driver has known constraints, see *Release Notes for Oracle GoldenGate for Big Data*.

  Redshift target table names in the Replicat parameter file must be in lower case and double quoted. For example:

```
MAP SourceSchema.SourceTable, target "public"."targetable";
```

- DDL operations are ignored by default and are logged with a `WARN` level.
- Coordinated Replicat is a multithreaded process that applies transactions in parallel instead of serially. Each thread handles all of the filtering, mapping, conversion, SQL construction, and error handling for its assigned workload. A coordinator thread coordinates transactions across threads to account for dependencies. It ensures that DML is applied in a synchronized manner preventing certain DMLs from occurring on the same object at the same time due to row locking, block locking, or table locking issues based on database specific rules. If there are database locking issue, then Coordinated Replicat performance can be extremely slow or pauses.

Delimited Formatter—Limitation
Handlers configured to generate delimited formatter output only allows single character delimiter fields. If your delimiter field length is greater than one character, then the handler displays an error message similar to the following and Replicat abends.

```
oracle.goldengate.util.ConfigException: Delimiter length cannot be more than one
character. Found delimiter [||]
```

DDL Event Handling
Only the `TRUNCATE TABLE` DDL statement is supported. All other DDL statements are ignored.
You can use the `TRUNCATE` statements one of these ways:

- In a DDL statement, `TRUNCATE TABLE`, `ALTER TABLE TRUNCATE PARTITION`, and other DDL `TRUNCATE` statements. This uses the `DDL` parameter.
- Standalone `TRUNCATE` support, which just has `TRUNCATE TABLE`. This uses the `GETTRUNCATES` parameter.

1.2 Setting Up Oracle GoldenGate for Big Data

The various tasks that you need to preform to set up Oracle GoldenGate for Big Data integrations with Big Data targets.

Topics:

- About Oracle GoldenGate Properties Files (page 1-4)
- Setting Up the Java Runtime Environment (page 1-5)
- Configuring Java Virtual Machine Memory (page 1-5)
- Grouping Transactions (page 1-6)

1.2.1 About Oracle GoldenGate 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. A Replicat properties file is required in order to run either process. The required naming convention for the Replicat file name is the *`process_name`*`.prm`. The exit syntax in the Replicat properties file provides the name and location of the Java Adapter properties file. It is the Java Adapter properties file that contains the configuration properties for the Java adapter include GoldenGate for Big

Data integrations. The Replicat and Java Adapters 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 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

1.2.2 Setting Up the Java Runtime Environment

The Oracle GoldenGate for Big Data integrations create an instance of the Java virtual machine at runtime. Oracle GoldenGate for Big Data requires that you install Oracle Java 8 Java Runtime Environment (JRE) at a minimum.

Oracle recommends that you set the JAVA_HOME environment variable to point to Java 8 installation directory. Additionally, the Java Delivery process needs to load the libjvm.so and libjsig.so 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).

1.2.3 Configuring Java Virtual Machine Memory

One of the difficulties of tuning Oracle GoldenGate for Big Data is deciding how much Java virtual machine (JVM) heap memory to allocate for the Replicat process hosting the Java Adapter. The JVM memory must be configured before starting the application. Otherwise, the default Java heap sizing is used. Specifying the JVM heap size correctly sized is important because if you size it to small, the JVM heap can cause runtime issues:

- A Java Out of Memory exception, which causes the Extract or Replicat process to abend.
- Increased frequency of Java garbage collections, which degrades performance. Java garbage collection invocations de-allocate all unreferenced Java objects resulting in reclaiming the heap memory for reuse.

Alternatively, too much heap memory is inefficient. The JVM reserves the maximum heap memory (-Xmx) when the JVM is launched. This reserved memory is generally not available to other applications even if the JVM is not using all of it. You can set the JVM memory with these two parameters:

- -Xmx — The maximum JVM heap size. This amount gets reserved.
- -Xms — The initial JVM heap size. Also controls the sizing of additional allocations.

The -Xmx and –Xms properties are set in the Java Adapter properties file as follows:

```
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=ggjava/ggjava.jar
```

There are no rules or equations for calculating the values of the maximum and initial JVM heap sizes. Java heap usage is variable and depends upon a number of factors many of which are widely variable at runtime. The Oracle GoldenGate Java Adapter

log file provides metrics on the Java heap when the status call is invoked. The information appears in the Java Adapter `log4j` log file similar to:

```
INFO 2017-12-21 10:02:02,037 [pool-1-thread-1] Memory at Status : Max: 455.00 MB,
Total: 58.00 MB, Free: 47.98 MB, Used: 10.02 MB
```

You can interpret these values as follows:

- `Max` – The value of heap memory reserved (typically the `-Xmx` setting reduced by approximately 10% due to overhead).
- `Total` – The amount currently allocated (typically a multiple of the `-Xms` setting reduced by approximately 10% due to overhead).
- `Free` – The heap memory currently allocated, but free to be used to allocate Java objects.
- `Used` – The heap memory currently allocated to Java objects.

You can control the frequency that the status is logged using the `gg.report.time=30sec` configuration parameter in the Java Adapter properties file.

You should execute test runs of the process with actual data and review the heap usage logging. Then analyze your peak memory usage and then allocate 25% - 30% more memory to accommodate infrequent spikes in memory use and to make the memory allocation and garbage collection processes efficient.

The following items can increase the heap memory required by the Replicat process:

- Operating in `tx` mod (For example, `gg.handler.name.mode=tx`.)
- Setting the Replicat property `GROUPTRANSOPS` to a large value
- Wide tables
- CLOB or BLOB data in the source
- Very large transactions in the source data

1.2.4 Grouping Transactions

The principal way to improve performance in Oracle GoldenGate for Big Data integrations is using 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 the Replicat process discussed in Running with Replicat (page 1-7).

1.3 Configuring GoldenGate for Big Data

This section describes how to configure GoldenGate for Big Data Handlers.

Topics:

- Running with Replicat (page 1-7)
- Overview of Logging (page 1-8)
- About Schema Evolution and Metadata Change Events (page 1-10)
- About Configuration Property CDATA[] Wrapping (page 1-10)

- Using Regular Expression Search and Replace (page 1-11)
- Scaling Oracle GoldenGate for Big Data Delivery (page 1-13)
- Using Identities in Oracle GoldenGate Credential Store (page 1-15)

1.3.1 Running with Replicat

This section explains how to run the Java Adapter with the Oracle GoldenGate Replicat process. It includes the following sections:

Topics:

- Configuring Replicat (page 1-7)
- Adding the Replicat Process (page 1-7)
- Replicat Grouping (page 1-8)
- About Replicat Checkpointing (page 1-8)
- About Initial Load Support (page 1-8)
- About the Unsupported Replicat Features (page 1-8)
- How the Mapping Functionality Works (page 1-8)

1.3.1.1 Configuring Replicat

The following is an example of how you can configure a Replicat process properties file for use with the 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 these Replicat configuration entries:

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

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

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

`GROUPTRANSOPS 1000` - Groups 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` - Sets the tables to exclude.

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

1.3.1.2 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
```

1.3.1.3 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`.

1.3.1.4 About Replicat Checkpointing

In addition to the Replicat checkpoint file `,.cpr`, an additional checkpoint file, `dirchk/group.cpj`, is created that contains information similar to `CHECKPOINTTABLE` in Replicat for the database.

1.3.1.5 About Initial Load Support

Replicat can already read trail files that come from both the online capture and initial load processes that write to a set of trail files. In addition, Replicat can also be configured to support the delivery of the special run initial load process using `RMTTASK` specification in the Extract parameter file. For more details about configuring the direct load, see Loading Data with an Oracle GoldenGate Direct Load.

1.3.1.6 About the Unsupported Replicat Features

The following Replicat features are not supported in this release:

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

1.3.1.7 How the Mapping Functionality Works

The Oracle GoldenGate Replicat process supports mapping functionality to custom target schemas. 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. For more information about the Replicat mapping syntax in the Replication configuration file, see Mapping and Manipulating Data.

1.3.2 Overview of 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.

Topics:

- About Replicat Process Logging (page 1-9)

- About Java Layer Logging (page 1-9)

1.3.2.1 About Replicat Process Logging

Oracle GoldenGate for Big Data integrations leverage the Java Delivery functionality described in the Delivering Java Messages. In this setup, either a Oracle GoldenGate Replicat 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:

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 you have 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 process creates or appends log files into the *GoldenGate_Home/*`dirrpt` directory that adheres to the following naming convention: *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 is critical for integrations with Big Data applications.

1.3.2.2 About 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, *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. These 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
```

You can modify the `bootoptions`in any of the files as follows:

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

You can use your 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.

1.3.3 About Schema Evolution and Metadata Change Events

The Metadata in trail is a feature that allows seamless runtime handling of metadata change events by Oracle GoldenGate for Big Data, including schema evolution and schema propagation to Big Data target applications. The `NO_OBJECTDEFS` is a sub-parameter of the Extract and Replicat `EXTTRAIL` and `RMTTRAIL` parameters that lets you suppress the important metadata in trail feature and revert to using a static metadata definition.

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 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.

1.3.4 About Configuration Property CDATA[] Wrapping

The GoldenGate for 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 trims 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. Alternatively, you can wrap white space values inside of special syntax to preserve the whites pace for selected configuration variables. GoldenGate 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 property with the `CDATA[]` wrapping preserves the white space and the line delimiter will then be a new-line character.

1.3.5 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 of **The Single UNIX Specification, Version 4**.

Topics:

- Using Schema Data Replace (page 1-11)
- Using Content Data Replace (page 1-12)

1.3.5.1 Using Schema Data Replace

You can replace schema data using the `gg.schemareplaceregex` and `gg.schemareplacestring` properties. 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 properties 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 supports using multiple search regular expressions and replacements strings using the following configuration syntax:

```
gg.schemareplaceregex=some_regex
gg.schemareplacestring=some_value
gg.schemareplaceregex1=some_regex
gg.schemareplacestring1=some_value
gg.schemareplaceregex2=some_regex
gg.schemareplacestring2=some_value
```

1.3.5.2 Using Content Data Replace

You can replace content data using the `gg.contentreplaceregex` and `gg.contentreplacestring` properties 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.contentreplaceregex=some_regex
gg.contentreplacestring=some_value
gg.contentreplaceregex1=some_regex
gg.contentreplacestring1=some_value
gg.contentreplaceregex2=some_regex
gg.contentreplacestring2=some_value
```

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

> **NOT_SUPPORTED:**
>
> 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 property 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[]
```

1.3.6 Scaling Oracle GoldenGate for Big Data Delivery

Oracle GoldenGate for Big Data supports breaking down the source trail files into either multiple Replicat processes or by using Coordinated Delivery to instantiate multiple Java Adapter instances inside a single Replicat process to improve throughput.. This allows you to scale Oracle GoldenGate for Big Data delivery.

There are some cases where the throughput to Oracle GoldenGate for Big Data integration targets is not sufficient to meet your service level agreements even after you have tuned your Handler for maximum performance. When this occurs, you can configure parallel processing and delivery to your targets using one of the following methods:

- Multiple Replicat processes can be configured to read data from the same source trail files. Each of these Replicat processes are configured to process a subset of the data in the source trail files so that all of the processes collectively process the source trail files in their entirety. There is no coordination between the separate Replicat processes using this solution.
- Oracle GoldenGate Coordinated Delivery can be used to parallelize processing the data from the source trail files within a single Replicat process. This solution involves breaking the trail files down into logical subsets for which each configured subset is processed by a different delivery thread. For more information about Coordinated Delivery, see https://blogs.oracle.com/dataintegration/entry/goldengate_12c_coordinated_replicat.

With either method, you can split the data into parallel processing for improved throughput. Oracle recommends breaking the data down in one of the following two ways:

- Splitting Source Data By Source Table –Data is divided into subsections by source table. For example, Replicat process 1 might handle source tables table1 and table2, while Replicat process 2 might handle data for source tables table3 and table2. Data is split for source table and the individual table data is not subdivided.
- Splitting Source Table Data into Sub Streams – Data from source tables is split. For example, Replicat process 1 might handle half of the range of data from source table1, while Replicat process 2 might handler the other half of the data from source table1.

Additional limitations:

- Parallel apply is *not* supported.
- The `BATCHSQL` parameter not supported.

Example 1-1 Scaling Support for the Oracle GoldenGate for Big Data Handlers

Handler Name	Splitting Source Data By Source Table	Splitting Source Table Data into Sub Streams
Cassandra	Supported	Supported when: • Required target tables in Cassandra are pre-created. • Metadata change events do not occur.
Elastic Search	Supported	Supported
Flume	Supported	Supported for formats that support schema propagation, such as Avro. This is less desirable due to multiple instances feeding the same schema information to the target.
HBase	Supported when all required HBase namespaces are pre-created in HBase.	Supported when: • All required HBase namespaces are pre-created in HBase. • All required HBase target tables are pre-created in HBase. Schema evolution is not an issue because HBase tables have no schema definitions so a source metadata change does not require any schema change in HBase. • The source data does not contain any truncate operations.

Handler Name	Splitting Source Data By Source Table	Splitting Source Table Data into Sub Streams
HDFS	Supported	Supported with some restrictions. • You must select a naming convention for generated HDFS files wherethe file names do not collide. Colliding HDFS file names results in a Replicat abend. When using coordinated apply it is suggested that you configure `${groupName}` as part of the configuration for the `gg.handler.name.fileNameMappingTemplate` property . The `${groupName}` template resolves to the Replicat name concatenated with the Replicat thread number, which provides unique naming per Replicat thread. • Schema propagatation to HDFS and Hive integration is *not* currently supported.
JDBC	Supported	Supported
Kafka	Supported	Supported for formats that support schema propagation, such as Avro. This is less desirable due to multiple instances feeding the same schema information to the target.
Kafka Connect	Supported	Supported
Kinesis Streams	Supported	Supported
MongoDB	Supported	Supported

1.3.7 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 and password pair in the credential store. The credential store is implemented as an auto login wallet within the Oracle Credential Store Framework (CSF). The use of an LDAP directory is not supported for the Oracle GoldenGate credential store. The auto login 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.

Topics:

- Creating a Credential Store (page 1-16)
- Adding Users to a Credential Store (page 1-16)
- Configuring Properties to Access the Credential Store (page 1-17)

1.3.7.1 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*.

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

1.3.7.2 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*.

1.3.7.3 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 JDBC the user name and password is encrypted.

Oracle Wallet integration only works for configuration properties which contain the string username or password. For example:

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

Consider the user name and password entries as accessible values in the Credential Store. Any configuration property resolved in the Java Adapter layer (not accessed in the C user exit layer) can be resolved from the Credential Store. This allows you more flexibility to be creative in how you protect sensitive configuration entries.

2
Using the BigQuery Handler

Learn how to use the Google BigQuery Handler, which streams change data capture data from source trail files into Google BigQuery.

BigQuery is a RESTful web service that enables interactive analysis of massively large datasets working in conjunction with Google Storage, see https://cloud.google.com/bigquery/.

Topics:

- Detailing the Functionality (page 2-1)
- Setting Up and Running the BigQuery Handler (page 2-3)

2.1 Detailing the Functionality

Topics:

- Data Types (page 2-1)
- Operation Modes (page 2-1)
- Operation Processing Support (page 2-2)

2.1.1 Data Types

The BigQuery Handler supports the standard SQL data types and most of these data types are supported by the BigQuery Handler. A data type conversion from the column value in the trail file to the corresponding Java type representing the BigQuery column type in the BigQuery Handler is required.

The following data types are supported:

```
STRING
BYTES
INTEGER
FLOAT
NUMERIC
BOOLEAN
TIMESTAMP
DATE
TIME
DATETIME
```

The BigQuery Handler does not support complex data types, such as `ARRAY` and `STRUCT`.

2.1.2 Operation Modes

You can configure the BigQuery Handler in one of these two modes:

`auditLogMode = true`
When the handler is configured to run with Audit log mode, the data is pushed into Google BigQuery without a unique id and primary key. As a result, Google BigQuery is not able to merge different operations on the same row.
Also, the order in which the audit log is displayed in the BigQuery data set is not deterministic.
To overcome these limitations, you need to specify `optype` and `position` in the meta columns template for the handler. This adds two columns of the same names in the schema for the table in Google BigQuery. For example:

```
gg.handler.bigquery.metaColumnsTemplate = ${optype}, ${position}
```

The `optype` is important to determine the operation type for the row in the audit log.
To view the audit log in order of the operations processed in the trail file, specify position which can be used in the `ORDER BY` clause while querying the table in Google BigQuery. For example:

```
SELECT * FROM [projectId:datasetId.tableId] ORDER BY position
```

`auditLogMode = false`
This causes the handler to write data into Google BigQuery specifying a unique id and primary key for each row. As a result, Google BigQuery is able to merge different operations on the same row.
The trail source needs to have a full image of the records in order to merge correctly.
Google BigQuery processes every operation as an `insert` for each row. As a result, there is a `deleted` column added to the schema for the table in this mode of operation.
When the handler encounters a delete operation on a row, it inserts the row into Google BigQuery and sets the deleted column to `true`.
To view data in the BigQuery table like it would ideally be seen in a RDBMS, specify a `WHERE deleted = false` clause while querying the table in Google BigQuery.

2.1.3 Operation Processing Support

The BigQuery Handler pushes operations to Google BigQuery using synchronous API. Insert, update, and delete operations are processed differently in BigQuery than in a traditional RDBMS.

The following explains how insert, update, and delete operations are interpreted by the handler depending on the mode of operation:

`auditLogMode = true`

- `insert` – Inserts the record with `optype` as an insert operation in the BigQuery table.
- `update` –Inserts the record with `optype` as an update operation in the BigQuery table.
- `delete` – Inserts the record with `optype` as a delete operation in the BigQuery table.
- `pkUpdate`—When `pkUpdateHandling` property is configured as `delete-insert`, the handler sends out a delete operation followed by an insert operation. Both these rows have the same position in the BigQuery table, which helps to identify it as a primary key operation and not a separate delete and insert operation.

auditLogMode = false

- `insert` – If the row does not already exist in Google BigQuery, then an insert operation is processed as an `insert`. If the row already exists in Google BigQuery, then an insert operation is processed as an `update`. The handler sets the deleted column to `false`.
- `update` –If a row does not exist in Google BigQuery, then an update operation is processed as an `insert`. If the row already exists in Google BigQuery, then an update operation is processed as `update`. The handler sets the deleted column to `false`.
- `delete` – If the row does not exist in Google BigQuery, then a delete operation has no effect. If the row exists in Google BigQuery, then a delete operation is processed as a `delete`. The handler sets the deleted column to `true`.
- `pkUpdate`—When `pkUpdateHandling` property is configured as `delete-insert`, the handler sets the deleted column to `true` for the row whose primary key is updated. It is followed by a separate insert operation with the new primary key and the deleted column set to `false` for this row.

Do not toggle the audit log mode because it forces the BigQuery handler to abend as Google BigQuery cannot alter schema of an existing table. The existing table needs to be deleted before switching audit log modes.

> **Note:**
>
> The BigQuery Handler does not support the `truncate` operation. It abends when it encounters a `truncate` operation.

2.2 Setting Up and Running the BigQuery Handler

The BigQuery Client library does not ship with Oracle GoldenGate for Big Data. You must download the latest version of the Java Client library for BigQuery at:

https://developers.google.com/api-client-library/java/apis/bigquery/v2

You must configure the `gg.classpath` configuration property in the Java Adapter properties file to specify the JARs for the Java Client Library for BigQuery. The path to the dependency JARs must include the asterisk (`*`) wildcard character to include all of the JAR files in that directory in the associated classpath. Do*not* use `*.jar`. This is an example of the correctly configured classpath:

```
gg.classpath= /path_to_repository/bigquery/libs/*:/path_to_repository/bigquery/*
```

Next, download the following JARs from Maven Central, and then include them in the classpath for the BigQuery Handler:

- `api-common-1.6.0.jar`
- `gax-1.28.0.jar`
- `gax-httpjson-0.45.0.jar`
- `google-auth-library-credentials-0.9.1.jar`
- `google-auth-library-oauth2-http-0.9.1.jar`
- `google-cloud-bigquery-1.31.0.jar`
- `google-cloud-core-1.35.0.jar`
- `google-cloud-core-http-1.35.0.jar`

- `google-http-client-jackson-1.23.0.jar`
- `guava-25.1-jre.jar`
- `threetenbp-1.3.6.jar`

- Understanding the BigQuery Handler Configuration (page 2-4)
- Review a Sample Configuration (page 2-6)
- Proxy Settings (page 2-6)
- Configuring Handler Authentication (page 2-6)

2.2.1 Understanding the BigQuery Handler Configuration

The following are the configurable values for the BigQuery Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the BigQuery Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=bigquery` and the other BigQuery properties as follows:

Table 2-1 BigQuery Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handlerlist`	Required	Any string	None	Provides a name for the BigQuery Handler. The BigQuery Handler name then becomes part of the property names listed in this table.
`gg.handler.`*`name`*`.type=bigquery`	Required	`bigquery`	None	Selects the BigQuery Handler for streaming change data capture into Google BigQuery.
`gg.handler.`*`name`*`.credentialsFile`	Optional	Relative or absolute path to the credentials file	None	The credentials file downloaded from Google BigQuery for authentication. If you do not specify the path to the credentials file, you need to set it as an environment variable, see Configuring Handler Authentication (page 2-6).
`gg.handler.name.projectId`	Required	Any string	None	The name of the project in Google BigQuery. The handler needs project ID to connect to Google BigQuery store.
`gg.handler.`*`name`*`.datasetId`	Optional	Any string	`default_dataset`	The name of the data set the tables are stored in. If not specified, the handler creates a new data set named `default_dataset` and inserts the table into it.
`gg.handler.`*`name`*`.batchSize`	Optional	Any number	`500`	The maximum number of operations to be batched together. This is applicable for all target table batches.
`gg.handler.`*`name`*`.batchFlushFrequency`	Optional	Any number	`1000`	The maximum amount of time in milliseconds to wait before executing the next batch of operations. This is applicable for all target table batches.

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

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.skipInvalidRows`	Optional	`true` | `false`	`false`	Sets whether to insert all valid rows of a request, even if invalid rows exist. If not set, the entire insert request fails if it contains an invalid row.
`gg.handler.name.ignoreUnknownValues`	Optional	`true` | `false`	`false`	Sets whether to accept rows that contain values that do not match the schema. If not set, rows with unknown values are considered to be invalid.
`gg.handler.name.connectionTimeout`	Optional	Positive integer	`20000`	The maximum amount of time, in milliseconds, to wait for the handler to establish a connection with Google BigQuery.
`gg.handler.name.readTimeout`	Optional	Positive integer	`30000`	The maximum amount of time in milliseconds to wait for the handler to read data from an established connection.
`gg.handler.name.metaColumnsTemplate`	Optional	A legal string	None	A legal string specifying the `metaColumns` to be included. If you set `auditLogMode` to `true`, it is important that you set the `metaColumnsTemplate` property to view the operation type for the row inserted in the audit log.
`gg.handler.name.auditLogMode`	Optional	`true` | `false`	`false`	Set to `true`, the handler writes each record to the target without any primary key. Everything is processed as insert. Set to `false`, the handler tries to merge incoming records into the target table if they have the same primary key. Primary keys are needed for this property. The trail source records need to have a full image updates to merge correctly.
`gg.handler.name.pkUpdateHandling`	Optional	`abend` | `delete-insert`	`abend`	Sets how the handler handles update operations that change a primary key. Primary key operations can be problematic for the BigQuery Handler and require special consideration: • `abend`- indicates the process abends. • `delete-insert`- indicates the process treats the operation as a delete and an insert. The full before image is required for this property to work correctly. Without full before and after row images the insert data are incomplete. Oracle recommends this option.

2.2.2 Review a Sample Configuration

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

```
gg.handlerlist = bigquery

#The handler properties
gg.handler.bigquery.type = bigquery
gg.handler.bigquery.projectId = festive-athlete-201315
gg.handler.bigquery.datasetId = oggbd
gg.handler.bigquery.credentialsFile = credentials.json
gg.handler.bigquery.auditLogMode = true
gg.handler.bigquery.pkUpdateHandling = delete-insert

gg.handler.bigquery.metaColumnsTemplate =${optype}, ${position}
```

2.2.3 Proxy Settings

To connect to BigQuery using a proxy server, you must configure the proxy host and the proxy port in the properties file as follows:

```
javawriter.bootoptions= -Dhttps.proxyHost=proxy_host_name
                                -Dhttps.proxyPort=proxy_port_number
```

2.2.4 Configuring Handler Authentication

You have to configure the BigQuery Handler authentication using the credentials in the JSON file downloaded from Google BigQuery.

Download the credentials file:

1. Login into your Google account at cloud.google.com.
2. Click **Console**, and then to go to the Dashboard where you can select your project.
3. From the navigation menu, click **APIs & Services** then select **Credentials**.
4. From the Create Credentials menu, choose **Service account key**.
5. Choose the JSON key type to download the JSON credentials file for your system.

Once you have the credentials file, you can authenticate the handler in one of these two ways:

- Specify the path to the credentials file in the properties file with the `gg.handler.name.credentialsFile` configuration property.

  The path of the credentials file must contain the path with no wildcard appended. If you include the `*` wildcard in the path to the credentials file, the file is not recognized.

  Or

- Set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable on your system. For example:

  ```
  export GOOGLE_APPLICATION_CREDENTIALS = credentials.json
  ```

Then restart the Oracle GoldenGate manager process.

3
Using the Cassandra Handler

Learn how to use the Cassandra Handler, which provides the interface to Apache Cassandra databases.

Topics:

- Overview (page 3-1)
- Detailing the Functionality (page 3-1)
- Setting Up and Running the Cassandra Handler (page 3-7)
- About Automated DDL Handling (page 3-11)
- Performance Considerations (page 3-12)
- Additional Considerations (page 3-12)
- Troubleshooting (page 3-13)

3.1 Overview

Apache Cassandra is a NoSQL Database Management System designed to store large amounts of data. A Cassandra cluster configuration provides horizontal scaling and replication of data across multiple machines. It can provide high availability and eliminate a single point of failure by replicating data to multiple nodes within a Cassandra cluster. Apache Cassandra is open source and designed to run on low-cost commodity hardware.

Cassandra relaxes the axioms of a traditional relational database management systems (RDBMS) regarding atomicity, consistency, isolation, and durability. When considering implementing Cassandra, it is important to understand its differences from a traditional RDBMS and how those differences affect your specific use case.

Cassandra provides eventual consistency. Under the eventual consistency model, accessing the state of data for a specific row eventually returns the latest state of the data for that row as defined by the most recent change. However, there may be a latency period between the creation and modification of the state of a row and what is returned when the state of that row is queried. The benefit of eventual consistency is that the latency period is predicted based on your Cassandra configuration and the level of work load that your Cassandra cluster is currently under, see http://cassandra.apache.org/.

The Cassandra Handler provides some control over consistency with the configuration of the `gg.handler.name.consistencyLevel` property in the Java Adapter properties file.

3.2 Detailing the Functionality

Topics:

- About the Cassandra Data Types (page 3-2)

- About Catalog, Schema, Table, and Column Name Mapping (page 3-3)
  Traditional RDBMSs separate structured data into tables. Related tables are included in higher-level collections called databases. Cassandra contains both of these concepts. Tables in an RDBMS are also tables in Cassandra, while database schemas in an RDBMS are keyspaces in Cassandra.
- About DDL Functionality (page 3-3)
- How Operations are Processed (page 3-5)
- About Compressed Updates vs. Full Image Updates (page 3-6)
- About Primary Key Updates (page 3-7)

3.2.1 About the Cassandra Data Types

Cassandra provides a number of column data types and most of these data types are supported by the Cassandra Handler.

Supported Cassandra Data Types

```
ASCII
BIGINT
BLOB
BOOLEAN
DATE
DECIMAL
DOUBLE
DURATION
FLOAT
INET
INT
SMALLINT
TEXT
TIME
TIMESTAMP
TIMEUUID
TINYINT
UUID
VARCHAR
VARINT
```

Unsupported Cassandra Data Types

```
COUNTER
MAP
SET
LIST
UDT (user defined type)
TUPLE
CUSTOM_TYPE
```

Supported Database Operations

```
INSERT
UPDATE (captured as INSERT)
DELETE
```

The Cassandra commit log files do *not* record any before images for the `UPDATE` or `DELETE` operations. So the captured operations never have a before image section.

Oracle GoldenGate features that rely on before image records, such as Conflict Detection and Resolution, are not available.

Unsupported Database Operations

```
TRUNCATE
DDL (CREATE, ALTER, DROP)
```

The data type of the column value in the source trail file must be converted to the corresponding Java type representing the Cassandra column type in the Cassandra Handler. This data conversion introduces the risk of a runtime conversion error. A poorly mapped field (such as `varchar` as the source containing alpha numeric data to a Cassandra `int`) may cause a runtime error and cause the Cassandra Handler to abend. You can view the Cassandra Java type mappings at:

https://github.com/datastax/java-driver/tree/3.x/manual#cql-to-java-type-mapping

It is possible that the data may require specialized processing to get converted to the corresponding Java type for intake into Cassandra. If this is the case, you have two options:

- Try to use the general regular expression search and replace functionality to format the source column value data in a way that can be converted into the Java data type for use in Cassandra.

  Or

- Implement or extend the default data type conversion logic to override it with custom logic for your use case. Contact Oracle Support for guidance.

3.2.2 About Catalog, Schema, Table, and Column Name Mapping

Traditional RDBMSs separate structured data into tables. Related tables are included in higher-level collections called databases. Cassandra contains both of these concepts. Tables in an RDBMS are also tables in Cassandra, while database schemas in an RDBMS are keyspaces in Cassandra.

It is important to understand how data maps from the metadata definition in the source trail file are mapped to the corresponding keyspace and table in Cassandra. Source tables are generally either two-part names defined as `schema.table`,or three-part names defined as `catalog.schema.table`.

The following table explains how catalog, schema, and table names map into Cassandra. Unless you use special syntax, Cassandra converts all keyspace, table names, and column names to lower case.

Table Name in Source Trail File	Cassandra Keyspace Name	Cassandra Table Name
`QASOURCE.TCUSTMER`	`qasource`	`tcustmer`
`dbo.mytable`	`dbo`	`mytable`
`GG.QASOURCE.TCUSTORD`	`gg_qasource`	`tcustord`

3.2.3 About DDL Functionality

Topics:

- About the Keyspaces (page 3-4)
- About the Tables (page 3-4)
- Addng Column Functionality (page 3-5)
- Dropping Column Functionality (page 3-5)

3.2.3.1 About the Keyspaces

The Cassandra Handler does *not* automatically create keyspaces in Cassandra. Keyspaces in Cassandra define a replication factor, the replication strategy, and topology. The Cassandra Handler does not have enough information to create the keyspaces, so you must manually create them.

You can create keyspaces in Cassandra by using the `CREATE KEYSPACE` command from the Cassandra shell.

3.2.3.2 About the Tables

The Cassandra Handler can automatically create tables in Cassandra if you configure it to do so. The source table definition may be a poor source of information to create tables in Cassandra. Primary keys in Cassandra are divided into:

- **Partitioning keys** that define how data for a table is separated into partitions in Cassandra.
- **Clustering keys** that define the order of items within a partition.

In the default mapping for automated table creation, the first primary key is the partition key, and any additional primary keys are mapped as clustering keys.

Automated table creation by the Cassandra Handler may be fine for proof of concept, but it may result in data definitions that do not scale well. When the Cassandra Handler creates tables with poorly constructed primary keys, the performance of ingest and retrieval may decrease as the volume of data stored in Cassandra increases. Oracle recommends that you analyze the metadata of your replicated tables, then manually create corresponding tables in Cassandra that are properly partitioned and clustered for higher scalability.

Primary key definitions for tables in Cassandra are immutable after they are created. Changing a Cassandra table primary key definition requires the following manual steps:

1. Create a staging table.
2. Populate the data in the staging table from original table.
3. Drop the original table.
4. Re-create the original table with the modified primary key definitions.
5. Populate the data in the original table from the staging table.
6. Drop the staging table.

3.2.3.3 Addng Column Functionality

You can configure the Cassandra Handler to add columns that exist in the source trail file table definition but are missing in the Cassandra table definition. The Cassandra Handler can accommodate metadata change events of this kind. A reconciliation process reconciles the source table definition to the Cassandra table definition. When the Cassandra Handler is configured to add columns, any columns found in the source table definition that do not exist in the Cassandra table definition are added. The reconciliation process for a table occurs after application startup the first time an operation for the table is encountered. The reconciliation process reoccurs after a metadata change event on a source table, when the first operation for the source table is encountered after the change event.

3.2.3.4 Dropping Column Functionality

You can configure the Cassandra Handler to drop columns that do not exist in the source trail file definition but exist in the Cassandra table definition. The Cassandra Handler can accommodate metadata change events of this kind. A reconciliation process reconciles the source table definition to the Cassandra table definition. When the Cassandra Handler is configured to drop, columns any columns found in the Cassandra table definition that are not in the source table definition are dropped.

> **Caution:**
>
> Dropping a column permanently removes data from a Cassandra table. Carefully consider your use case before you configure this mode.

> **Note:**
>
> Primary key columns cannot be dropped. Attempting to do so results in an abend.

> **Note:**
>
> Column name changes are not well-handled because there is no DDL is processed. When a column name changes in the source database, the Cassandra Handler interprets it as dropping an existing column and adding a new column.

3.2.4 How Operations are Processed

The Cassandra Handler pushes operations to Cassandra using either the asynchronous or synchronous API. In asynchronous mode, operations are flushed at transaction commit (grouped transaction commit using `GROUPTRANSOPS`) to ensure write

durability. The Cassandra Handler does not interface with Cassandra in a transactional way.

Supported Database Operations

```
INSERT
UPDATE (captured as INSERT)
DELETE
```

The Cassandra commit log files do *not* record any before images for the `UPDATE` or `DELETE` operations. So the captured operations never have a before image section. Oracle GoldenGate features that rely on before image records, such as Conflict Detection and Resolution, are not available.

Unsupported Database Operations

```
TRUNCATE
DDL (CREATE, ALTER, DROP)
```

Insert, update, and delete operations are processed differently in Cassandra than a traditional RDBMS. The following explains how insert, update, and delete operations are interpreted by Cassandra:

- Inserts: If the row does not exist in Cassandra, then an insert operation is processed as an insert. If the row already exists in Cassandra, then an insert operation is processed as an update.
- Updates: If a row does not exist in Cassandra, then an update operation is processed as an insert. If the row already exists in Cassandra, then an update operation is processed as insert.
- Delete:If the row does not exist in Cassandra, then a delete operation has no effect. If the row exists in Cassandra, then a delete operation is processed as a delete.

The state of the data in Cassandra is idempotent. You can replay the source trail files or replay sections of the trail files. The state of the Cassandra database must be the same regardless of the number of times that the trail data is written into Cassandra.

3.2.5 About Compressed Updates vs. Full Image Updates

Oracle GoldenGate allows you to control the data that is propagated to the source trail file in the event of an update. The data for an update in the source trail file is either a compressed or a full image of the update, and the column information is provided as follows:

Compressed
For the primary keys and the columns for which the value changed. Data for columns that have not changed is not provided in the trail file.

Full Image
For all columns, including primary keys, columns for which the value has changed, and columns for which the value has not changed.

The amount of information about an update is important to the Cassandra Handler. If the source trail file contains full images of the change data, then the Cassandra Handler can use prepared statements to perform row updates in Cassandra. Full images also allow the Cassandra Handler to perform primary key updates for a row in Cassandra. In Cassandra, primary keys are immutable, so an update that changes a

primary key must be treated as a delete and an insert. Conversely, when compressed updates are used, prepared statements cannot be used for Cassandra row updates. Simple statements identifying the changing values and primary keys must be dynamically created and then executed. With compressed updates, primary key updates are not possible and as a result, the Cassandra Handler will abend.

You must set the control properties `gg.handler.name.compressedUpdates` and `gg.handler.name.compressedUpdatesfor` so that the handler expects either compressed or full image updates.

The default value, `true`, sets the Cassandra Handler to expect compressed updates. Prepared statements are not be used for updates, and primary key updates cause the handler to abend.

When the value is `false`, prepared statements are used for updates and primary key updates can be processed. A source trail file that does not contain full image data can lead to corrupted data columns, which are considered null. As a result, the null value is pushed to Cassandra. If you are not sure about whether the source trail files contains compressed or full image data, set `gg.handler.name.compressedUpdates` to `true`.

CLOB and BLOB data types do not propagate LOB data in updates unless the LOB column value changed. Therefore, if the source tables contain LOB data, set `gg.handler.name.compressedUpdates` to `true`.

3.2.6 About Primary Key Updates

Primary key values for a row in Cassandra are immutable. An update operation that changes any primary key value for a Cassandra row must be treated as a delete and insert. The Cassandra Handler can process update operations that result in the change of a primary key in Cassandra only as a delete and insert. To successfully process this operation, the source trail file *must* contain the complete before and after change data images for all columns. The `gg.handler.name.compressed` configuration property of the Cassandra Handler must be set to `false` for primary key updates to be successfully processed.

3.3 Setting Up and Running the Cassandra Handler

Instructions for configuring the Cassandra Handler components and running the handler are described in the following sections.

Before you run the Cassandra Handler, you must install the Datastax Driver for Cassandra and set the `gg.classpath` configuration property.

Get the Driver Libraries

The Datastax Java Driver for Cassandra does not ship with Oracle GoldenGate for Big Data. You can download the recommended version of the Datastax Java Driver for Cassandra 3.1 at:

https://github.com/datastax/java-driver

Set the Classpath

You must configure the `gg.classpath` configuration property in the Java Adapter properties file to specify the JARs for the Datastax Java Driver for Cassandra. Ensure that this JAR is first in the list.

```
gg.classpath=/path_to_repository/com/datastax/cassandra/cassandra-drive r-core/3.3.1/
cassandra-driver-core-3.3.1.jar:/path_to_apache_cassandra/cassandra-3.11.0/ lib/*
```

Topics:

- Understanding the Cassandra Handler Configuration (page 3-8)
- Review a Sample Configuration (page 3-10)
- Configuring Security (page 3-11)

3.3.1 Understanding the Cassandra Handler Configuration

The following are the configurable values for the Cassandra Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Cassandra Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=cassandra` and the other Cassandra properties as follows:

Table 3-1 Cassandra Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handlerlist`	Required	Any string	None	Provides a name for the Cassandra Handler. The Cassandra Handler name then becomes part of the property names listed in this table.
`gg.handler.`*`name`*`.type=cassandra`	Required	`cassandra`	None	Selects the Cassandra Handler for streaming change data capture into name.
`gg.handler.`*`name`*`.mode`	Optional	`op` | `tx`	`op`	The default is recommended. In `op` mode, operations are processed as received. In `tx` mode, operations are cached and processed at transaction commit. The `tx`mode is slower and creates a larger memory footprint.
`gg.handler.name.contactPoints=`	Optional	A comma separated list of host names that the Cassandra Handler will connect to.	`localhost`	A comma-separated list of the Cassandra host machines for the driver to establish an initial connection to the Cassandra cluster. This configuration property does *not* need to include all the machines enlisted in the Cassandra cluster. By connecting to a single machine, the driver can learn about other machines in the Cassandra cluster and establish connections to those machines as required.
`gg.handler.`*`name`*`.username`	Optional	A legal username string.	None	A user name for the connection to name. Required if Cassandra is configured to require credentials.
`gg.handler.`*`name`*`.password`	Optional	A legal password string.	None	A password for the connection to name. Required if Cassandra is configured to require credentials.

Table 3-1 (Cont.) Cassandra Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.compressedUpdates`	Optional	`true` | `false`	`true`	Sets the Cassandra Handler whether to expect full image updates from the source trail file. A value of `true` means that updates in the source trail file only contain column data for the primary keys and for columns that changed. The Cassandra Handler executes updates as simple statements updating only the columns that changed. A value of `false` means that updates in the source trail file contain column data for primary keys and all columns regardless of whether the column value has changed. The Cassandra Handler is able to use prepared statements for updates, which can provide better performance for streaming data to name.
`gg.handler.name.ddlHandling`	Optional	`CREATE` | `ADD` | `DROP` in any combination with values delimited by a comma	None	Configures the Cassandra Handler for the DDL functionality to provide. Options include `CREATE`, `ADD`, and `DROP`. These options can be set in any combination delimited by commas. When `CREATE` is enabled, the Cassandra Handler creates tables in Cassandra if a corresponding table does not exist. When `ADD` is enabled, the Cassandra Handler adds columns that exist in the source table definition that do *not* exist in the corresponding Cassandra table definition. When `DROP` is enabled, the handler drops columns that exist in the Cassandra table definition that do *not* exist in the corresponding source table definition.
`gg.handler.name.cassandraMode`	Optional	`async` | `sync`	`async`	Sets the interaction between the Cassandra Handler and name. Set to `async` for asynchronous interaction. Operations are sent to Cassandra asynchronously and then flushed at transaction commit to ensure durability. Asynchronous provides better performance. Set to `sync` for synchronous interaction. Operations are sent to Cassandra synchronously.

Table 3-1 (Cont.) Cassandra Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.consistencyLevel`	Optional	`ALL` | `ANY` | `EACH_QUORUM` | `LOCAL_ONE` | `LOCAL_QUORUM` | `ONE` | `QUORUM` | `THREE` | `TWO`	The Cassandra default.	Sets the consistency level for operations with name. It configures the criteria that must be met for storage on the Cassandra cluster when an operation is executed. Lower levels of consistency may provide better performance, while higher levels of consistency are safer. An advanced configuration property so that you can override the SSL `javax.net.ssl.SSLContext` and cipher suites. The fully qualified class name is provided here and the class must be included in the classpath. The class must implement the `com.datastax.driver.core.SSLOptions` interface in the Datastax Cassandra Java driver. This configuration property is only applicable if `gg.handler.name.withSSL` is set to `true`, see http://docs.datastax.com/en/developer/java-driver/3.3/manual/ssl/.
`gg.handler.name.withSSL`	Optional	`true` | `false`	`false`	Set to true to enable secured connections to the Cassandra cluster using SSL. This requires additional Java boot options configuration, see http://docs.datastax.com/en/developer/java-driver/3.3/manual/ssl/.
`gg.handler.name.port`	Optional	Integer	`9042`	Set to configure the port number that the Cassandra Handler attempts to connect to Cassandra server instances. You can override the default in the Cassandra YAML files.

3.3.2 Review a Sample Configuration

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

```
gg.handlerlist=cassandra

#The handler properties
gg.handler.cassandra.type=cassandra
gg.handler.cassandra.mode=op
gg.handler.cassandra.contactPoints=localhost
gg.handler.cassandra.ddlHandling=CREATE,ADD,DROP
gg.handler.cassandra.compressedUpdates=true
gg.handler.cassandra.cassandraMode=async
gg.handler.cassandra.consistencyLevel=ONE
```

3.3.3 Configuring Security

The Cassandra Handler connection to the Cassandra Cluster can be secured using user name and password credentials. These are set using the following configuration properties:

```
gg.handler.name.username
gg.handler.name.password
```

Optionally, the connection to the Cassandra cluster can be secured using SSL. To enable SSL security set the following parameter:

```
gg.handler.name.withSSL=true
```

Additionally, the Java `bootoptions` must be configured to include the location and password of the `keystore` and the location and password of the `truststore`. For example:

```
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./
dirprm
-Djavax.net.ssl.trustStore=/path/to/client.truststore
-Djavax.net.ssl.trustStorePassword=password123
-Djavax.net.ssl.keyStore=/path/to/client.keystore
-Djavax.net.ssl.keyStorePassword=password123
```

3.4 About Automated DDL Handling

The Cassandra Handler performs the table check and reconciliation process the first time an operation for a source table is encountered. Additionally, a DDL event or a metadata change event causes the table definition in the Cassandra Handler to be marked as dirty. Therefore, the next time an operation for the table is encountered, the handler repeats the table check, and reconciliation process as described in the following section.

Topics:

- About the Table Check and Reconciliation Process (page 3-11)
- Capturing New Change Data (page 3-12)

3.4.1 About the Table Check and Reconciliation Process

The Cassandra Handler first interrogates the target Cassandra database to determine whether the target Cassandra keyspace exists. If the target Cassandra keyspace does not exist, then the Cassandra Handler abends. Keyspaces must be created by the user. The log file must contain the error of the exact keyspace name that the Cassandra Handler is expecting.

Next, the Cassandra Handler interrogates the target Cassandra database for the table definition. If the table does not exist, the Cassandra Handler either creates a table if `gg.handler.name.ddlHandling` includes the `CREATE` option or abends the process. A message is logged that shows you the table that does not exist in Cassandra.

If the table exists in Cassandra, then the Cassandra Handler reconciles the table definition from the source trail file and the table definition in Cassandra. This reconciliation process searches for columns that exist in the source table definition and

not in the corresponding Cassandra table definition. If it locates columns fitting this criteria and the `gg.handler.name.ddlHandling` property includes `ADD`, then the Cassandra Handler adds the columns to the target table in Cassandra. Otherwise, it ignores these columns.

Next, the Cassandra Handler searches for columns that exist in the target Cassandra table but do not exist in the source table definition. If it locates columns that fit this criteria and the `gg.handler.name.ddlHandling` property includes `DROP`, then the Cassandra Handler removes these columns from the target table in Cassandra. Otherwise those columns are ignored.

Finally, the prepared statements are built.

3.4.2 Capturing New Change Data

You can capture all of the new change data into your Cassandra database, including the DDL changes in the trail, for the target apply. Following is the acceptance criteria:

```
AC1: Support Cassandra as a bulk extract
AC2: Support Cassandra as a CDC source
AC4: All Cassandra supported data types are supported
AC5: Should be able to write into different tables based on any filter conditions,
like Updates to Update tables or based on primary keys
AC7: Support Parallel processing with multiple threads
AC8: Support Filtering based on keywords
AC9: Support for Metadata provider
AC10: Support for DDL handling on sources and target
AC11: Support for target creation and updating of metadata.
AC12: Support for error handling and extensive logging
AC13: Support for Conflict Detection and Resolution
AC14: Performance should be on par or better than HBase
```

3.5 Performance Considerations

Configuring the Cassandra Handler for `async` mode provides better performance than `sync` mode. Set Replicat property `GROUPTRANSOPS` must be set to the default value of 1000.

Setting the consistency level directly affects performance. The higher the consistency level, the more work must occur on the Cassandra cluster before the transmission of a given operation can be considered complete. Select the minimum consistency level that still satisfies the requirements of your use case.

The Cassandra Handler can work in either operation (`op`) or transaction (`tx`) mode. For the best performance operation mode is recommended:

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

3.6 Additional Considerations

- Cassandra database requires at least one primary key. The value of any primary key cannot be null. Automated table creation fails for source tables that do not have a primary key.
- When `gg.handler.name.compressedUpdates=false` is set, the Cassandra Handler expects to update full before and after images of the data.

> **Note:**
>
> Using this property setting with a source trail file with partial image updates results in null values being updated for columns for which the data is missing. This configuration is incorrect and update operations pollute the target data with null values in columns that did not change.

- The Cassandra Handler does *not* process DDL from the source database, even if the source database provides DDL Instead, it reconciles between the source table definition and the target Cassandra table definition. A DDL statement executed at the source database that changes a column name appears to the Cassandra Handler as if a column is dropped from the source table and a new column is added. This behavior depends on how the `gg.handler.name.ddlHandling` property is configured.

`gg.handler.name.ddlHandling` Configuration	**Behavior**
Not configured for `ADD` or `DROP`	Old column name and data maintained in Cassandra. New column is not created in Cassandra, so no data is replicated for the new column name from the DDL change forward.
Configured for `ADD` only	Old column name and data maintained in Cassandra. New column iscreated in Cassandra and data replicated for the new column name from the DDL change forward. Column mismatch between the data is located before and after the DDL change.
Configured for `DROP` only	Old column name and data dropped in Cassandra. New column is not created in Cassandra, so no data replicated for the new column name.
Configured for `ADD` and `DROP`	Old column name and data dropped in Cassandra. New column is created in Cassandra, and data is replicated for the new column name from the DDL change forward.

3.7 Troubleshooting

This section contains information to help you troubleshoot various issues. Review the following topics for additional help:

- Java Classpath (page 3-14)
- Logging (page 3-14)
- Write Timeout Exception (page 3-14)
- Logging (page 3-15)
- Datastax Driver Error (page 3-15)

3.7.1 Java Classpath

When the classpath that is intended to include the required client libraries, a `ClassNotFound` exception appears in the log file. To troubleshoot, set the Java Adapter logging to `DEBUG`, and then run the process again. At the debug level, the log contains data about the JARs that were added to the classpath from the `gg.classpath` configuration variable. The `gg.classpath` variable selects the asterisk (`*`) wildcard character to select all JARs in a configured directory. For example, `/usr/cassandra/cassandra-java-driver-3.3.1/*:/usr/cassandra/cassandra-java-driver-3.3.1/lib/*`.

For more information about setting the classpath, see Setting Up and Running the Cassandra Handler (page 3-7)and Cassandra Handler Client Dependencies (page A-1).

3.7.2 Logging

The Cassandra Handler logs the state of its configuration to the Java log file. This is helpful because you can review the configuration values for the Cassandra Handler. A sample of the logging of the state of the configuration follows:

```
**** Begin Cassandra Handler - Configuration Summary ****
  Mode of operation is set to op.
  The Cassandra cluster contact point(s) is [localhost].
  The handler has been configured for GoldenGate compressed updates (partial image
updates).
  Sending data to Cassandra in [ASYNC] mode.
  The Cassandra consistency level has been set to [ONE].
  Cassandra Handler DDL handling:
    The handler will create tables in Cassandra if they do not exist.
    The handler will add columns to Cassandra tables for columns in the source
metadata that do not exist in Cassandra.
    The handler will drop columns in Cassandra tables for columns that do not exist
in the source metadata.
**** End Cassandra Handler - Configuration Summary ****
```

3.7.3 Write Timeout Exception

When running the Cassandra handler, you may experience a `com.datastax.driver.core.exceptions.WriteTimeoutException` exception that causes the Replicat process to abend. It is likely to occur under some or all of the following conditions:

- The Cassandra Handler processes large numbers of operations, putting the Cassandra cluster under a significant processing load.
- `GROUPTRANSOPS` is configured higher than the value of 1000 default.
- The Cassandra Handler is configured in asynchronous mode.
- The Cassandra Handler is configured with a consistency level higher than `ONE`.

When this problem occurs, the Cassandra Handler is streaming data faster than the Cassandra cluster can process it. The write latency in the Cassandra cluster finally exceeds the write request timeout period, which in turn results in the exception.

The following are potential solutions:

- Increase the write request timeout period. This is controlled with the `write_request_timeout_in_ms` property in Cassandra and is located in the `cassandra.yaml` file in the *`cassandra_install`*`/conf` directory. The default is 2000 (2 seconds). You can increase this value to move past the error, and then restart the Cassandra node or nodes for the change to take effect.
- Decrease the `GROUPTRANSOPS` configuration value of the Replicat process. Typically, decreasing the `GROUPTRANSOPS` configuration decreases the size of transactions processed and reduces the likelihood that the Cassandra Handler can overtax the Cassandra cluster.
- Reduce the consistency level of the Cassandra Handler. This in turn reduces the amount of work the Cassandra cluster has to complete for an operation to be considered as written.

3.7.4 Logging

The `java.lang.NoClassDefFoundError: io/netty/util/Timer` error can occur in both the 3.3 and 3.2 versions of downloaded Datastax Java Driver. This is because the `netty-common` JAR file is inadvertently missing from the Datastax driver tar file. You must manually obtain the`netty-common` JAR file of the same netty version, and then add it to the classpath.

3.7.5 Datastax Driver Error

If you didn't add the `cassandra-driver-core-3.3.1.jar` file in the `gg.classpath` property, then this exception can occur:

```
com.datastax.driver.core.exceptions.UnresolvedUserTypeException: Cannot
resolve user type keyspace.duration
```

If there are tables with a `duration` data type column, this exception occurs. Using the `Cassandra driver, cassandra-driver-core-3.3.1.jar` in the `gg.classpath` property resolves the error. See Setting Up and Running the Cassandra Handler (page 3-7).

4
Using the Elasticsearch Handler

Learn how to use the Elasticsearch Handler, which allows you to store, search, and analyze large volumes of data quickly and in near real time.

Topics:

- Overview (page 4-1)
- Detailing the Functionality (page 4-1)
- Setting Up and Running the Elasticsearch Handler (page 4-4)
- Performance Consideration (page 4-7)
- About the Shield Plug-In Support (page 4-8)
- About DDL Handling (page 4-8)
- Troubleshooting (page 4-8)
- Logging (page 4-10)
- Known Issues in the Elasticsearch Handler (page 4-11)

4.1 Overview

Elasticsearch is a highly scalable open-source full-text search and analytics engine. Elasticsearch allows you to store, search, and analyze large volumes of data quickly and in near real time. It is generally used as the underlying engine or technology that drives applications with complex search features.

The Elasticsearch Handler uses the Elasticsearch Java client to connect and receive data into Elasticsearch node, see https://www.elastic.co.

4.2 Detailing the Functionality

Topics:

- About the Elasticsearch Version Property (page 4-2)
- About the Index and Type (page 4-2)
- About the Document (page 4-2)
- About the Primary Key Update (page 4-2)
- About the Data Types (page 4-3)
- Operation Mode (page 4-3)
- Operation Processing Support (page 4-3)
- About the Connection (page 4-3)

4.2.1 About the Elasticsearch Version Property

The Elasticsearch Handler property `gg.handler.name.version` should be set according to the version of the Elasticsearch cluster. The Elasticsearch Handler uses a Java Transport client, which must have the same major version (such as, 2.*x*, or 5.*x*) as the nodes in the cluster. The Elasticsearch Handler can connect to clusters that have a different minor version (such as, 2.3.*x*) though new functionality may not be supported.

4.2.2 About the Index and Type

An Elasticsearch **index** is a collection of documents with similar characteristics. An index can only be created in lowercase. An Elasticsearch **type** is a logical group within an index. All the documents within an index or type should have same number and type of fields.

The Elasticsearch Handler maps the source trail schema concatenated with source trail table name to construct the index. For three-part table names in source trail, the index is constructed by concatenating source catalog, schema, and table name.

The Elasticsearch Handler maps the source table name to the Elasticsearch type. The type name is case-sensitive.

Table 4-1 Elasticsearch Mapping

Source Trail	Elasticsearch Index	Elasticsearch Type
`schema.tablename`	`schema_tablename`	`tablename`
`catalog.schema.tablename`	`catalog_schema_tablename`	`tablename`

If an index does not already exist in the Elasticsearch cluster, a new index is created when Elasticsearch Handler receives (`INSERT` or `UPDATE` operation in source trail) data.

4.2.3 About the Document

An Elasticsearch document is a basic unit of information that can be indexed. Within an index or type, you can store as many documents as you want. Each document has an unique identifier based on the `_id` field.

The Elasticsearch Handler maps the source trail primary key column value as the document identifier.

4.2.4 About the Primary Key Update

The Elasticsearch document identifier is created based on the source table's primary key column value. The document identifier cannot be modified. The Elasticsearch handler processes a source primary key's update operation by performing a `DELETE` followed by an `INSERT`. While performing the `INSERT`, there is a possibility that the new document may contain fewer fields than required. For the `INSERT` operation to contain all the fields in the source table, enable trail Extract to capture the full data before images for update operations or use `GETBEFORECOLS` to write the required column's before images.

4.2.5 About the Data Types

Elasticsearch supports the following data types:

- `32-bit integer`
- `64-bit integer`
- `Double`
- `Date`
- `String`
- `Binary`

4.2.6 Operation Mode

The Elasticsearch Handler uses the operation mode for better performance. The `gg.handler.name.mode` property is not used by the handler.

4.2.7 Operation Processing Support

The Elasticsearch Handler maps the source table name to the Elasticsearch type. The type name is case-sensitive.

For three-part table names in source trail, the index is constructed by concatenating source catalog, schema, and table name.

`INSERT`
The Elasticsearch Handler creates a new index if the index does not exist, and then inserts a new document.

`UPDATE`
If an Elasticsearch index or document exists, the document is updated. If an Elasticsearch index or document does not exist, a new index is created and the column values in the `UPDATE` operation are inserted as a new document.

`DELETE`
If an Elasticsearch index or document exists, the document is deleted. If Elasticsearch index or document does not exist, a new index is created with zero fields.

The `TRUNCATE` operation is not supported.

4.2.8 About the Connection

A **cluster** is a collection of one or more nodes (servers) that holds the entire data. It provides federated indexing and search capabilities across all nodes.

A **node** is a single server that is part of the cluster, stores the data, and participates in the cluster's indexing and searching.

The Elasticsearch Handler property `gg.handler.name.ServerAddressList` can be set to point to the nodes available in the cluster.

4.3 Setting Up and Running the Elasticsearch Handler

You must ensure that the Elasticsearch cluster is setup correctly and the cluster is up and running, see https://www.elastic.co/guide/en/elasticsearch/reference/current/_installation.html. Alternatively, you can use Kibana to verify the setup.

Set the Classpath

The property `gg.classpath` must include all the jars required by the Java transport client. For a listing of the required client JAR files by version, see Elasticsearch Handler Client Dependencies (page C-1).

```
Default location of 2.X JARs:

    Elasticsearch_Home/lib/*
    Elasticsearch_Home/plugins/shield/*

Default location of 5.X JARs:

  Elasticsearch_Home/lib/*
    Elasticsearch_Home/plugins/x-pack/*
    Elasticsearch_Home/modules/transport-netty3/*
    Elasticsearch_Home/modules/transport-netty4/*
    Elasticsearch_Home/modules/reindex/*
```

The inclusion of the * wildcard in the path can include the * wildcard 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=Elasticsearch_Home/lib/*
```

Topics:

- Configuring the Elasticsearch Handler (page 4-4)
- About the Transport Client Settings Properties File (page 4-7)

4.3.1 Configuring the Elasticsearch Handler

The following are the configurable values for the Elasticsearch handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Elasticsearch Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=elasticsearch` and the other Elasticsearch properties as follows:

Table 4-2 Elasticsearch Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handlerlist`	Required	Name (any name of your choice)	None	The list of handlers to be used.

Table 4-2 (Cont.) Elasticsearch Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.type`	Required	`elasticsearch`	None	Type of handler to use. For example, Elasticsearch, Kafka, Flume, or HDFS.
`gg.handler.name.ServerAddressList`	Optional	*`Server:Port`*`[,` *`Server:Port`*`]`	`localhost:9300`	Comma separated list of contact points of the nodes to connect to the Elasticsearch cluster.
`gg.handler.name.clientSettingsFile`	Required	Transport client properties file.	None	The filename in classpath that holds Elasticsearch transport client properties used by the Elasticsearch Handler.
`gg.handler.name.version`	Optional	`2.x` | `5.x`	`2.x`	The version of the transport client used by the Elasticsearch Handler, this should be compatible with the Elasticsearch cluster.
`gg.handler.name.bulkWrite`	Optional	`true` | `false`	`false`	When this property is `true`, the Elasticsearch Handler uses the bulk write API to ingest data into Elasticsearch cluster. The batch size of bulk write can be controlled using the `MAXTRANSOPS` Replicat parameter.

Table 4-2 (Cont.) Elasticsearch Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.numberAsString`	Optional	`true | false`	`false`	When this property is `true`, the Elasticsearch Handler would receive all the number column values (Long, Integer, or Double) in the source trail as strings into the Elasticsearch cluster.
`gg.handler.elasticsearch.upsert`	Optional	`true | false`	`true`	When this property is `true`, a new document is inserted if the document does not already exist when performing an `UPDATE` operation.

Example 4-1 Sample Handler Properties file:

For 2.*x* Elasticsearch cluster:

```
gg.handlerlist=elasticsearch
gg.handler.elasticsearch.type=elasticsearch
gg.handler.elasticsearch.ServerAddressList=localhost:9300
gg.handler.elasticsearch.clientSettingsFile=client.properties
gg.handler.elasticsearch.version=2.x
gg.classpath=/path/to/elastic/lib/*
```

For 2.*x* Elasticsearch cluster with Shield:

```
gg.handlerlist=elasticsearch
gg.handler.elasticsearch.type=elasticsearch
gg.handler.elasticsearch.ServerAddressList=localhost:9300
gg.handler.elasticsearch.clientSettingsFile=client.properties
gg.handler.elasticsearch.version=2.x
gg.classpath=/path/to/elastic/lib/*:/path/to/elastic/plugins/shield/*
```

For 5.*x* Elasticsearch cluster:

```
gg.handlerlist=elasticsearch
gg.handler.elasticsearch.type=elasticsearch
gg.handler.elasticsearch.ServerAddressList=localhost:9300
gg.handler.elasticsearch.clientSettingsFile=client.properties
gg.handler.elasticsearch.version=5.x
gg.classpath=/path/to/elastic/lib/*:/path/to/elastic/modules/transport-netty4/*:/
path/to/elastic/modules/reindex/*
```

For 5.*x* Elasticsearch cluster with x-pack:

```
gg.handlerlist=elasticsearch
gg.handler.elasticsearch.type=elasticsearch
gg.handler.elasticsearch.ServerAddressList=localhost:9300
gg.handler.elasticsearch.clientSettingsFile=client.properties
gg.handler.elasticsearch.version=5.x
gg.classpath=/path/to/elastic/lib/*:/path/to/elastic/plugins/x-pack/*:/path/to/
elastic/modules/transport-netty4/*:/path/to/elastic/modules/reindex/*
```

Sample Replicat configuration and a Java Adapter Properties files can be found at the following directory:

```
GoldenGate_install_directory/AdapterExamples/big-data/elasticsearch
```

4.3.2 About the Transport Client Settings Properties File

The Elasticsearch Handler uses a Java Transport client to interact with Elasticsearch cluster. The Elasticsearch cluster may have addional plug-ins like shield or x-pack, which may require additional configuration.

The `gg.handler.name.clientSettingsFile` property should point to a file that has additional client settings based on the version of Elasticsearch cluster. The Elasticsearch Handler attempts to locate and load the client settings file using the Java classpath. Te Java classpath must include the directory containing the properties file.

The client properties file for Elasticsearch 2.*x*(without any plug-in) is:

```
cluster.name=Elasticsearch_cluster_name
```

The client properties file for Elasticsearch 2.X with the Shield plug-in:

```
cluster.name=Elasticsearch_cluster_name
shield.user=shield_username:shield_password
```

The Shield plug-in also supports additional capabilities like SSL and IP filtering. The properties can be set in the `client.properties` file, see https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.4/transport-client.html and https://www.elastic.co/guide/en/shield/current/_using_elasticsearch_java_clients_with_shield.html.

The `client.properties` file for Elasticsearch 5.*x* with the X-Pack plug-in is:

```
cluster.name=Elasticsearch_cluster_name
xpack.security.user=x-pack_username:x-pack-password
```

The X-Pack plug-in also supports additional capabilities. The properties can be set in the `client.properties` file, see https://www.elastic.co/guide/en/elasticsearch/client/java-api/5.1/transport-client.html and https://www.elastic.co/guide/en/x-pack/current/java-clients.html.

4.4 Performance Consideration

The Elasticsearch Handler `gg.handler.name.bulkWrite` property is used to determine whether the source trail records should be pushed to the Elasticsearch cluster one at a time or in bulk using the bulk write API. When this property is **true**, the source trail operations are pushed to the Elasticsearch cluster in batches whose size can be controlled by the `MAXTRANSOPS` parameter in the generic Replicat parameter file. Using the bulk write API provides better performance.

Elasticsearch uses different thread pools to improve how memory consumption of threads are managed within a node. Many of these pools also have queues associated with them, which allow pending requests to be held instead of discarded.

For bulk operations, the default queue size is 50 (in version 5.2) and 200 (in version 5.3).

To avoid bulk API errors, you must set the Replicat `MAXTRANSOPS` size to match the bulk thread pool queue size at a minimum. The configuration `thread_pool.bulk.queue_size` property can be modified in the `elasticsearch.yaml` file.

4.5 About the Shield Plug-In Support

Elasticsearch versions 2.*x* supports a Shield plug-in which provides basic authentication, SSL and IP filtering. Similar capabilities exists in the X-Pack plug-in for Elasticsearch 5.*x*. The additional transport client settings can be configured in the Elasticsearch Handler using the `gg.handler.`*`name`*`.clientSettingsFile` property.

4.6 About DDL Handling

The Elasticsearch Handler does not react to any DDL records in the source trail. Any data manipulation records for a new source table results in auto-creation of index or type in the Elasticsearch cluster.

4.7 Troubleshooting

This section contains information to help you troubleshoot various issues.

Topics:

- Incorrect Java Classpath (page 4-8)
- Elasticsearch Version Mismatch (page 4-9)
- Transport Client Properties File Not Found (page 4-9)
- Cluster Connection Problem (page 4-9)
- Unsupported Truncate Operation (page 4-9)
- Bulk Execute Errors (page 4-10)

4.7.1 Incorrect Java Classpath

The most common initial error is an incorrect classpath to include all the required client libraries and creates a `ClassNotFound` exception in the `log4j` log file.

Also, it may be due to an error resolving the classpath if there is a typographic error in the `gg.classpath` variable.

The Elasticsearch transport client libraries do not ship with the Oracle GoldenGate for Big Data product. You should properly configure the `gg.classpath` property in the Java Adapter Properties file to correctly resolve the client libraries, see Setting Up and Running the Elasticsearch Handler (page 4-4).

4.7.2 Elasticsearch Version Mismatch

The Elasticsearch Handler `gg.handler.name.version` property must be set to 2.*x* or 5.*x* to match the major version number of the Elasticsearch cluster.

The following errors may occur when there is a wrong version configuration:

```
Error: NoNodeAvailableException[None of the configured nodes are available:]
```

```
ERROR 2017-01-30 22:35:07,240 [main] Unable to establish connection. Check handler
properties and client settings configuration.
```

```
java.lang.IllegalArgumentException: unknown setting [shield.user]
```

Ensure that all required plug-ins are installed and review documentation changes for any removed settings.

4.7.3 Transport Client Properties File Not Found

To resolve this exception:

```
ERROR 2017-01-30 22:33:10,058 [main] Unable to establish connection. Check handler
properties and client settings configuration.
```

Verify that the `gg.handler.name.clientSettingsFile` configuration property is correctly setting the Elasticsearch transport client settings file name. Verify that the `gg.classpath` variable includes the path to the correct file name and that the path to the properties file does not contain an asterisk (*) wildcard at the end.

4.7.4 Cluster Connection Problem

This error occurs when the Elasticsearch Handler is unable to connect to the Elasticsearch cluster:

```
Error: NoNodeAvailableException[None of the configured nodes are available:]
```

Use the following steps to debug the issue:

1. Ensure that the Elasticsearch server process is running.
2. Validate the `cluster.name` property in the client properties configuration file.
3. Validate the authentication credentials for the x-Pack or Shield plug-in in the client properties file.
4. Validate the `gg.handler.name.ServerAddressList` handler property.

4.7.5 Unsupported Truncate Operation

The following error occurs when the Elasticsearch Handler finds a `TRUNCATE` operation in the source trail:

```
oracle.goldengate.util.GGException: Elasticsearch Handler does not support the
operation: TRUNCATE
```

This exception error message is written to the handler log file before the RAeplicat process abends. Removing the `GETTRUNCATES` parameter from the Replicat parameter file resolves this error.

4.7.6 Bulk Execute Errors

```
""

DEBUG [main] (ElasticSearch5DOTX.java:130) - Bulk execute status: failures:[true]
buildFailureMessage:[failure in bulk execution: [0]: index [cs2cat_s1sch_n1tab],
type [N1TAB], id [83], message [RemoteTransportException[[UOvac8l][127.0.0.1:9300]
[indices:data/write/bulk[s][p]]]; nested: EsRejectedExecutionException[rejected
execution of org.elasticsearch.transport.TransportService$7@43eddfb2 on
EsThreadPoolExecutor[bulk, queue capacity = 50,
org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@5ef5f412[Running, pool
size = 4, active threads = 4, queued tasks = 50, completed tasks = 84]]];]
```

It may be due to the Elasticsearch running out of resources to process the operation. You can limit the Replicat batch size using `MAXTRANSOPS` to match the value of the `thread_pool.bulk.queue_size` Elasticsearch configuration parameter.

> **Note:**
>
> Changes to the Elasticsearch parameter, `thread_pool.bulk.queue_size`, are effective only after the Elasticsearch node is restarted.

4.8 Logging

The following log messages appear in the handler log file on successful connection:

Connection to 2.*x* Elasticsearch cluster:

```
INFO 2017-01-31 01:43:38,814 [main] **BEGIN Elasticsearch client settings**
INFO 2017-01-31 01:43:38,860 [main]     key[cluster.name] value[elasticsearch-user1-
myhost]
INFO 2017-01-31 01:43:38,860 [main]     key[request.headers.X-Found-Cluster]
value[elasticsearch-user1-myhost]
INFO 2017-01-31 01:43:38,860 [main]     key[shield.user] value[es_admin:user1]
INFO 2017-01-31 01:43:39,715 [main] Connecting to Server[myhost.us.example.com]
Port[9300]
INFO 2017-01-31 01:43:39,715 [main] Client node name:  Smith
INFO 2017-01-31 01:43:39,715 [main] Connected nodes: [{node-myhost}{vqtHikpGQP-
IXieHsgqXjw}{10.196.38.196}{198.51.100.1:9300}]
INFO 2017-01-31 01:43:39,715 [main] Filtered nodes: []
INFO 2017-01-31 01:43:39,715 [main] **END Elasticsearch client settings**
```

Connection to a 5.*x* Elasticsearch cluster:

```
INFO [main] (Elasticsearch5DOTX.java:38) - **BEGIN Elasticsearch client settings**
INFO [main] (Elasticsearch5DOTX.java:39) - {xpack.security.user=user1:user1_kibana,
cluster.name=elasticsearch-user1-myhost, request.headers.X-Found-
Cluster=elasticsearch-user1-myhost}
INFO [main] (Elasticsearch5DOTX.java:52) - Connecting to
Server[myhost.us.example.com] Port[9300]
INFO [main] (Elasticsearch5DOTX.java:64) - Client node name:  _client_
INFO [main] (Elasticsearch5DOTX.java:65) - Connected nodes: [{node-myhost}
```

```
{w9N25BrOSZeGsnUsogFn1A}{bIiIultVRjm0Ze57I3KChg}{myhost}{198.51.100.1:9300}]
INFO [main] (Elasticsearch5DOTX.java:66) - Filtered nodes: []
INFO [main] (Elasticsearch5DOTX.java:68) - **END Elasticsearch client settings**
```

4.9 Known Issues in the Elasticsearch Handler

Elasticsearch: Trying to input very large number

Very large numbers result in inaccurate values with Elasticsearch document. For example, 9223372036854775807, -9223372036854775808. This is an issue with the Elasticsearch server and not a limitation of the Elasticsearch Handler.

The workaround for this issue is to ingest all the number values as strings using the `gg.handler.name.numberAsString=true` property.

Elasticsearch: Issue with index

The Elasticsearch Handler is not able to input data into the same index if there are more than one table with similar column names and different column data types.

Index names are always lowercase though the `catalog/schema/tablename` in the trail may be case-sensitive.

5
Using the File Writer Handler

Learn how to use the File Writer Handler and associated event handlers, which enables you to write data files to a local system.

Topics:

- Overview (page 5-1)
  Learn how to use the File Writer Handler and the event handlers to transform data.
- Using the HDFS Event Handler (page 5-15)
  Learn how to use the HDFS Event Handler to load files generated by the File Writer Handler into HDFS.
- Using the Optimized Row Columnar Event Handler (page 5-19)
  Learn how to use the Optimized Row Columnar (ORC) Event Handler to generate data files in ORC format.
- Using the Oracle Cloud Infrastructure Event Handler (page 5-24)
  Learn how to use the Oracle Cloud Infrastructure Event Handler to load files generated by the File Writer Handler into an Oracle Cloud Infrastructure Object Store.
- Using the Oracle Cloud Infrastructure Classic Event Handler (page 5-30)
  Learn how to use the Oracle Cloud Infrastructure Classic Event Handler to load files generated by the File Writer Handler into an Oracle Cloud Infrastructure Classic Object Store.
- Using the Parquet Event Handler (page 5-36)
  Learn how to use the Parquet Event Handler to load files generated by the File Writer Handler into HDFS.
- Using the S3 Event Handler (page 5-41)
  Learn how to use the S3 Event Handler, which provides the interface to Amazon S3 web services.

5.1 Overview

Learn how to use the File Writer Handler and the event handlers to transform data.

The File Writer Handler supports generating data files in delimited text, XML, JSON, Avro, and Avro Object Container File formats. It is intended to fulfill an extraction, load, and transform use case. Data files are staged on your local file system. Then when writing to a data file is complete, you can use a third party application to read the file to perform additional processing.

The File Writer Handler also supports the event handler framework. The event handler framework allows data files generated by the File Writer Handler to be transformed into other formats, such as Optimized Row Columnar (ORC) or Parquet. Data files can be loaded into third party applications, such as HDFS or Amazon S3. The event handler framework is extensible allowing more event handlers performing different transformations or loading to different targets to be developed. Additionally, you can develop a custom event handler for your big data environment.

Oracle GoldenGate for Big Data provides two handlers to write to HDFS. Oracle recommends that you use the HDFS Handler or the File Writer Handler in the following situations:

The HDFS Event Handler is designed to stream data directly to HDFS.
No post write processing is occurring in HDFS. The HDFS Event Handler does not change the contents of the file, it simply uploads the existing file to HDFS.
Analytical tools are accessing data written to HDFS in real time including data in files that are open and actively being written to.

The File Writer Handler is designed to stage data to the local file system and then to load completed data files to HDFS when writing for a file is complete.
Analytic tools are not accessing data written to HDFS in real time.
Post write processing is occurring in HDFS to transform, reformat, merge, and move the data to a final location.
You want to write data files to HDFS in ORC or Parquet format.

Topics:

- Detailing the Functionality (page 5-2)
- Configuring the File Writer Handler (page 5-6)
- Review a Sample Configuration (page 5-14)

5.1.1 Detailing the Functionality

Topics:

- Using File Roll Events (page 5-2)
- Automatic Directory Creation (page 5-4)
- About the Active Write Suffix (page 5-4)
- Maintenance of State (page 5-4)
- Using Templated Strings (page 5-5)

5.1.1.1 Using File Roll Events

A **file roll event** occurs when writing to a specific data file is completed. No more data is written to that specific data file.

Finalize Action Operation

You can configure the finalize action operation to clean up a specific data file after a successful file roll action using the `finalizeaction` parameter with the following options:

`none`
Leave the data file in place (removing any active write suffix, see About the Active Write Suffix (page 5-4)).

`delete`
Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application).

move
Maintain the file name (removing any active write suffix), but move the file to the directory resolved using the `movePathMappingTemplate` property.

rename
Maintain the current directory, but rename the data file using the `fileRenameMappingTemplate` property.

move-rename
Rename the file using the file name generated by the `fileRenameMappingTemplate` property and move the file the file to the directory resolved using the `movePathMappingTemplate` property.

Typically, event handlers offer a subset of these same actions.

A sample Configuration of a finalize action operation:

```
gg.handlerlist=filewriter
#The File Writer Handler
gg.handler.filewriter.type=filewriter
gg.handler.filewriter.mode=op
gg.handler.filewriter.pathMappingTemplate=./dirout/evActParamS3R
gg.handler.filewriter.stateFileDirectory=./dirsta
gg.handler.filewriter.fileNameMappingTemplate=${fullyQualifiedTableName}_$
{currentTimestamp}.txt
gg.handler.filewriter.fileRollInterval=7m
gg.handler.filewriter.finalizeAction=delete
gg.handler.filewriter.inactivityRollInterval=7m
```

File Rolling Actions

Any of the following actions trigger a file roll event.

- A metadata change event.
- The maximum configured file size is exceeded
- The file roll interval is exceeded (the current time minus the time of first file write is greater than the file roll interval).
- The inactivity roll interval is exceeded (the current time minus the time of last file write is greater than the file roll interval).
- The File Writer Handler is configured to roll on shutdown and the Replicat process is stopped.

Operation Sequence

The file roll event triggers a sequence of operations to occur. It is important that you understand the order of the operations that occur when an individual data file is rolled:

1. The active data file is switched to inactive, the data file is flushed, and state data file is flushed.
2. The configured event handlers are called in the sequence that you specified.
3. The finalize action is executed on all the event handlers in the reverse order in which you configured them. Any finalize action that you configured is executed.
4. The finalize action is executed on the data file and the state file. If all actions are successful, the state file is removed. Any finalize action that you configured is executed.

For example, if you configured the File Writer Handler with the Parquet Event Handler and then the S3 Event Handler, the order for a roll event is:

1. The active data file is switched to inactive, the data file is flushed, and state data file is flushed.
2. The Parquet Event Handler is called to generate a Parquet file from the source data file.
3. The S3 Event Handler is called to load the generated Parquet file to S3.
4. The finalize action is executed on the S3 Parquet Event Handler. Any finalize action that you configured is executed.
5. The finalize action is executed on the Parquet Event Handler. Any finalize action that you configured is executed.
6. The finalize action is executed for the data file in the File Writer Handler

5.1.1.2 Automatic Directory Creation

You do not have to configure write directories before you execute the handler. The File Writer Handler checks to see if the specified write directory exists before creating a file and recursively creates directories as needed.

5.1.1.3 About the Active Write Suffix

A common use case is using a third party application to monitor the write directory to read data files. Third party application can only read a data file when writing to that file has completed. These applications need a way to determine if writing to a data file is active or complete. The File Writer Handler allows you to configure an **active write suffix** using this property:

```
gg.handler.name.fileWriteActiveSuffix=.tmp
```

The value of this property is appended to the generated file name. When writing to the file is complete, the data file is renamed and the active write suffix is removed from the file name. You can set your third party application to monitor your data file names to identify when the active write suffix is removed.

5.1.1.4 Maintenance of State

Previously, all Oracle GoldenGate for Big Data Handlers have been stateless. These stateless handlers only maintain state in the context of the Replicat process that it was running. If the Replicat process was stopped and restarted, then all the state was lost. With a Replicat restart, the handler began writing with no contextual knowledge of the previous run.

The File Writer Handler provides the ability of maintaining state between invocations of the Replicat process. By default with a restart:

- the state saved files are read,
- the state is restored,
- and appending active data files continues where the previous run stopped.

You can change this default action to require all files be rolled on shutdown by setting this property:

```
gg.handler.name.rollOnShutdown=true
```

5.1.1.5 Using Templated Strings

Templated strings can contain a combination of string constants and keywords that are dynamically resolved at runtime. The ORC Event Handler makes extensive use of templated strings to generate the ORC directory names, data file names, and ORC bucket names. These strings give you the flexibility to select where to write data files and the names of those data files. You should exercise caution when choosing file and directory names to avoid file naming collisions that can result in an abend.

Supported Templated Strings

Keyword	Description
`${fullyQualifiedTableName}`	The fully qualified source table name delimited by a period (`.`). For example, `MYCATALOG.MYSCHEMA.MYTABLE`.
`${catalogName}`	The individual source catalog name. For example, `MYCATALOG`.
`${schemaName}`	The individual source schema name. For example, `MYSCHEMA`.
`${tableName}`	The individual source table name. For example, `MYTABLE`.
`${groupName}`	The name of the Replicat process (with the thread number appended if you're using coordinated apply).
`${emptyString}`	Evaluates to an empty string. For example,`""`
`${operationCount}`	The total count of operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"1024"`.
`${insertCount}`	The total count of insert operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"125"`.
`${updateCount}`	The total count of update operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"265"`.
`${deleteCount}`	The total count of delete operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"11"`.
`${truncateCount}`	The total count of truncate operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"5"`.

Keyword	Description
${currentTimestamp}	The current timestamp. The default output format for the date time is yyyy-MM-dd_HH-mm-ss.SSS. For example, 2017-07-05_04-31-23.123. Alternatively, you can customize the format of the current timestamp by inserting the format inside square brackets like: ${currentTimestamp[MM-dd_HH]} This format uses the syntax defined in the Java SimpleDateFormat class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
${toUpperCase[]}	Converts the contents inside the square brackets to uppercase. For example, ${toUpperCase[${fullyQualifiedTableName}]}.
${toLowerCase[]}	Converts the contents inside the square brackets to lowercase. For example, ${toLowerCase[${fullyQualifiedTableName}]}.

Configuration of template strings can use a mix of keywords and static strings to assemble path and data file names at runtime.

Path Configuration Example

/usr/local/${*fullyQualifiedTableName*}

Data File Configuration Example

${*fullyQualifiedTableName*}_${currentTimestamp}_${*groupName*}.*handlerIndicator*

Requirements

The directory and file names generated using the templates must be legal on the system being written to. File names must be unique to avoid a file name collision. You can avoid a collision by adding a current timestamp using the ${currentTimestamp} keyword. If you are using coordinated apply, then adding ${groupName} into the data file name is recommended.

5.1.2 Configuring the File Writer Handler

Lists the configurable values for the File Writer Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file)

To enable the selection of the File Writer Handler, you must first configure the handler type by specifying gg.handler.jdbc.type=filewriter and the other File Writer properties as follows:

Table 5-1 File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.type`	Required	`filewriter`	None	Selects the File Writer Handler for use.
`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	Sets the maximum file size of files generated by the File Writer Handler. When the file size is exceeded, a roll event is triggered.

Table 5-1 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	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 a file is created. If the file is still open when the interval elapses then the a file roll event will be triggered.

Table 5-1 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`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`, `10m`, or `1.5h`. Values of `0` or less indicate that file rolling on time is turned off.	File inactivity rolling is turned off.	The timer starts from the latest write to a generated file. New writes to a generated file restart the counter. If the file is still open when the timer elapses a roll event is triggered..
`gg.handler.name.fileNameMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate File Writer Handler data file names at runtime.	None	Use keywords interlaced with constants to dynamically generate a unique path names at runtime. Typically, path names follow the format, `${fullyQualifiedTableName}_${groupName}_${currentTimeStamp}{.txt}`.

Table 5-1 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.pathMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the directory to which a file is written.	None	Use keywords interlaced with constants to dynamically generate a unique path names at runtime. Typically, path names follow the format, `${fullyQualifiedTableName}_${groupName}_${currentTimeStamp}{.txt}`.
`gg.handler.name.fileWriteActiveSuffix`	Optional	A string.	None	An optional suffix that is appended to files generated by the File Writer Handler to indicate that writing to the file is active. At the finalize action the suffix is removed.
`gg.handler.name.stateFileDirectory`	Required	A directory on the local machine to store the state files of the File Writer Handler.	None	Sets the directory on the local machine to store the state files of the File Writer Handler. The group name is appended to the directory to ensure that the functionality works when operating in a coordinated apply environment.
`gg.handler.name.rollOnShutdown`	Optional	`true | false`	`false`	Set to `true`, on normal shutdown of the Replicat process all open files are closed and a file roll event is triggered. If successful, the File Writer Handler has no state to carry over to a restart of the File Writer Handler.

Table 5-1 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.finalizeAction`	Optional	`none` | `delete` | `move` | `rename` | `move-rename`	`none`	Indicates what the File Writer Handler should do at the finalize action. **`none`** Leave the data file in place (removing any active write suffix, see About the Active Write Suffix (page 5-4)). **`delete`** Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application). **`move`** Maintain the file name (removing any active write suffix), but move the file to the directory resolved using the `movePathMappingTemplate` property. **`rename`** Maintain the current directory, but rename the data file using the `fileRenameMappingTemplate` property. **`move-rename`** Rename the file using the file name generated by the `fileRenameMappingTemplate` property and move the file the file to the directory resolved using the `movePathMappingTemplate` property.
`gg.handler.name.partitionByTable`	Optional	`true` | `false`	`true`	Set to `true` so that data from different source tables is partitioned into separate files. Set to `false` to interlace operation data from all source tables into a single output file. It cannot be set to `false` if the file format is the Avro OCF (Object Container File) format.
`gg.handler.name.eventHandler`	Optional	`HDFS` | `ORC` | `PARQUET` | `S3`	No event handler configured.	A unique string identifier cross referencing an event handler. The event handler will be invoked on the file roll event. Event handlers can do thing file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.

Table 5-1 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.fileRenameMappingTemplate`	Required if `gg.handler.name.finalizeAction` is set to `rename` or `move-rename`.	A string with resolvable keywords and constants used to dynamically generate File Writer Handler data file names for file renaming in the finalize action.	None.	Use keywords interlaced with constants to dynamically generate unique file names at runtime. Typically, file names follow the format, `${fullyQualifiedTableName}_${groupName}_${currentTimeStamp}{.txt}`.
`gg.handler.name.movePathMappingTemplate`	Required if `gg.handler.name.finalizeAction` is set to `rename` or `move-rename`.	A string with resolvable keywords and constants used to dynamically generate the directory to which a file is written.	None	Use keywords interlaced with constants to dynamically generate a unique path names at runtime. Typically, path names typically follow the format, `/ogg/data/${groupName}/${fullyQualifiedTableName}`.

Table 5-1 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format`	Required	`delimitedtext | json | json_row | xml | avro_row | avro_op | avro_row_ocf | avro_op_ocf`	`delimitedtext`	Selects the formatter for the HDFS Handler for how output data will be formatted **`delimitedtext`** Delimited text. **`json`** JSON **`json_row`** JSON output modeling row data **`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 (OCF) format. **`avro_op_ocf`** Avro in the more verbose format written into HDFS in the Avro OCF format. If you want to use the Parquet or ORC Event Handlers, then the selected format must be `avro_row_ocf` or `avro_op_ocf`.
`gg.handler.name.bom`	Optional	An even number of hex characters.	None	Enter an even number of hex characters where every two characters correspond to a single byte in the byte order mark (BOM). For example, the string `efbbbf` represents the 3-byte BOM for UTF-8.
`gg.handler.name.createControlFile`	Optional	`true | false`	`false`	Set to `true` to create a control file. A **control file** contains all of the completed file names including the path separated by a delimiter. The name of the control file is `{groupName}.control`. For example, if the Replicat process name is `fw`, then the control file name is `FW.control`.
`gg.handler.name.controlFileDelimiter`	Optional	Any string	new line (`\n`)	Allows you to control the delimiter separating file names in the control file. You can use`CDATA[]` wrapping with this property.

Table 5-1 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.controlFileDirectory`	Optional	A path to a directory to hold the control file.	A period (`.`) or the Oracle Golden Gate installation directory.	Set to specify where you want to write the control file.
`gg.handler.name.createOwnerFile`	Optional	`true` | `false`	`false`	Set to `true` to create an owner file. The **owner file** is created when the Replicat process starts and is removed when it terminates normally. The owner file allows other applications to determine if the process is running. The owner file remains in place when the Replicat process ends abnormally. The name of the owner file is the `{groupName}.owner`. For example, if the replicat process is name `fw`, then the owner file name is `FW.owner`. The file is create in the . directory or the Oracle GoldenGate installation directory.

`gg.handler.name.atTime`	Optional	One or more times to trigger a roll action of all open files.	None	Configure one or more trigger times in the following format:
				`HH:MM,HH:MM,HH:MM`
				
				Entries are based on a 24 hour clock. For example, an entry to configure rolled actions at three discrete times of day is:
				`gg.handler.fw.atTime=03:30,21:00,23:51`

5.1.3 Review a Sample Configuration

This File Writer Handler configuration example is using the Parquet Event Handler to convert data files to Parquet, and then for the S3 Event Handler to load Parquet files into S3:

```
gg.handlerlist=filewriter

#The handler properties
gg.handler.name.type=filewriter
gg.handler.name.mode=op
gg.handler.name.pathMappingTemplate=./dirout
gg.handler.name.stateFileDirectory=./dirsta
gg.handler.name.fileNameMappingTemplate=${fullyQualifiedTableName}_$
{currentTimestamp}.txt
gg.handler.name.fileRollInterval=7m
```

```
gg.handler.name.finalizeAction=delete
gg.handler.name.inactivityRollInterval=7m
gg.handler.name.format=avro_row_ocf
gg.handler.name.includetokens=true
gg.handler.name.partitionByTable=true
gg.handler.name.eventHandler=parquet
gg.handler.name.rollOnShutdown=true

gg.eventhandler.parquet.type=parquet
gg.eventhandler.parquet.pathMappingTemplate=./dirparquet
gg.eventhandler.parquet.writeToHDFS=false
gg.eventhandler.parquet.finalizeAction=delete
gg.eventhandler.parquet.eventHandler=s3
gg.eventhandler.parquet.fileNameMappingTemplate=${tableName}_$
{currentTimestamp}.parquet

gg.handler.filewriter.eventHandler=s3
gg.eventhandler.s3.type=s3
gg.eventhandler.s3.region=us-west-2
gg.eventhandler.s3.proxyServer=www-proxy.us.oracle.com
gg.eventhandler.s3.proxyPort=80
gg.eventhandler.s3.bucketMappingTemplate=tomsfunbucket
gg.eventhandler.s3.pathMappingTemplate=thepath
gg.eventhandler.s3.finalizeAction=none
```

5.2 Using the HDFS Event Handler

Learn how to use the HDFS Event Handler to load files generated by the File Writer Handler into HDFS.

See Using the File Writer Handler (page 5-1).

Topics:

- Detailing the Functionality (page 5-15)

5.2.1 Detailing the Functionality

Topics:

- Configuring the Handler (page 5-15)
- Using Templated Strings (page 5-16)
- Configuring the HDFS Event Handler (page 5-17)

5.2.1.1 Configuring the Handler

The HDFS Event Handler can can upload data files to HDFS. These additional configuration steps are required:

The HDFS Event Handler dependencies and considerations are the same as the HDFS Handler, see HDFS Additional Considerations (page 8-13).

Ensure that `gg.classpath` includes the HDFS client libraries.

Ensure that the directory containing the HDFS `core-site.xml` file is in `gg.classpath`. This is so the `core-site.xml` file can be read at runtime and the connectivity information to HDFS can be resolved. For example:

```
gg.classpath=/{HDFSinstallDirectory}/etc/hadoop
```

If Kerberos authentication is enabled on the HDFS cluster, you have to configure the Kerberos principal and the location of the `keytab` file so that the password can be resolved at runtime:

```
gg.eventHandler.name.kerberosPrincipal=principal
gg.eventHandler.name.kerberosKeytabFile=pathToTheKeytabFile
```

5.2.1.2 Using Templated Strings

Templated strings can contain a combination of string constants and keywords that are dynamically resolved at runtime. The HDFS Event Handler makes extensive use of templated strings to generate the HDFS directory names, data file names, and HDFS bucket names. This gives you the flexibility to select where to write data files and the names of those data files.

Supported Templated Strings

Keyword	Description
`${fullyQualifiedTableName}`	The fully qualified source table name delimited by a period (`.`). For example, `MYCATALOG.MYSCHEMA.MYTABLE`.
`${catalogName}`	The individual source catalog name. For example, `MYCATALOG`.
`${schemaName}`	The individual source schema name. For example, `MYSCHEMA`.
`${tableName}`	The individual source table name. For example, `MYTABLE`.
`${groupName}`	The name of the Replicat process (with the thread number appended if you're using coordinated apply).
`${emptyString}`	Evaluates to an empty string. For example,`""`
`${operationCount}`	The total count of operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"1024"`.
`${insertCount}`	The total count of insert operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"125"`.
`${updateCount}`	The total count of update operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"265"`.
`${deleteCount}`	The total count of delete operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"11"`.

Keyword	Description
${truncateCount}	The total count of truncate operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "5".
${currentTimestamp}	The current timestamp. The default output format for the date time is yyyy-MM-dd_HH-mm-ss.SSS. For example, 2017-07-05_04-31-23.123. Alternatively, you can customize the format of the current timestamp by inserting the format inside square brackets like: ${currentTimestamp[MM-dd_HH]} This format uses the syntax defined in the Java SimpleDateFormat class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
${toUpperCase[]}	Converts the contents inside the square brackets to uppercase. For example, ${toUpperCase[${fullyQualifiedTableName}]}.
${toLowerCase[]}	Converts the contents inside the square brackets to lowercase. For example, ${toLowerCase[${fullyQualifiedTableName}]}.

Configuration of template strings can use a mix of keywords and static strings to assemble path and data file names at runtime.

Path Configuration Example

```
/usr/local/${fullyQualifiedTableName}
```

Data File Configuration Example

```
${fullyQualifiedTableName}_${currentTimestamp}_${groupName}.handlerIndicator
```

5.2.1.3 Configuring the HDFS Event Handler

You configure the HDFS Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the HDFS Event Handler, you must first configure the handler type by specifying gg.eventhandler.jdbc.type=hdfs and the other HDFS Event properties as follows:

Table 5-2 HDFS Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.name.type	Required	hdfs	None	Selects the HDFS Event Handler for use.

Table 5-2 (Cont.) HDFS Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.pathMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in HDFS to write data files.	None	Use keywords interlaced with constants to dynamically generate a unique path names at runtime. Path names typically follow the format, `/ogg/data/${groupName}/${fullyQualifiedTableName}`.
`gg.eventhandler.name.fileNameMappingTemplate`	Optional	A string with resolvable keywords and constants used to dynamically generate the HDFS file name at runtime.	None	Use keywords interlaced with constants to dynamically generate a unique file names at runtime. If not set, the upstream file name is used.
`gg.eventhandler.name.finalizeAction`	Optional	`none | delete`	`none`	Indicates what the File Writer Handler should do at the finalize action. **`none`** Leave the data file in place (removing any active write suffix, see About the Active Write Suffix (page 5-4)). **`delete`** Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application).
`gg.eventhandler.name.kerberosPrincipal`	Optional	The Kerberos principal name.	None	Set to the Kerberos principal when HDFS Kerberos authentication is enabled.
`gg.eventhandler.name.keberosKeytabFile`	Optional	The path to the Keberos `keytab` file.	None	Set to the path to the Kerberos `keytab` file when HDFS Kerberos authentication is enabled.

Table 5-2 (Cont.) HDFS Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.eventHandler`	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	A unique string identifier cross referencing an event handler. The event handler will be invoked on the file roll event. Event handlers can do thing file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.

5.3 Using the Optimized Row Columnar Event Handler

Learn how to use the Optimized Row Columnar (ORC) Event Handler to generate data files in ORC format.

Topics:

- Overview (page 5-19)
- Detailing the Functionality (page 5-19)
- Configuring the ORC Event Handler (page 5-22)

5.3.1 Overview

ORC is a row columnar format that can substantially improve data retrieval times and the performance of Big Data analytics. You can use the ORC Event Handler to write ORC files to either a local file system or directly to HDFS. For information, see https://orc.apache.org/.

5.3.2 Detailing the Functionality

Topics:

- About the Upstream Data Format (page 5-19)
- About the Library Dependencies (page 5-20)
- Requirements (page 5-20)
- Using Templated Strings (page 5-20)

5.3.2.1 About the Upstream Data Format

The ORC Event Handler can only convert Avro Object Container File (OCF) generated by the File Writer Handler. The ORC Event Handler cannot convert other formats to ORC data files. The format of the File Writer Handler must be `avro_row_ocf` or `avro_op_ocf`, see Using the File Writer Handler (page 5-1).

5.3.2.2 About the Library Dependencies

Generating ORC files requires both the Apache ORC libraries and the HDFS client libraries, see Optimized Row Columnar Event Handler Client Dependencies (page J-1) and HDFS Handler Client Dependencies (page F-1).

Oracle GoldenGate for Big Data does not include the Apache ORC libraries nor does it include the HDFS client libraries. You must configure the `gg.classpath` variable to include the dependent libraries.

5.3.2.3 Requirements

The ORC Event Handler can write ORC files directly to HDFS. You must set the `writeToHDFS` property to `true`:

```
gg.eventhandler.orc.writeToHDFS=true
```

Ensure that the directory containing the HDFS `core-site.xml` file is in `gg.classpath`. This is so the `core-site.xml` file can be read at runtime and the connectivity information to HDFS can be resolved. For example:

```
gg.classpath=/{HDFS_install_directory}/etc/hadoop
```

If you enable Kerberos authentication is on the HDFS cluster, you have to configure the Kerberos principal and the location of the `keytab` file so that the password can be resolved at runtime:

```
gg.eventHandler.name.kerberosPrincipal=principal
gg.eventHandler.name.kerberosKeytabFile=path_to_the_keytab_file
```

5.3.2.4 Using Templated Strings

Templated strings can contain a combination of string constants and keywords that are dynamically resolved at runtime. The ORC Event Handler makes extensive use of templated strings to generate the ORC directory names, data file names, and ORC bucket names. This gives you the flexibility to select where to write data files and the names of those data files.

Supported Templated Strings

Keyword	Description
`${fullyQualifiedTableName}`	The fully qualified source table name delimited by a period (`.`). For example, `MYCATALOG.MYSCHEMA.MYTABLE`.
`${catalogName}`	The individual source catalog name. For example, `MYCATALOG`.
`${schemaName}`	The individual source schema name. For example, `MYSCHEMA`.
`${tableName}`	The individual source table name. For example, `MYTABLE`.
`${groupName}`	The name of the Replicat process (with the thread number appended if you're using coordinated apply).

Keyword	Description
`${emptyString}`	Evaluates to an empty string. For example,`""`
`${operationCount}`	The total count of operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"1024"`.
`${insertCount}`	The total count of insert operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"125"`.
`${updateCount}`	The total count of update operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"265"`.
`${deleteCount}`	The total count of delete operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"11"`.
`${truncateCount}`	The total count of truncate operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"5"`.
`${currentTimestamp}`	The current timestamp. The default output format for the date time is `yyyy-MM-dd_HH-mm-ss.SSS`. For example, `2017-07-05_04-31-23.123`. Alternatively, you can customize the format of the current timestamp by inserting the format inside square brackets like: `${currentTimestamp[MM-dd_HH]}` This format uses the syntax defined in the Java `SimpleDateFormat` class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
`${toUpperCase[]}`	Converts the contents inside the square brackets to uppercase. For example, `${toUpperCase[${fullyQualifiedTableName}]}`.
`${toLowerCase[]}`	Converts the contents inside the square brackets to lowercase. For example, `${toLowerCase[${fullyQualifiedTableName}]}`.

Configuration of template strings can use a mix of keywords and static strings to assemble path and data file names at runtime.

Path Configuration Example

`/usr/local/${fullyQualifiedTableName}`

Data File Configuration Example

`${fullyQualifiedTableName}_${currentTimestamp}_${groupName}.handlerIndicator`

5.3.3 Configuring the ORC Event Handler

You configure the ORC Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

The ORC Event Handler works only in conjunction with the File Writer Handler.

To enable the selection of the ORC Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=orc` and the other ORC properties as follows:

Table 5-3 ORC Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.type`	Required	`ORC`	None	Selects the ORC Event Handler.
`gg.eventhandler.name.writeToHDFS`	Optional	`true | false`	`false`	The ORC framework allows direct writing to HDFS. Set to `false` to write to the local file system. Set to `true` to write directly to HDFS.
`gg.eventhandler.name.pathMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in the ORC bucket to write the file.	None	Use keywords interlaced with constants to dynamically generate a unique ORC path names at runtime. Typically, path names follow the format, `/ogg/data/${groupName}/${fullyQualifiedTableName}`.
`gg.eventhandler.name.fileMappingTemplate`	Optional	A string with resolvable keywords and constants used to dynamically generate the ORC file name at runtime.	None	Use resolvable keywords and constants used to dynamically generate the ORC data file name at runtime. If not set, the upstream file name is used.
`gg.eventhandler.name.compressionCodec`	Optional	`LZ4 | LZO | NONE | SNAPPY | ZLIB`	`NONE`	Sets the compression codec of the generated ORC file.

Table 5-3 (Cont.) ORC Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.finalizeAction`	Optional	`none | delete`	`none`	Set to `none` to leave the ORC data file in place on the finalize action. Set to `delete` if you want to delete the ORC data file with the finalize action.
`gg.eventhandler.name.kerberosPrincipal`	Optional	The Kerberos principal name.	None	Sets the Kerberos principal when writing directly to HDFS and Kerberos authentication is enabled.
`gg.eventhandler.name.keberosKeytabFile`	Optional	The path to the Keberos `keytab` file.	`none`	Sets the path to the Kerberos `keytab` file with writing directly to HDFS and Kerberos authentication is enabled.
`gg.eventhandler.name.blockPadding`	Optional	`true | false`	`true`	Set to `true` to enable block padding in generated ORC files or `false` to disable.
`gg.eventhandler.name.blockSize`	Optional	`long`	The ORC default.	Sets the block size of generated ORC files.
`gg.eventhandler.name.bufferSize`	Optional	`integer`	The ORC default.	Sets the buffer size of generated ORC files.
`gg.eventhandler.name.encodingStrategy`	Optional	`COMPRESSION | SPEED`	The ORC default.	Set if the ORC encoding strategy is optimized for compression or for speed..
`gg.eventhandler.name.paddingTolerance`	Optional	A percentage represented as a floating point number.	The ORC default.	Sets the percentage for padding tolerance of generated ORC files.
`gg.eventhandler.name.rowIndexStride`	Optional	`integer`	The ORC default.	Sets the row index stride of generated ORC files.
`gg.eventhandler.name.stripeSize`	Optional	`integer`	The ORC default.	Sets the stripe size of generated ORC files.
`gg.eventhandler.name.eventHandler`	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	The event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3 or HDFS.

5.4 Using the Oracle Cloud Infrastructure Event Handler

Learn how to use the Oracle Cloud Infrastructure Event Handler to load files generated by the File Writer Handler into an Oracle Cloud Infrastructure Object Store.

Topics:

- Overview (page 5-24)
- Detailing the Functionality (page 5-24)
- Configuring the Oracle Cloud Infrastructure Event Handler (page 5-24)
- Configuring Credentials for Oracle Cloud Infrastructure (page 5-27)
- Using Templated Strings (page 5-28)
- Troubleshooting (page 5-30)

5.4.1 Overview

The Oracle Cloud Infrastructure Object Storage service is an internet-scale, high-performance storage platform that offers reliable and cost-efficient data durability. The Object Storage service can store an unlimited amount of unstructured data of any content type, including analytic data and rich content, like images and videos, see https://cloud.oracle.com/en_US/cloud-infrastructure.

You can use any format handler that the File Writer Handler supports.

5.4.2 Detailing the Functionality

The Oracle Cloud Infrastructure Event Handler requires the Oracle Cloud Infrastructure Java software development kit (SDK) to transfer files to Oracle Cloud Infrastructure Object Storage. Oracle GoldenGate for Big Data does not include the Oracle Cloud Infrastructure Java SDK, seehttps://docs.cloud.oracle.com/iaas/Content/API/Concepts/sdkconfig.htm.

You must download the Oracle Cloud Infrastructure Java SDK at:

https://docs.us-phoenix-1.oraclecloud.com/Content/API/SDKDocs/javasdk.htm

Extract the JAR files to a permanent directory. There are two directories required by the handler, the JAR library directory that has Oracle Cloud Infrastructure SDK JAR and a third-party JAR library. Both directories must be in the `gg.classpath`.

Specify the `gg.classpath` environment variable to include the JAR files of the Oracle Cloud Infrastructure Java SDK.

Example

```
gg.classpath=/usr/var/oci/lib/*:/usr/var/oci/third-party/lib/*
```

5.4.3 Configuring the Oracle Cloud Infrastructure Event Handler

You configure the Oracle Cloud Infrastructure Event Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

The Oracle Cloud Infrastructure Event Handler works only in conjunction with the File Writer Handler.

To enable the selection of the Oracle Cloud Infrastructure Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=oci` and the other Oracle Cloud Infrastructure properties as follows:

Table 5-4 Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.type`	Required	`oci`	None	Selects the Oracle Cloud Infrastructure Event Handler.
`gg.eventhandler.name.configFilePath`	Required	Path to the event handler `config` file.	None	The configuration file name and location.
`gg.eventhandler.name.profile`	Required	Valid string representing the profile name.	None	In the Oracle Cloud Infrastructure `config` file, the entries are identified by the profile name. The default profile is `DEFAULT`. You can have an additional profile like `ADMIN_USER`. Any value that isn't explicitly defined for the `ADMIN_USER` profile (or any other profiles that you add to the `config` file) is inherited from the `DEFAULT` profile.
`gg.eventhandler.name.namespace`	Required	Oracle Cloud Infrastructure namespace.	None	The namespace serves as a top-level container for all buckets and objects and allows you to control bucket naming within user's tenancy. The Object Storage namespace is a system-generated string assigned during account creation. Your namespace string is listed in Object Storage Settings while using the Oracle Cloud Infrastructure Console.

`gg.eventhandler.name.region`	Required	Oracle Cloud Infrastructure region	None	Oracle Cloud Infrastructure Servers and Data is hosted in a region and is a localized geographic area. There are four supported regions. For example:
				`London Heathrow("uk-london-1")`
				`Frankfurt("eu-frankfurt-1")`
				`Ashburn("us-ashburn-1")`
				`Phoenix("us-phoenix-1").`

`gg.eventhandler.name.compartmentID`	Required	Valid compartment id.	None	A compartment is a logical container to organize Oracle Cloud Infrastructure resources. The `compartmentID` is listed in Bucket Details while using the Oracle Cloud Infrastructure Console.

Table 5-4 (Cont.) Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.pathMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in the Oracle Cloud Infrastructure bucket to write the file.	None	Use keywords interlaced with constants to dynamically generate unique Oracle Cloud Infrastructure path names at runtime.
`gg.eventhandler.name.fileMappingTemplate`	Optional	A string with resolvable keywords and constants used to dynamically generate the Oracle Cloud Infrastructure file name at runtime.	None	Use resolvable keywords and constants to dynamically generate the Oracle Cloud Infrastructure data file name at runtime. If not set, the upstream file name is used.
`gg.eventhandler.name.bucketMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in the Oracle Cloud Infrastructure bucket to write the file.	None	Use resolvable keywords and constants used to dynamically generate the Oracle Cloud Infrastructure bucket name at runtime. The event handler attempts to create the Oracle Cloud Infrastructure bucket if it does not exist.

Table 5-4 (Cont.) Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.finalizeAction`	Optional	`none` | `delete`	`none`	Set to `none` to leave the Oracle Cloud Infrastructure data file in place on the finalize action. Set to `delete` if you want to delete the Oracle Cloud Infrastructure data file with the finalize action.
`gg.eventhandler.name.eventHandler`	Optional	A unique string identifier cross referencing a child event handler.	No event handler is configured.	Sets the event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3, converting to Parquet or ORC format, loading files to HDFS, loading files to Oracle Cloud Infrastructure Storage Classic, or loading file to Oracle Cloud Infrastructure.

Sample Configuration

```
gg.eventhandler.oci.type=oci
gg.eventhandler.oci.configFilePath=~/.oci/config
gg.eventhandler.oci.profile=DEFAULT
gg.eventhandler.oci.namespace=dwcsdemo
gg.eventhandler.oci.region=us-ashburn-1
gg.eventhandler.oci.compartmentID=ocid1.compartment.oc1..aaaaaaaajdg6iblwgqlyqpegf6kw
dais2gyx3guspboa7fsi72tfihz2wrba
gg.eventhandler.oci.pathMappingTemplate=${schemaName}
gg.eventhandler.oci.bucketMappingTemplate=${schemaName}
gg.eventhandler.oci.fileNameMappingTemplate=${tableName}_${currentTimestamp}.txt
gg.eventhandler.oci.finalizeAction=NONE
```

5.4.4 Configuring Credentials for Oracle Cloud Infrastructure

Basic configuration information like user credentials and tenancy Oracle Cloud IDs (OCIDs) of Oracle Cloud Infrastructure is required for the Java SDKs to work, see https://docs.cloud.oracle.com/iaas/Content/General/Concepts/identifiers.htm.

The ideal configuration file include keys `user`, `fingerprint`, `key_file`, `tenancy`, and `region` with their respective values. The default configuration file name and location is `~/.oci/config`.

Create the `config` file as follows:

1. Create a directory called `.oci` in the Oracle GoldenGate for Big Data home directory
2. Create a text file and name it `config`.
3. Obtain the values for these properties:

`user`

a. Login to the Oracle Cloud Infrastructure Console https://console.us-ashburn-1.oraclecloud.com.

b. Click **Username**.

c. Click **User Settings**.

   The User's OCID is displayed and is the value for the key user.

`tenancy`

The Tenancy ID is displayed at the bottom of the Console page.

`region`

The region is displayed with the header session drop-down menu in the Console.

`fingerprint`

To generate the fingerprint, use the *How to Get the Key's Fingerprint* instructions at:

https://docs.cloud.oracle.com/iaas/Content/API/Concepts/apisigningkey.htm

`key_file`

You need to share the public and private key to establish a connection with Oracle Cloud Infrastructure. To generate the keys, use the *How to Generate an API Signing Key*at:

https://docs.cloud.oracle.com/iaas/Content/API/Concepts/apisigningkey.htm

Sample config File

```
user=ocid1.user.oc1..aaaaaaaat5nvwcna5j6aqzqedqw3rynjq
fingerprint=20:3b:97:13::4e:c5:3a:34
key_file=~/.oci/oci_api_key.pem
tenancy=ocid1.tenancy.oc1..aaaaaaaaba3pv6wkcr44h25vqstifs
```

5.4.5 Using Templated Strings

Templated strings can contain a combination of string constants and keywords that are dynamically resolved at runtime. This event handler makes extensive use of templated strings to generate the Oracle Cloud Infrastructure directory names, data file names, and Oracle Cloud Infrastructure bucket names. These strings give you the flexibility to select where to write data files and the names of those data files. You should exercise caution when choosing file and directory names to avoid file naming collisions that can result in an abend.

Supported Templated Strings

Keyword	Description
`${fullyQualifiedTableName}`	The fully qualified source table name delimited by a period (`.`). For example, `MYCATALOG.MYSCHEMA.MYTABLE`.
`${catalogName}`	The individual source catalog name. For example, `MYCATALOG`.
`${schemaName}`	The individual source schema name. For example, `MYSCHEMA`.

Keyword	Description
${tableName}	The individual source table name. For example, MYTABLE.
${groupName}	The name of the Replicat process (with the thread number appended if you're using coordinated apply).
${emptyString}	Evaluates to an empty string. For example,""
${operationCount}	The total count of operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "1024".
${insertCount}	The total count of insert operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "125".
${updateCount}	The total count of update operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "265".
${deleteCount}	The total count of delete operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "11".
${truncateCount}	The total count of truncate operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "5".
${currentTimestamp}	The current timestamp. The default output format for the date time is yyyy-MM-dd_HH-mm-ss.SSS. For example, 2017-07-05_04-31-23.123. Alternatively, you can customize the format of the current timestamp by inserting the format inside square brackets like: ${currentTimestamp[MM-dd_HH]} This format uses the syntax defined in the Java SimpleDateFormat class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
${toUpperCase[]}	Converts the contents inside the square brackets to uppercase. For example, ${toUpperCase[${fullyQualifiedTableName}]}.
${toLowerCase[]}	Converts the contents inside the square brackets to lowercase. For example, ${toLowerCase[${fullyQualifiedTableName}]}.

Configuration of template strings can use a mix of keywords and static strings to assemble path and data file names at runtime.

Path Configuration Example

```
/usr/local/${fullyQualifiedTableName}
```

Data File Configuration Example

```
${fullyQualifiedTableName}_${currentTimestamp}_${groupName}.handlerIndicator
```

Requirements

The directory and file names generated using the templates must be legal on the system being written to. File names must be unique to avoid a file name collision. You can avoid a collision by adding a current timestamp using the `${currentTimestamp}` keyword. If you are using coordinated apply, then adding `${groupName}` into the data file name is recommended.

5.4.6 Troubleshooting

Connectivity Issues

If the event handler is unable to connect to the Oracle Cloud Infrastructure Classic Object when running on-premise, it's likely your connectivity to the public internet is protected by a proxy server. Proxy servers act a gateway between the private network of a company and the public internet. Contact your network administrator to get the URLs of your proxy server, and then setup up a proxy server.

Oracle GoldenGate for Big Data can be used with a proxy server using the following Java run time arguments to enable the proxy server as in this example:

```
-Dhttps.proxyHost=www-proxy.us.company.com
-Dhttps.proxyPort=80
```

ClassNotFoundException Error

The most common initial error is an incorrect classpath that does not include all the required client libraries so results in a `ClassNotFoundException` error. Specify the `gg.classpath` variable to include all of the required JAR files for the Oracle Cloud Infrastructure Java SDK, see Detailing the Functionality (page 5-24).

5.5 Using the Oracle Cloud Infrastructure Classic Event Handler

Learn how to use the Oracle Cloud Infrastructure Classic Event Handler to load files generated by the File Writer Handler into an Oracle Cloud Infrastructure Classic Object Store.

Topics:

- Overview (page 5-31)
- Detailing the Functionality (page 5-31)
- Configuring the Oracle Cloud Infrastructure Classic Event Handler (page 5-31)
- Using Templated Strings (page 5-34)
- Troubleshooting (page 5-35)

5.5.1 Overview

The Oracle Cloud Infrastructure Object Classic service is an Infrastructure as a Service (IaaS) product that provides an enterprise-grade, large-scale, object storage solution for files and unstructured data., see https://cloud.oracle.com/storage-classic.

You can use any format handler that the File Writer Handler supports.

5.5.2 Detailing the Functionality

The Oracle Cloud Infrastructure Classic Event Handler requires File Transfer Manager (FTM), a Java SDK to transfer files to Oracle Cloud Infrastructure Classic. Oracle GoldenGate for Big Data does not include the FTM Java SDK. .

You must download the FTM Java SDK at:

http://www.oracle.com/technetwork/topics/cloud/downloads/index.html#storejavasdk

Extract the JAR files to a permanent directory. There are two directories required by the handler, the JAR library directory that has Oracle Cloud Infrastructure SDK JAR and a third-party JAR library. Both directories must be in the `gg.classpath`.

Specify the `gg.classpath` environment variable to include the JAR files of the FTM Java SDK.

These are the required third-party JARs:

```
ftm-api-2.4.4.jar
javax.json-1.0.4.jar
slf4j-api-1.7.7.jar
slf4j-log4j12-1.7.7.jar
log4j-1.2.17-16.jar
low-level-api-core-1.14.22.jar
```

Example

```
gg.classpath=/usr/var/ftm-sdk/libs/*:
```

5.5.3 Configuring the Oracle Cloud Infrastructure Classic Event Handler

You configure the Oracle Cloud Infrastructure Classic Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

The Oracle Cloud Infrastructure Classic Event Handler works only in conjunction with the File Writer Handler.

To enable the selection of the Oracle Cloud Infrastructure Classic Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=oci-c` and the other Oracle Cloud Infrastructure properties as follows:

Table 5-5 Oracle Cloud Infrastructure Classic Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.type`	Required	`oci-c`	None	Selects the Oracle Cloud Infrastructure Classic Event Handler.
`gg.eventhandler.name.serverUrl`	Required	Server URL	None	The server URL for the Oracle Cloud Infrastructure Classic Event Handler.
`gg.eventhandler.name.tenantID`	Required	Valid string representing `tenantID`.	None	The case-sensitive tenant id that you specify when signing in to the Oracle Cloud Infrastructure Console.
`gg.eventhandler.name.serviceName`	Required	The Oracle Cloud Infrastructure Classic Event Handler service instance name.	None	The Oracle Cloud Infrastructure Classic Event Handler service instance name that you specified.
`gg.eventhandler.name.username`	Required	Valid user name.	None	The user name for the Oracle Cloud Infrastructure user account.
`gg.eventhandler.name.password`	Required	Valid password.	None	The password for the Oracle Cloud Infrastructure user account.
`gg.eventhandler.name.pathMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in the Oracle Cloud Infrastructure bucket to write the file.	None	Use resolvable keywords and constants to dynamically generate a unique Oracle Cloud Infrastructure Classic path names at runtime.

Table 5-5 (Cont.) Oracle Cloud Infrastructure Classic Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.containerMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in the Oracle Cloud Infrastructure container to write the file.	None	Use resolvable keywords and constants used to dynamically generate the Oracle Cloud Infrastructure container name at runtime. The event handler attempts to create the Oracle Cloud Infrastructure container if it does not exist.
`gg.eventhandler.name.fileMappingTemplate`	Optional	A string with resolvable keywords and constants used to dynamically generate the Oracle Cloud Infrastructure file name at runtime.	None	Use resolvable keywords and constants used to dynamically generate the Oracle Cloud Infrastructure file name at runtime. If not set, the upstream file name is used.
`gg.eventhandler.name.finalizeAction`	Optional	`none` | `delete`	`none`	Set to `none` to leave the Oracle Cloud Infrastructure Classic data file in place on the finalize action. Set to `delete` if you want to delete the Oracle Cloud Infrastructure Classic data file with the finalize action.
`gg.eventhandler.name.eventHandler`	Optional	A unique string identifier cross referencing a child event handler.	No event handler is configured.	Sets the event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3, converting to Parquet or ORC format, loading files to HDFS, loading files to Oracle Cloud Infrastructure Classic, or loading file to Oracle Cloud Infrastructure.

Sample Configuration

```
#The OCI-C Event handler
gg.eventhandler.oci-c.type=oci-c
```

```
gg.eventhandler.oci-c.serverURL=https://storage.companycloud.com/
gg.eventhandler.oci-c.tenantID=usoraclebig
gg.eventhandler.oci-c.serviceName=dev1
gg.eventhandler.oci-c.username=user@company.com
gg.eventhandler.oci-c.password=pass
gg.eventhandler.oci-c.pathMappingTemplate=${schemaName}
gg.eventhandler.oci-c.containerMappingTemplate=${schemaName}
gg.eventhandler.oci-c.fileNameMappingTemplate=${tableName}_${currentTimestamp}.json
gg.eventhandler.oci-c.finalizeAction=NONE
```

5.5.4 Using Templated Strings

Templated strings can contain a combination of string constants and keywords that are dynamically resolved at runtime. This event handler makes extensive use of templated strings to generate the Oracle Cloud Infrastructure directory names, data file names, and Oracle Cloud Infrastructure bucket names. These strings give you the flexibility to select where to write data files and the names of those data files. You should exercise caution when choosing file and directory names to avoid file naming collisions that can result in an abend.

Supported Templated Strings

Keyword	Description
`${fullyQualifiedTableName}`	The fully qualified source table name delimited by a period (`.`). For example, `MYCATALOG.MYSCHEMA.MYTABLE`.
`${catalogName}`	The individual source catalog name. For example, `MYCATALOG`.
`${schemaName}`	The individual source schema name. For example, `MYSCHEMA`.
`${tableName}`	The individual source table name. For example, `MYTABLE`.
`${groupName}`	The name of the Replicat process (with the thread number appended if you're using coordinated apply).
`${emptyString}`	Evaluates to an empty string. For example,`""`
`${operationCount}`	The total count of operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"1024"`.
`${insertCount}`	The total count of insert operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"125"`.
`${updateCount}`	The total count of update operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"265"`.
`${deleteCount}`	The total count of delete operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"11"`.

Keyword	Description
`${truncateCount}`	The total count of truncate operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"5"`.
`${currentTimestamp}`	The current timestamp. The default output format for the date time is `yyyy-MM-dd_HH-mm-ss.SSS`. For example, `2017-07-05_04-31-23.123`. Alternatively, you can customize the format of the current timestamp by inserting the format inside square brackets like: `${currentTimestamp[MM-dd_HH]}` This format uses the syntax defined in the Java `SimpleDateFormat` class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
`${toUpperCase[]}`	Converts the contents inside the square brackets to uppercase. For example, `${toUpperCase[${fullyQualifiedTableName}]}`.
`${toLowerCase[]}`	Converts the contents inside the square brackets to lowercase. For example, `${toLowerCase[${fullyQualifiedTableName}]}`.

Configuration of template strings can use a mix of keywords and static strings to assemble path and data file names at runtime.

Path Configuration Example

`/usr/local/${fullyQualifiedTableName}`

Data File Configuration Example

`${fullyQualifiedTableName}_${currentTimestamp}_${groupName}.handlerIndicator`

Requirements

The directory and file names generated using the templates must be legal on the system being written to. File names must be unique to avoid a file name collision. You can avoid a collision by adding a current timestamp using the `${currentTimestamp}` keyword. If you are using coordinated apply, then adding `${groupName}` into the data file name is recommended.

5.5.5 Troubleshooting

Connectivity Issues

If the event handler is unable to connect to the Oracle Cloud Infrastructure Classic Object when running on-premise, it's likely your connectivity to the public internet is protected by a proxy server. Proxy servers act a gateway between the private network of a company and the public internet. Contact your network administrator to get the URLs of your proxy server, and then setup up a proxy server.

Oracle GoldenGate for Big Data can be used with a proxy server using the following Java run time arguments to enable the proxy server as in this example:

```
-Dhttps.proxyHost=www-proxy.us.company.com
-Dhttps.proxyPort=80
```

ClassNotFoundException Error

The most common initial error is an incorrect classpath that does not include all the required client libraries so results in a `ClassNotFoundException` error. Specify the `gg.classpath` variable to include all of the required JAR files for the Oracle Cloud Infrastructure Java SDK, see Detailing the Functionality (page 5-24).

5.6 Using the Parquet Event Handler

Learn how to use the Parquet Event Handler to load files generated by the File Writer Handler into HDFS.

See Using the File Writer Handler (page 5-1).

Topics:

- Overview (page 5-36)
- Detailing the Functionality (page 5-36)
- Configuring the Parquet Event Handler (page 5-38)

5.6.1 Overview

The Parquet Event Handler enables you to generate data files in Parquet format. Parquet files can be written to either the local file system or directly to HDFS. Parquet is a columnar data format that can substantially improve data retrieval times and improve the performance of Big Data analytics, see https://parquet.apache.org/.

5.6.2 Detailing the Functionality

Topics:

- Configuring the Parquet Event Handler to Write to HDFS (page 5-36)
- About the Upstream Data Format (page 5-37)
- Using Templated Strings (page 5-37)

5.6.2.1 Configuring the Parquet Event Handler to Write to HDFS

The Apache Parquet framework supports writing directly to HDFS. The Parquet Event Handler can write Parquet files directly to HDFS. These additional configuration steps are required:

The Parquet Event Handler dependencies and considerations are the same as the HDFS Handler, see HDFS Additional Considerations (page 8-13).

Set the `writeToHDFS` property to `true`:

```
gg.eventhandler.parquet.writeToHDFS=true
```

Ensure that `gg.classpath` includes the HDFS client libraries.

Ensure that the directory containing the HDFS `core-site.xml` file is in `gg.classpath`. This is so the `core-site.xml` file can be read at runtime and the connectivity information to HDFS can be resolved. For example:

```
gg.classpath=/{HDFS_install_directory}/etc/hadoop
```

If Kerberos authentication is enabled on the HDFS cluster, you have to configure the Kerberos principal and the location of the `keytab` file so that the password can be resolved at runtime:

```
gg.eventHandler.name.kerberosPrincipal=principal
gg.eventHandler.name.kerberosKeytabFile=path_to_the_keytab_file
```

5.6.2.2 About the Upstream Data Format

The Parquet Event Handler can only convert Avro Object Container File (OCF) generated by the File Writer Handler. The Parquet Event Handler cannot convert other formats to Parquet data files. The format of the File Writer Handler must be `avro_row_ocf` or `avro_op_ocf`, see Using the File Writer Handler (page 5-1).

5.6.2.3 Using Templated Strings

Templated strings can contain a combination of string constants and keywords that are dynamically resolved at runtime. The Parquet Event Handler makes extensive use of templated strings to generate the HDFS directory names, data file names, and HDFS bucket names. This gives you the flexibility to select where to write data files and the names of those data files.

Supported Templated Strings

Keyword	Description
`${fullyQualifiedTableName}`	The fully qualified source table name delimited by a period (`.`). For example, `MYCATALOG.MYSCHEMA.MYTABLE`.
`${catalogName}`	The individual source catalog name. For example, `MYCATALOG`.
`${schemaName}`	The individual source schema name. For example, `MYSCHEMA`.
`${tableName}`	The individual source table name. For example, `MYTABLE`.
`${groupName}`	The name of the Replicat process (with the thread number appended if you're using coordinated apply).
`${emptyString}`	Evaluates to an empty string. For example,`""`
`${operationCount}`	The total count of operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"1024"`.
`${insertCount}`	The total count of insert operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"125"`.

Keyword	Description
`${updateCount}`	The total count of update operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"265"`.
`${deleteCount}`	The total count of delete operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"11"`.
`${truncateCount}`	The total count of truncate operations in the data file. It must be used either on rename or by the event handlers or it will be zero (`0`) because nothing is written yet. For example, `"5"`.
`${currentTimestamp}`	The current timestamp. The default output format for the date time is `yyyy-MM-dd_HH-mm-ss.SSS`. For example, `2017-07-05_04-31-23.123`. Alternatively, you can customize the format of the current timestamp by inserting the format inside square brackets like: `${currentTimestamp[MM-dd_HH]}` This format uses the syntax defined in the Java `SimpleDateFormat` class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
`${toUpperCase[]}`	Converts the contents inside the square brackets to uppercase. For example, `${toUpperCase[${fullyQualifiedTableName}]}`.
`${toLowerCase[]}`	Converts the contents inside the square brackets to lowercase. For example, `${toLowerCase[${fullyQualifiedTableName}]}`.

Configuration of template strings can use a mix of keywords and static strings to assemble path and data file names at runtime.

Path Configuration Example

`/usr/local/${fullyQualifiedTableName}`

Data File Configuration Example

`${fullyQualifiedTableName}_${currentTimestamp}_${groupName}.handlerIndicator`

5.6.3 Configuring the Parquet Event Handler

You configure the Parquet Event Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

The Parquet Event Handler works only in conjunction with the File Writer Handler.

To enable the selection of the Parquet Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=parquet` and the other Parquet Event properties as follows:

Table 5-6 Parquet Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.type`	Required	`parquet`	None	Selects the Parquet Event Handler for use.
`gg.eventhandler.name.writeToHDFS`	Optional	`true | false`	`false`	Set to `false` to write to the local file system. Set to `true` to write directly to HDFS.
`gg.eventhandler.name.pathMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path to write generated Parquet files.	None	Use keywords interlaced with constants to dynamically generate a unique path names at runtime. Typically, path names follow the format, `/ogg/data/${groupName}/${fullyQualifiedTableName}`.
`gg.eventhandler.name.fileNameMappingTemplate`	Optional	A string with resolvable keywords and constants used to dynamically generate the Parquet file name at runtime	None	Sets the Parquet file name. If not set, the upstream file name is used.
`gg.eventhandler.name.compressionCodec`	Optional	`GZIP | LZO | SNAPPY | UNCOMPRESSED`	`UNCOMPRESSED`	Sets the compression codec of the generated Parquet file.

Table 5-6 (Cont.) Parquet Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.finalizeAction`	Optional	`none | delete`	`none`	Indicates what the Parquet Event Handler should do at the finalize action. **`none`** Leave the data file in place. **`delete`** Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application).
`gg.eventhandler.name.dictionaryEncoding`	Optional	`true | false`	The Parquet default.	Set to `true` to enable Parquet dictionary encoding.
`gg.eventhandler.name.validation`	Optional	`true | false`	The Parquet default.	Set to `true` to enable Parquet validation.
`gg.eventhandler.name.dictionaryPageSize`	Optional	Integer	The Parquet default.	Sets the Parquet dictionary page size.
`gg.eventhandler.name.maxPaddingSize`	Optional	Integer	The Parquet default.	Sets the Parquet padding size.
`gg.eventhandler.name.pageSize`	Optional	Integer	The Parquet default.	Sets the Parquet page size.
`gg.eventhandler.name.rowGroupSize`	Optional	Integer	The Parquet default.	Sets the Parquet row group size.
`gg.eventhandler.name.kerberosPrincipal`	Optional	The Kerberos principal name.	None	Set to the Kerberos principal when writing directly to HDFS and Kerberos authentication is enabled.
`gg.eventhandler.name.keberosKeytabFile`	Optional	The path to the Keberos `keytab` file.	The Parquet default.	Set to the path to the Kerberos `keytab` file with writing directly to HDFS and Kerberos authentication is enabled.

Table 5-6 (Cont.) Parquet Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.eventHandler`	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	The event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.

5.7 Using the S3 Event Handler

Learn how to use the S3 Event Handler, which provides the interface to Amazon S3 web services.

Topics:

- Overview (page 5-41)
- Detailing Functionality (page 5-41)
- Configuring the S3 Event Handler (page 5-44)

5.7.1 Overview

Amazon S3 is object storage hosted in the Amazon cloud. The purpose of the S3 Event Handler is to load data files generated by the File Writer Handler into Amazon S3, see https://aws.amazon.com/s3/.

You can use any format that the File Writer Handler, see Using the File Writer Handler (page 5-1).

5.7.2 Detailing Functionality

The S3 Event Handler requires the Amazon Web Services (AWS) Java SDK to transfer files to S3 object storage.Oracle GoldenGate for Big Data does not include the AWS Java SDK. You have to download and install the AWS Java SDK from:

https://aws.amazon.com/sdk-for-java/

Then you have to configure the `gg.classpath` variable to include the JAR files in the AWS Java SDK and are divided into two directories. Both directories must be in `gg.classpath`, for example:

```
gg.classpath=/usr/var/aws-java-sdk-1.11.240/lib/*:/usr/var/aws-java-sdk-1.11.240/
third-party/lib/
```

Topics:

- Configuring the Client ID and Secret (page 5-42)
- About the AWS S3 Buckets (page 5-42)

- Using Templated Strings (page 5-42)
- Troubleshooting (page 5-44)

5.7.2.1 Configuring the Client ID and Secret

A client ID and secret are the required credentials for the S3 Event Handler to interact with Amazon S3. A client ID and secret are generated using the Amazon AWS website. The retrieval of these credentials and presentation to the S3 server are performed on the client side by the AWS Java SDK. The AWS Java SDK provides multiple ways that the client ID and secret can be resolved at runtime.

The client ID and secret can be set as Java properties in the Java Adapter properties file as follows:

```
 javawriter.bootoptions=-Xmx512m -Xms32m
-Djava.class.path=ggjava/ggjava.jar
-Daws.accessKeyId=your_access_key
-Daws.secretKey=your_secret_key
```

This sets environmental variables using the Amazon Elastic Compute Cloud (Amazon EC2) `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` variables on the local machine.

5.7.2.2 About the AWS S3 Buckets

AWS divides S3 storage into separate file systems called **buckets**. The S3 Event Handler can write to pre-created buckets. Alternatively, if the S3 bucket does not exist, the S3 Event Handler attempts to create the specified S3 bucket. AWS requires that S3 bucket names are lowercase. Amazon S3 bucket names must be globally unique. If you attempt to create an S3 bucket that already exists in any Amazon account, it causes the S3 Event Handler to abend.

5.7.2.3 Using Templated Strings

Templated strings can contain a combination of string constants and keywords that are dynamically resolved at runtime. The S3 Event Handler makes extensive use of templated strings to generate the S3 directory names, data file names, and S3 bucket names. This gives you the flexibility to select where to write data files and the names of those data files.

Supported Templated Strings

Keyword	Description
`${fullyQualifiedTableName}`	The fully qualified source table name delimited by a period (`.`). For example, `MYCATALOG.MYSCHEMA.MYTABLE`.
`${catalogName}`	The individual source catalog name. For example, `MYCATALOG`.
`${schemaName}`	The individual source schema name. For example, `MYSCHEMA`.
`${tableName}`	The individual source table name. For example, `MYTABLE`.

Keyword	Description
${groupName}	The name of the Replicat process (with the thread number appended if you're using coordinated apply).
${emptyString}	Evaluates to an empty string. For example,""
${operationCount}	The total count of operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "1024".
${insertCount}	The total count of insert operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "125".
${updateCount}	The total count of update operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "265".
${deleteCount}	The total count of delete operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "11".
${truncateCount}	The total count of truncate operations in the data file. It must be used either on rename or by the event handlers or it will be zero (0) because nothing is written yet. For example, "5".
${currentTimestamp}	The current timestamp. The default output format for the date time is yyyy-MM-dd_HH-mm-ss.SSS. For example, 2017-07-05_04-31-23.123. Alternatively, you can customize the format of the current timestamp by inserting the format inside square brackets like: ${currentTimestamp[MM-dd_HH]} This format uses the syntax defined in the Java SimpleDateFormat class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
${toUpperCase[]}	Converts the contents inside the square brackets to uppercase. For example, ${toUpperCase[${fullyQualifiedTableName}]}.
${toLowerCase[]}	Converts the contents inside the square brackets to lowercase. For example, ${toLowerCase[${fullyQualifiedTableName}]}.

Configuration of template strings can use a mix of keywords and static strings to assemble path and data file names at runtime.

Path Configuration Example

`/usr/local/${fullyQualifiedTableName}`

Data File Configuration Example

`${fullyQualifiedTableName}_${currentTimestamp}_${groupName}.handlerIndicator`

5.7.2.4 Troubleshooting

Connectivity Issues

If the S3 Event Handler is unable to connect to the S3 object storage when running on premise, it's likely your connectivity to the public internet is protected by a proxy server. Proxy servers act a gateway between the private network of a company and the public internet. Contact your network administrator to get the URLs of your proxy server, and then setup up a proxy server.

Oracle GoldenGate can be used with a proxy server using the following parameters to enable the proxy server:

- `gg.handler.name.proxyServer=`
- `gg.handler.name.proxyPort=80`

Access to the proxy servers can be secured using credentials and the following configuration parameters:

- `gg.handler.name.proxyUsername=username`
- `gg.handler.name.proxyPassword=password`

Sample configuration:

```
gg.handlerlist=s3
gg.handler.s3.type=s3
gg.handler.s3.mode=op
gg.handler.s3.format=json
gg.handler.s3.region=us-west-2
gg.handler.s3.partitionMappingTemplate=TestPartitionName
gg.handler.s3.streamMappingTemplate=TestStream
gg.handler.s3.deferFlushAtTxCommit=true
gg.handler.s3.deferFlushOpCount=1000
gg.handler.s3.formatPerOp=true
#gg.handler.s3.customMessageGrouper=oracle.goldengate.handler.s3.s3JsonTxMessageGroup
er
gg.handler.s3.proxyServer=www-proxy.myhost.com
gg.handler.s3.proxyPort=80
```

5.7.3 Configuring the S3 Event Handler

You configure the S3 Event Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the S3 Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=s3` and the other S3 Event properties as follows:

Table 5-7 S3 Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.type`	Required	`s3`	None	Selects the S3 Event Handler for use with Replicat.
`gg.eventhandler.name.region`	Required	The AWS region name that is hosting your S3 instance.	None	Setting the legal AWS region name is required.
`gg.eventhandler.name.proxyServer`	Optional	The host name of your proxy server.	None	Sets the host name of your proxy server if connectivity to AWS is required use a proxy server.
`gg.eventhandler.name.proxyPort`	Optional	The port number of the proxy server.	None	Sets the port number of the proxy server if connectivity to AWS is required use a proxy server.
`gg.eventhandler.name.proxyUsername`	Optional	The username of the proxy server.	None	Sets the user name of the proxy server if connectivity to AWS is required use a proxy server and the proxy server requires credentials.
`gg.eventhandler.name.proxyPassword`	Optional	The password of the proxy server.	None	Sets the password for the user name of the proxy server if connectivity to AWS is required use a proxy server and the proxy server requires credentials.
`gg.eventhandler.name.bucketMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in the S3 bucket to write the file.	None	Use resolvable keywords and constants used to dynamically generate the S3 bucket name at runtime. The handler attempts to create the S3 bucket if it does not exist. AWS requires bucket names to be all lowercase. A bucket name with uppercase characters results in a runtime exception.

Table 5-7 (Cont.) S3 Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.eventhandler.name.pathMappingTemplate`	Required	A string with resolvable keywords and constants used to dynamically generate the path in the S3 bucket to write the file.	None	Use keywords interlaced with constants to dynamically generate a unique S3 path names at runtime. Typically, path names follow the format, `ogg/data/${groupName}/${fullyQualifiedTableName}` In S3, the convention is *not* to begin the path with the backslash (`/`) because it results in a root directory of `""`.
`gg.eventhandler.name.fileMappingTemplate`	Optional	A string with resolvable keywords and constants used to dynamically generate the S3 file name at runtime.	None	Use resolvable keywords and constants used to dynamically generate the S3 data file name at runtime. If not set, the upstream file name is used.
`gg.eventhandler.name.finalizeAction`	Optional	`none | delete`	`none`	Set to `none` to leave the S3 data file in place on the finalize action. Set to `delete` if you want to delete the S3 data file with the finalize action.
`gg.eventhandler.name.eventHandler`	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	The event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.
`gg.eventhandler.name.url`	Optional(unless Dell ECS, then required)	A legal URL to connect to cloud storage.	None	Not required for Amazon AWS S3. Required for Dell ECS. Sets the URL to connect to cloud storage.

6
Using the Flume Handler

Learn how to use the Flume Handler to stream change capture data to a Flume source database.

Topics:

- Overview (page 6-1)
- Setting Up and Running the Flume Handler (page 6-1)
- Data Mapping of Operations to Flume Events (page 6-3)
- Performance Considerations (page 6-5)
- Metadata Change Events (page 6-5)
- Example Flume Source Configuration (page 6-5)
- Advanced Features (page 6-6)
- Troubleshooting the Flume Handler (page 6-8)

6.1 Overview

The Flume Handler is designed to stream change capture data from an 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, sources, channels, and sinks that 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.

6.2 Setting Up and Running the Flume Handler

Instructions for configuring the Flume Handler components and running the handler are described in this section.

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 with a network connection. For instructions on how to configure and start a Flume Agent process, see the *Flume User Guide* `https://flume.apache.org/releases/content/1.6.0/FlumeUserGuide.pdf`.

Topics:

- Classpath Configuration (page 6-2)
- Flume Handler Configuration (page 6-2)
- Review a Sample Configuration (page 6-3)

6.2.1 Classpath Configuration

For the Flume Handler to connect to the Flume source and run, the Flume Agent configuration file and the Flume client jars must be configured in `gg.classpathconfiguration` variable. 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 list of the required Flume client JAR files by version, see Flume Handler Client Dependencies (page D-1).

The Oracle GoldenGate property, `gg.classpath` variable must 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 here. The path to the Flume Agent configuration file must contain the path with no wild card appended. The inclusion of the wildcard in the path to the Flume Agent configuration file will make the file inaccessible. Conversely, pathing to the dependency jars must include the `*` wildcard 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 a correctly configured `gg.classpath` variable:

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

If the Flume Handler and Flume are not collocated, then the Flume Agent configuration file and the Flume client libraries must be copied to the machine hosting the Flume Handler process.

6.2.2 Flume Handler Configuration

The following are the configurable values for the Flume Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Flume Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=flume` and the other Flume properties as follows:

Property Name	Property Value	Required / Optional	Default	Description
`gg.handlerlist`	`flumehandler` (choice of any name)	Yes		List of handlers. Only one is allowed with grouping properties `ON`.
`gg.handler.name.type`	`flume`	Yes		Type of handler to use.

Property Name	Property Value	Required / Optional	Default	Description
gg.handler.name.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 • json_row You can also write a custom formatter and include the fully qualified class name here.
gg.handler.name.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 must exist in the classpath.
gg.handler.name.mode	op|tx	No. Defaults to op		Operation mode (op) or Transaction Mode (tx). Java Adapter grouping options can be used only in tx mode.
gg.handler.name.EventHeaderClass	A custom implementation fully qualified class name	No. Defaults to DefaultFlumeEventHeader		Class to be used to define what header properties are to be added to a flume event.
gg.handler.name.EventMapsTo	op|tx	No. Defaults to op		Defines whether each flume event represents an operation or a transaction. If handler mode = op, EventMapsTo will always be op.
gg.handler.name.PropagateSchema	true|false	No. Defaults to false		When set to true, the Flume handler publishes schema events.
gg.handler.name.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.

6.2.3 Review a 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
```

6.3 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).

Topics:

- Operation Mode (page 6-4)
- Transaction Mode and EventMapsTo Operation (page 6-4)
- Transaction Mode and EventMapsTo Transaction (page 6-4)

6.3.1 Operation Mode

The configuration for the Flume Handler in the Oracle GoldenGate Java configuration file is as follows:

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

The data for each operation from an Oracle GoldenGate trail file maps into a single Flume Event. Each event is immediately flushed to Flume. Each Flume Event has 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)

6.3.2 Transaction Mode and `EventMapsTo` Operation

The configuration for the Flume Handler in the Oracle GoldenGate Java configuration file is as follows:

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

The data for each operation from Oracle GoldenGate trail file maps into a single Flume Event. Events are flushed to Flume when the transaction is committed. Each Flume Event has 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)

We recommend that you use this mode when formatting data as Avro or delimited text. It is important to understand that configuring Replicat batching functionality increases the number of operations that are processed in a transaction.

6.3.3 Transaction Mode and `EventMapsTo` Transaction

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

```
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 when the transaction is committed. 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

We recommend that you use this mode only when using self describing formats such as JSON or XML. In is important to understand that configuring Replicat batching functionality increases the number of operations that are processed in a transaction.

6.4 Performance Considerations

Consider the following options for enhanced performance:

- Set Replicat-based grouping
- Set the transaction mode with `gg.handler.flume_handler_name. EventMapsTo=tx`
- Increase the maximum heap size of the JVM in Oracle GoldenGate Java properties file (the maximum heap size of the Flume Handler may affect performance)

6.5 Metadata Change Events

The Flume Handler is adaptive to metadata change events. To handle metadata change events, the source trail files must have metadata in the trail file. 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 available for all database implementations in Oracle GoldenGate. To determine whether DDL replication is supported, see the Oracle GoldenGate installation and configuration guide for the appropriate database.

Whenever a metadata change occurs at the source, the Flume Handler notifies 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 that the associated formatter encounters an operation for that table the schema is regenerated.

6.6 Example Flume Source Configuration

Topics:

- Avro Flume Source (page 6-5)
- Thrift Flume Source (page 6-6)

6.6.1 Avro Flume Source

The following is a 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
```

6.6.2 Thrift Flume Source

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

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

6.7 Advanced Features

You may choose to implement the following advanced features of the Flume Handler:

Topics:

- Schema Propagation (page 6-6)
- Security (page 6-7)
- Fail Over Functionality (page 6-7)
- Load Balancing Functionality (page 6-7)

6.7.1 Schema Propagation

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

```
gg.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 a new schema is generated the next time an operation is encountered for that table.

When schema propagation is enabled, the Flume Handler propagates schemas in an Avro Event when they are encountered.

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

- `SCHEMA_EVENT`: true
- `GENERIC_WRAPPER`: true or 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

6.7.2 Security

Kerberos authentication for the Oracle GoldenGate for Big Data Flume Handler connection to the Flume source is possible. This feature is supported only in Flume 1.6.0 and later using the Thrift Flume source. You can enable it by changing the configuration of the Flume source in the Flume Agent configuration file.

The following is an example of the Flume source configuration from the Flume Agent configuration file that shows how to enable Kerberos authentication. You must provide Kerberos principal name of the client and the server. 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 administer Kerberos, on Kerberos principals and their associated passwords, and about the creation of a Kerberos `keytab` file, see 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
```

6.7.3 Fail Over Functionality

It is possible to configure the Flume Handler so that it fails over when the primary Flume source becomes unavailable. This feature is currently supported only in Flume 1.6.0 and later using the Avro Flume source. It is enabled with Flume source configuration in the Flume Agent configuration file. The following is a 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
```

6.7.4 Load Balancing Functionality

You can configure the Flume Handler so that produced Flume events are load-balanced across multiple Flume sources. This feature is currently supported only in Flume 1.6.0 and later using the Avro Flume source. You can enable it by changing the Flume source configuration in the Flume Agent configuration file. The following is a 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
```

6.8 Troubleshooting the Flume Handler

Topics:

- Java Classpath (page 6-8)
- Flume Flow Control Issues (page 6-8)
- Flume Agent Configuration File Not Found (page 6-8)
- Flume Connection Exception (page 6-9)
- Other Failures (page 6-9)

6.8.1 Java Classpath

Issues with the Java classpath are common. A `ClassNotFoundException` in the Oracle GoldenGate Java `log4j` log file indicates a classpath problem. You can use the Java `log4j` log file 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. This way, you can make sure that all of the required dependency JARs are resolved. For more information, see Classpath Configuration (page 6-2).

6.8.2 Flume Flow Control Issues

In some situations, the Flume Handler may write to the Flume source faster than the Flume sink can dispatch messages. When this happens, the Flume Handler works for a while, but when Flume can no longer accept messages, it will abend. The cause that is logged in the Oracle GoldenGate Java log file may probably be an `EventDeliveryException`, indicating that the Flume Handler was unable to send an event. Check the Flume log for the exact cause of the problem. You may be able to re-configure the Flume channel to increase capacity or increase the Java heap size if the Flume Agent is experiencing an `OutOfMemoryException`. This may not solve the problem. If the Flume Handler can push data to the Flume source faster than messages are dispatched by the Flume sink, then any change may only extend the period the Flume Handler can run before failing.

6.8.3 Flume Agent Configuration File Not Found

If the Flume Agent configuration file is not in the classpath,, Flume Handler abends at startup. The result is usually a `ConfigException` that reports the issue as an error loading the Flume producer properties. Check the `gg.handler.`*`name`*. `RpcClientProperites` configuration file to make sure that the naming of the Flume Agent properties file is correct. Check the Oracle GoldenGate `gg.classpath` properties to make sure 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.

6.8.4 Flume Connection Exception

The Flume Handler terminates abnormally at start up if it cannot connect to the Flume source. The root cause of this problem may probably be reported as an `IOExeption` in the Oracle GoldenGate Java `log4j`, file indicating a problem connecting to Flume at a given host and port. Make sure that the following are both true:

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

6.8.5 Other Failures

Review the contents of the Oracle GoldenGate Java `log4j` file to identify any other issues.

7

Using the HBase Handler

Learn how to use the HBase Handler to populate HBase tables from existing Oracle GoldenGate supported sources.

Topics:

- Overview (page 7-1)
- Detailed Functionality (page 7-1)
- Setting Up and Running the HBase Handler (page 7-2)
- Security (page 7-6)
- Metadata Change Events (page 7-6)
- Additional Considerations (page 7-7)
- Troubleshooting the HBase Handler (page 7-7)

7.1 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 replicated through Oracle GoldenGate is highly structured. HBase provides a way to maintain the important structure of data while taking advantage of the horizontal scaling that is offered by the Hadoop Distributed File System (HDFS).

7.2 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. Table name 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 to 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 way to group 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`. You can configure the column family name. However, after a table is created with a specific column family name, you cannot reconfigure the column family name in the HBase example, without first modifying or dropping the table results in an abend of the Oracle GoldenGateReplicat processes.

7.3 Setting Up and Running the HBase Handler

HBase must run either collocated with the HBase Handler process or on a machine that can connect from the network that is hosting the HBase Handler process. The underlying HDFS single instance or clustered instance serving as the repository for HBase data must also run.

Instructions for configuring the HBase Handler components and running the handler are described in this section.

Topics:

- Classpath Configuration (page 7-2)
- HBase Handler Configuration (page 7-3)
- Sample Configuration (page 7-5)
- Performance Considerations (page 7-6)

7.3.1 Classpath Configuration

For the HBase Handler to connect to HBase and stream data, the hbase-site.xml file and the HBase client jars must be configured in `gg.classpath` variable. 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 (page E-1) lists the 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. The path to the `hbase-site.xml` file must contain only the path with no wild card appended. The inclusion of the * wildcard in the path to the `hbase-site.xml` file will cause it to be inaccessible. Conversely, the path to the dependency jars must include the (*) wildcard character in

order to include all the jar files in that directory, in the associated classpath. Do not use `*.jar`. The following is an example of a correctly configured `gg.classpath` variable:

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

7.3.2 HBase Handler Configuration

The following are the configurable values for the HBase Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the HBase Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=hbase` and the other HBase properties as follows:

Table 7-1 HBase Handler Configuration Properties

Properties	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`	Required	`hbase`.	None	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`	Using `true` indicates that token values are included in the output to HBase. Using `false` means token values are not to 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 7-1 (Cont.) HBase Handler Configuration Properties

Properties	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, see 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 end abnormally. • `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.
`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.

Table 7-1 (Cont.) HBase Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`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.
`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.
`gg.handler.name.setHBaseOperationTimestamp`	Optional	`true` | `false`	`false`	Set to `true` to set the timestamp for HBase operations in the HBase Handler instead of allowing HBase is assign the timestamps on the server side. This property can be used to solve the problem of a row delete followed by an immediate reinsert of the row not showing up in HBase, see HBase Handler Delete-Insert Problem (page 7-8).
`gg.handler.name.omitNullValues`	Optional	`true` | `false`	`false`	Set to `true` to omit null fields from being written.

[1] See *Java Internalization Support* at `https://docs.oracle.com/javase/8/docs/technotes/guides/intl/`.

7.3.3 Sample Configuration

The following is a 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
```

7.3.4 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 using the Replicat `GROUPTRANSOPS` parameter to group multiple smaller transactions in the source trail file into a larger single transaction applied to HBase. You can use Replicat base-batching by adding the 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 is committed.

7.4 Security

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

You have to include the directory containing the HDFS `core-site.xml` file in the classpath. Kerberos authentication is performed using the Hadoop `UserGroupInformation` class. This class relies on the Hadoop configuration property `hadoop.security.authentication` being set to `kerberos` to successfully perform the `kinit` command.

Additionally, you must set the following properties in the 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}.
```

You may encounter the inability to decrypt the Kerberos password from the `keytab` file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html.

7.5 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 is that the new column and its associated data are streamed to HBase after the metadata change event.

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 later. However, these trail files do not include metadata in trail, and therefore metadata change events cannot be handled at runtime.

7.6 Additional Considerations

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.

Classpath issues are common during the initial setup of the HBase Handler. The typical indicators are occurrences of the `ClassNotFoundException` in the Java `log4j` log file. The HBase client jars do not ship with Oracle GoldenGate for Big Data. You must resolve the required HBase client jars. HBase Handler Client Dependencies (page E-1) includes a list 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 Classpath Configuration (page 7-2).

7.7 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.

Topics:

- Java Classpath (page 7-7)
- HBase Connection Properties (page 7-8)
- Logging of Handler Configuration (page 7-8)
- HBase Handler Delete-Insert Problem (page 7-8)
- Cloudera CDH HBase Compatibility (page 7-9)

7.7.1 Java Classpath

Issues with the Java classpath are common. A `ClassNotFoundException` in the Java `log4j` log file indicates a classpath problem. You can use the Java `log4j` log file to troubleshoot this issue. Setting the log level to `DEBUG` logs each of the jars referenced in the `gg.classpath` object to the log file. You can make sure that all of the required dependency jars are resolved by enabling `DEBUG` level logging, and then searching 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"
```

7.7.2 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 file shows 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].
```

Commonly, for the `hbase-site.xml` file is not included in the classpath or the path to the `hbase-site.xml` file is incorrect. In this case, the HBase Handler cannot establish a connection to HBase, and the Oracle GoldenGate process abends. The following error is 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.

7.7.3 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. The following is a sample output:

```
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]
```

7.7.4 HBase Handler Delete-Insert Problem

If you are using the HBase Handler `gg.handler.`*`name`*`.setHBaseOperationTimestamp` configuration property, the source database may get out of sync with data in the HBase Handler tables. This is caused by the deletion of a row followed by the immediate reinsertion of the row. HBase creates a tombstone marker for the delete that is identified by a specific timestamp. This tombstone marker marks any row records in HBase with the same row key as deleted that have a timestamp before or the same as the tombstone marker. This can occur when the deleted row is immediately reinserted. The insert operation can inadvertently have the same timestamp as the delete operation so the delete operation causes the subsequent insert operation to incorrectly appear as deleted.

To work around this issue, you need to set the `gg.handler.name.setHbaseOperationTimestamp=` to `true`, which does two things:

- Sets the timestamp for row operations in the HBase Handler.
- Detection of a delete-insert operation that ensures that the insert operation has a timestamp that is after the insert.

The default for `gg.handler.name.setHbaseOperationTimestamp` is`false`, which means that the HBase server supplies the timestamp for a row. This can cause the out of sync problem.

Setting the row operation timestamp in the HBase Handler can have these consequences:

1. Since the timestamp is set on the client side, this could create problems if multiple applications are feeding data to the same HBase table.
2. If delete and reinsert is a common pattern in your use case, then the HBase Handler has to increment the timestamp 1 millisecond each time this scenario is encountered.

Processing cannot be allowed to get too far into the future so the HBase Handler only allows the timestamp to increment 100 milliseconds into the future before it attempts to wait the process so that the client side HBase operation timestamp and real time are back in sync. When a delete-insert is used instead of an update in the source database so this sync scenario would be quite common. Processing speeds may be affected by not allowing the HBase timestamp to go over 100 milliseconds into the future if this scenario is common.

7.7.5 Cloudera CDH HBase Compatibility

The Cloudera CDH has moved to HBase 1.0.0 in the CDH 5.4.0 version. To keep reverse compatibility with HBase 0.98.*x* and before, the HBase client in the CDH broke the binary compatibility with Apache HBase 1.0.0. This created a compatibility problem for the HBase Handler when connecting to Cloudera CDH HBase for CDH versions 5.4 - 5.11. You may have been advised to solve this problem by using the old 0.98 HBase interface and setting the following configuration parameter:

```
gg.handler.name.hBase98Compatible=true
```

This compatibility problem is solved using Java Refection. If you are using the HBase Handler to connect to CDH 5.4*x*, then you should changed the HBase Handler configuration property to the following:

```
gg.handler.name.hBase98Compatible=false
```

Optionally, you can omit the property entirely because the default value is `false`.

8
Using the HDFS Handler

Learn how to use the HDFS Handler, which is designed to stream change capture data into the Hadoop Distributed File System (HDFS).

Topics:

- Overview (page 8-1)
- Writing into HDFS in SequenceFile Format (page 8-1)
- Setting Up and Running the HDFS Handler (page 8-2)
- Writing in HDFS in Avro Object Container File Format (page 8-11)
- Generating HDFS File Names Using Template Strings (page 8-11)
- Metadata Change Events (page 8-12)
- Partitioning (page 8-13)
- HDFS Additional Considerations (page 8-13)
- Best Practices (page 8-14)
- Troubleshooting the HDFS Handler (page 8-15)

8.1 Overview

The HDFS is the primary file system for Big Data. Hadoop is typically installed on multiple machines that work together as a Hadoop cluster. Hadoop allows you 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.

8.2 Writing into HDFS in SequenceFile 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.

Topics:

- Integrating with Hive (page 8-1)
- Understanding the Data Format (page 8-2)

8.2.1 Integrating with Hive

Oracle GoldenGate for Big Data release does not include a Hive storage handler because the HDFS Handler provides all of the necessary Hive functionality .

You can create a Hive integration to create tables and update table definitions in case of DDL events. This is limited to data formatted in Avro Object Container File format. For more information, see Writing in HDFS in Avro Object Container File Format (page 8-11) and HDFS Handler Configuration (page 8-3).

For Hive to consume sequence files, the DDL creates Hive tables including `STORED as sequencefile` . The 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`.

8.2.2 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 Using the Delimited Text Formatter (page 17-25) 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
```

8.3 Setting Up and Running the HDFS Handler

To 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 you can download it from:

http://hadoop.apache.org/

Follow the Getting Started links for information on how to install a single-node cluster (for pseudo-distributed operation mode) or a clustered setup (for fully-distributed operation mode).

Instructions for configuring the HDFS Handler components and running the handler are described in the following sections.

- Classpath Configuration (page 8-3)
- HDFS Handler Configuration (page 8-3)
- Review a Sample Configuration (page 8-9)

- Performance Considerations (page 8-10)
- Security (page 8-10)

8.3.1 Classpath Configuration

For the HDFS Handler to connect to HDFS and run, the HDFS `core-site.xml` file and the HDFS client jars must be configured in `gg.classpath` variable. The HDFS client jars must match the version of HDFS that the HDFS Handler is connecting. For a list of the required client jar files by release, see HDFS Handler Client Dependencies (page F-1).

The default location of the `core-site.xml` file is *`Hadoop_Home`*`/etc/hadoop`

The default locations 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. The path to the `core-site.xml` file must contain the path to the directory containing the `core-site.xml`file with no wildcard appended. If you include a (*) wildcard in the path to the `core-site.xml` file, the file is not picked up. Conversely, the path to the dependency jars must include the (*) wildcard character in order to include all the jar files in that directory in the associated classpath. Do not use `*.jar`.

The following is an example of a correctly configured `gg.classpath` variable:

```
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` must also be in the classpath if Kerberos security is enabled. By default, the `hdfs-site.xml` file is located in the *`Hadoop_Home`*`/etc/hadoop` directory. If the HDFS Handler is not collocated with Hadoop, either or both files can be copied to another machine.

8.3.2 HDFS Handler Configuration

The following are the configurable values for the HDFSHandler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the HDFS Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=hdfs` and the other HDFS properties as follows:

Table 8-1 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`	Required	`hdfs`	None	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	The default unit of measure is bytes. You can use `k`, `m`, or `g` to specify kilobytes, megabytes, or gigabytes. Examples of legal values include `10000`, `10k`, `100m`, `1.1g`.	`1g`	Selects the maximum file size of the created HDFS files.
`gg.handler.`*`name`*`.pathMappingTemplate`	Optional	Any legal templated string to resolve the target write directory in HDFS. Templates can contain a mix of constants and keywords which are dynamically resolved at runtime to generate the HDFS write directory.	`/ogg/${toLowerCase[${`*`fullyQualifiedTableName`*`}]}`	You can use keywords interlaced with constants to dynamically generate the HDFS write directory at runtime, see Generating HDFS File Names Using Template Strings (page 8-11).
`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 is closed. A new file is not immediately opened. New HDFS files are created on a just-in-time basis.

Table 8-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.name.inactivityRollInterval`	Optional	The default unit of measure is milliseconds. You can use `ms`, `s`, `m`, `h` to specify milliseconds, seconds, minutes, or hours. 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 is closed. A new file is not immediately opened. New HDFS files are created on a just-in-time basis.
`gg.handler.name.fileNameMappingTemplate`	Optional	A string with resolvable keywords and constants used to dynamically generate HDFS file names at runtime.	`${fullyQualifiedTableName}_${groupName}_${currentTimeStamp}.txt`	You can use keywords interlaced with constants to dynamically generate unique HDFS file names at runtime, see Generating HDFS File Names Using Template Strings (page 8-11). File names typically follow the format, `${fullyQualifiedTableName}_${groupName}_${currentTimeStamp}{.txt}`.
`gg.handler.name.partitionByTable`	Optional	`true` | `false`	`true` (data is partitioned by table)	Determines whether data written into HDFS must be partitioned by table. If set to `true`, then data for different tables are written to different HDFS files. If set 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. If set to `false`, a configuration exception occurs at initialization.
`gg.handler.name.rollOnMetadataChange`	Optional	`true` | `false`	`true` (HDFS files are rolled on a metadata change event)	Determines whether HDFS files are 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. If set to `false`, a configuration exception occurs at initialization.

Table 8-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.name.format`	Optional	`delimitedtext` | `json` | `json_row` | `xml` | `avro_row` | `avro_op` | `avro_row_ocf` | `avro_op_ocf` | `sequencefile`	`delimitedtext`	Selects the formatter for the HDFS Handler for how output data is formatted. • `delimitedtext`: Delimited text • `json`: JSON • `json_row`: JSON output modeling row data • `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 (OCF) 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. Set to `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}`
`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.

Table 8-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`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 is 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`
`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. Snappy includes native binaries in the Snappy JAR file and performs a Java-native traversal when compressing or decompressing. Use of Snappy may introduce runtime issues and platform porting issues that you may not experience when working with Java. You may need to perform additional testing to ensure that 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 8-1 (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 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 following Java call: `System.getProperty(user.name)`
`gg.handler.name.hiveJdbcPassword`	Optional	A legal password if the Hive JDBC connection requires a password.	None	Only applicable to the Avro 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. This property is required if Hive is configured to require passwords for the JDBC connection.
`gg.handler.name.hiveJdbcDriver`	Optional	The fully qualified Hive JDBC driver class name.	`org.apache.hive.jdbc.HiveDriver`	Only applicable to the Avro 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 when Apache Hive introduces a new JDBC driver class.

Table 8-1 (Cont.) HDFS Handler Configuration Properties

Property	Optional / Required	Legal Values	Default	Explanation
`gg.handler.name.openNextFileAtRoll`	Optional	`true | false`	`false`	Applicable only to the HDFS Handler that is not writing an Avro OCF or sequence file to support extract, load, transform (ELT) situations. When set to `true`, this property creates a new file immediately on the occurrence of a file roll. File rolls can be triggered by any one of the following: • Metadata change • File roll interval elapsed • Inactivity interval elapsed Data files are being loaded into HDFS and a monitor program is monitoring the write directories waiting to consume the data. The monitoring programs use the appearance of a new file as a trigger so that the previous file can be consumed by the consuming application.
`gg.handler.name.hsync`	Optional	`true | false`	`false`	Set to use an `hflush` call to ensure that data is transferred from the HDFS Handler to the HDFS cluster. When set to `false`, `hflush` is called on open HDFS write streams at transaction commit to ensure write durability. Setting `hsync` to `true` calls `hsync` instead of `hflush` at transaction commit. Using `hsync` ensures that data has moved to the HDFS cluster and that the data is written to disk. This provides a higher level of write durability though it adversely effects performance. Also, it does not make the write data immediately available to analytic tools. For most applications setting this property to `false` is appropriate.

8.3.3 Review a Sample Configuration

The following is a 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.pathMappingTemplate=/ogg/${fullyQualifiedTableName
gg.handler.hdfs.fileRollInterval=0
```

```
gg.handler.hdfs.inactivityRollInterval=0
gg.handler.hdfs.partitionByTable=true
gg.handler.hdfs.rollOnMetadataChange=true
gg.handler.hdfs.authType=none
gg.handler.hdfs.format=delimitedtext
```

8.3.4 Performance Considerations

The HDFS Handler calls the HDFS flush method on the HDFS write stream to flush data to the HDFS data nodes at the end of each transaction in order to maintain write durability. This is an expensive call and performance can adversely affect, especially in the case of transactions of one or few operations that result 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 require high performance, configure batching functionality for the Replicat process. For more information, see Replicat Grouping (page 1-8).

The HDFS client libraries spawn threads for every HDFS file stream opened by the HDFS Handler. Therefore, the number of threads executing in the JMV grows proportionally to the number of HDFS file streams that are open. Performance of the HDFS Handler may 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) may result in degraded performance. If your use case requires writing to many tables, then Oracle recommends that you 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.

8.3.5 Security

The HDFS cluster can be secured using Kerberos authentication. 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 the `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.kerberosPrincipalName=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
```

You may encounter the inability to decrypt the Kerberos password from the `keytab` file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html.

8.4 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 because it:

- integrates with Apache Hive (Raw Avro written to HDFS is not supported by Hive.)
- 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 and Avro OCF integration includes 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.

8.5 Generating HDFS File Names Using Template Strings

The HDFS Handler can dynamically generate HDFS file names using a template string. The template string allows you to generate a combination of keywords that are dynamically resolved at runtime with static strings to provide you more control of generated HDFS file names. You can control the template file name using the `gg.handler.name.fileNameMappingTemplate` configuration property. The default value for this parameters is:

```
${fullyQualifiedTableName}_${groupName}_${currentTimestamp}.txt
```

Supported keywords which are dynamically replaced at runtime include the following:

Keyword
Replacement

`${fullyQualifiedTableName}`
The fully qualified table name with period (.) delimiting the names. For example, `oracle.test.table1`.

`${catalogName}`
The catalog name of the source table. For example, `oracle`.

`${schemaName}`
The schema name of the source table. For example, `test`.

`${tableName}`
The short table name of the source table. For example, `table1`.

${*groupName*}
The Replicat process name concatenated with the thread id if using coordinated apply. For example, `HDFS001`.

${*currentTimestamp*}
The default output format for the date time is `yyyy-MM-dd_HH-mm-ss.SSS`. For example, `2017-07-05_04-31-23.123`.
Alternatively, your can configure your own format mask for the date using the syntax, `${currentTimestamp[yyyy-MM-dd_HH-mm-ss.SSS]}`. Date time format masks follow the convention in the `java.text.SimpleDateFormat` Java class.

${toLowerCase[]}
Converts the argument inside of the square brackets to lower case. Keywords can be nested inside of the square brackets as follows:

```
${toLowerCase[${fullyQualifiedTableName}]}
```

This is important because source table names are normalized in Oracle GoldenGate to upper case.

${toUpperCase[]}
Converts the arguments inside of the square brackets to upper case. Keywords can be nested inside of the square brackets.

Following are examples of legal templates and the resolved strings:

Legal Template	Replacement
${*schemaName*}.${*tableName*}__${*groupName*}_${currentTimestamp}.txt	`test.table1__HDFS001_2017-07-05_04-31-23.123.txt`
${*fullyQualifiedTableName*}--${currentTimestamp}.avro	`oracle.test.table1–2017-07-05_04-31-23.123.avro`
${*fullyQualifiedTableName*}_${currentTimestamp[yyyy-MM-ddTHH-mm-ss.SSS]}.json	`oracle.test.table1–2017-07-05T04-31-23.123.json`

Be aware of these restrictions when generating HDFS file names using templates:

- Generated HDFS file names must be legal HDFS file names.
- Oracle strongly recommends that you use `${groupName}` as part of the HDFS file naming template when using coordinated apply and breaking down source table data to different Replicat threads. The group name provides uniqueness of generated HDFS names that `${currentTimestamp}` alone does not guarantee.. HDFS file name collisions result in an abend of the Replicat process.

8.6 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 GoldenGatedoes not support DDL replication for all database implementations. See the Oracle GoldenGateinstallation and configuration guide for the appropriate database to determine whether DDL replication is supported.

8.7 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
```

You should exercise care when choosing columns for partitioning. The key is to choose columns that contain only a few (10 or less) possible values, and to make sure that those values are also helpful for grouping and analyzing the data. For example, a column of sales regions would be good for partitioning. A column that contains the customers dates of birth would not be good for partitioning. Configuring partitioning on a column that has many possible values can cause problems. A poor choice can result in hundreds of HDFS file streams being opened, and performance may degrade for the reasons discussed in Performance Considerations (page 8-10). Additionally, poor partitioning can result in problems during data analysis. Apache Hive requires that all `where` clauses specify partition criteria if the Hive data is partitioned.

8.8 HDFS Additional Considerations

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 list of required client JAR files by version, see HDFS Handler Client Dependencies (page F-1). The HDFS client jars do not ship with the Oracle GoldenGate for Big Dataproduct. The HDFS Handler supports multiple versions of HDFS, and the HDFS client jars must be the same version as the HDFS version to which the HDFS Handler is connecting. The HDFS client jars are open source and are 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 must 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 be advantageous for troubleshooting, building a point of contact (POC), or as a step in the process of building an HDFS integration.

Another common issue is that data streamed to HDFS using the HDFS Handler may not be 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 128 MB 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 may occur. This is an anomaly of HDFS streaming and is discussed in the HDFS specification. This anomaly of HDFS leads to a potential 128 MB 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 because closing the HDFS write stream finalizes the block writing. Data is immediately visible to analytic tools, and file sizing metrics become consistent again. Therefore, the new file rolling feature in the HDFS Handler can be used to close HDFS writes streams, making all data visible.

> **Important:**
>
> The file rolling solution may present its own problems. Extensive use of file rolling can result in many small files in HDFS. Many small files in HDFS may result 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 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 ultimately recognizes that the write stream has been broken, so HDFS finalizes the write block. In this scenario, you may experience a short term delay before the HDFS process finalizes the write block.

8.9 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. Because the HDFS cluster nodes and the edge nodes are different servers, the following benefits are seen:

- The HDFS cluster nodes do not compete for resources with the applications interfacing with the cluster.
- The requirements for the HDFS cluster nodes and edge nodes probably differ. This physical topology allows the appropriate hardware to be tailored to specific needs.

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 are 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.

8.10 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 configure the runtime to correctly generate the Java `log4j` log file.

Topics:

- Java Classpath (page 8-15)
- HDFS Connection Properties (page 8-15)
- Handler and Formatter Configuration (page 8-15)

8.10.1 Java Classpath

Problems with the Java classpath are common. 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 by enabling `DEBUG` level logging and search the log file for messages, as in 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
```

8.10.2 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 output shows 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 points to 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].
```

8.10.3 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. The output resembles the following:

```
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 ****
```

9

Using the Java Database Connectivity Handler

Learn how to use the Java Database Connectivity (JDBC) Handler, which can replicate source transactional data to a target or database.

Topics:

- Overview (page 9-1)
- Detailed Functionality (page 9-1)
- Setting Up and Running the JDBC Handler (page 9-3)
- Sample Configurations (page 9-7)

9.1 Overview

The Generic Java Database Connectivity (JDBC) Handler lets you replicate source transactional data to a target system or database by using a JDBC interface. You can use it with targets that support JDBC connectivity.

You can use the JDBC API to access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which the JDBC Handler was built. The JDBC handler with the JDBC metadata provider also lets you use Replicat features such as column mapping and column functions. For more information about using these features, see Using the Metadata Providers (page 15-1)

For more information about using the JDBC API, see http://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/index.html.

9.2 Detailed Functionality

The JDBC Handler replicates source transactional data to a target or database by using a JDBC interface.

Topics:

- Single Operation Mode (page 9-2)
- Oracle Database Data Types (page 9-2)
- MySQL Database Data Types (page 9-2)
- Netezza Database Data Types (page 9-3)
- Redshift Database Data Types (page 9-3)

9.2.1 Single Operation Mode

The JDBC Handler performs SQL operations on every single trail record (row operation) when the trail record is processed by the handler. The JDBC Handler does not use the `BATCHSQL` feature of the JDBC API to batch operations.

9.2.2 Oracle Database Data Types

The following column data types are supported for Oracle Database targets:

```
NUMBER
DECIMAL
INTEGER
FLOAT
REAL
DATE
TIMESTAMP
INTERVAL YEAR TO MONTH
INTERVAL DAY TO SECOND
CHAR
VARCHAR2
NCHAR
NVARCHAR2
RAW
CLOB
NCLOB
BLOB
TIMESTAMP WITH TIMEZONE[1]
TIME WITH TIMEZONE[2]
```

9.2.3 MySQL Database Data Types

The following column data types are supported for MySQL Database targets:

```
INT
REAL
FLOAT
DOUBLE
NUMERIC
DATE
DATETIME
TIMESTAMP
TINYINT
BOOLEAN
SMALLINT
BIGINT
```

[1] Time zone with a two-digit hour and a two-digit minimum offset.
[2] Time zone with a two-digit hour and a two-digit minimum offset.

```
MEDIUMINT
DECIMAL
BIT
YEAR
ENUM
CHAR
VARCHAR
```

9.2.4 Netezza Database Data Types

The following column data types are supported for Netezza database targets:

```
byteint
smallint
integer
bigint
numeric(p,s)
numeric(p)
float(p)
Real
double
char
varchar
nchar
nvarchar
date
time
Timestamp
```

9.2.5 Redshift Database Data Types

The following column data types are supported for Redshift database targets:

```
SMALLINT
INTEGER
BIGINT
DECIMAL
REAL
DOUBLE
CHAR
VARCHAR
DATE
TIMESTAMP
```

9.3 Setting Up and Running the JDBC Handler

The following sections provide instructions for configuring the JDBC Handler components and running the handler.

> **Note:**
>
> Use the JDBC Metadata Provider with the JDBC Handler to obtain column mapping features, column function features, and better data type mapping.

Topics:

- Java Classpath (page 9-4)
- Handler Configuration (page 9-4)
- Statement Caching (page 9-5)
- Setting Up Error Handling (page 9-6)

9.3.1 Java Classpath

The JDBC Java Driver location must be included in the class path of the handler using the `gg.classpath` property.

For example, the configuration for a MySQL database could be:

```
gg.classpath= /path/to/jdbc/driver/jar/mysql-connector-java-5.1.39-bin.jar
```

9.3.2 Handler Configuration

You configure the JDBC Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the JDBC Handler, you must first configure the handler type by specifying `gg.handler.name.type=jdbc` and the other JDBC properties as follows:

Table 9-1 JDBC Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.type`	Required	`jdbc`	None	Selects the JDBC Handler for streaming change data capture into name.
`gg.handler.`*`name`*`.connectionURL`	Required	A valid JDBC connection URL	None	The target specific JDBC connection URL.
`gg.handler.`*`name`*`.DriverClass`	Target database dependent.	The target specific JDBC driver class name	None	The target specific JDBC driver class name.

Table 9-1 (Cont.) JDBC Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.userName`	Target database dependent.	A valid user name	None	The user name used for the JDBC connection to the target database.
`gg.handler.name.password`	Target database dependent.	A valid password	None	The password used for the JDBC connection to the target database.
`gg.handler.name.maxActiveStatements`	Optional	Unsigned integer	Target database dependent	If this property is not specified, the JDBC Handler queries the target dependent database metadata indicating maximum number of active prepared SQL statements. Some targets do not provide this metadata so then the default value of 256 active SQL statements is used. If this property is specified, the JDBC Handler will not query the target database for such metadata and use the property value provided in the configuration. In either case, when the JDBC handler finds that the total number of active SQL statements is about to be exceeded, the oldest SQL statement is removed from the cache to add one new SQL statement.

9.3.3 Statement Caching

To speed up DML operations, JDBC driver implementations typically allow multiple statements to be cached. This configuration avoids repreparing a statement for operations that share the same profile or template.

The JDBC Handler uses statement caching to speed up the process and caches as many statements as the underlying JDBC driver supports. The cache is implemented by using an LRU cache where the key is the profile of the operation (stored internally in the memory as an instance of `StatementCacheKey` class), and the value is the `PreparedStatement` object itself.

A `StatementCacheKey` object contains the following information for the various DML profiles that are supported in the JDBC Handler:

DML operation type	`StatementCacheKey` contains a tuple of:
`INSERT`	(table name, operation type, ordered after-image column indices)
`UPDATE`	(table name, operation type, ordered after-image column indices)

DML operation type	StatementCacheKey contains a tuple of:
DELETE	(table name, operation type)
TRUNCATE	(table name, operation type)

9.3.4 Setting Up Error Handling

The JDBC Handler supports using the REPERROR and HANDLECOLLISIONS Oracle GoldenGate parameters. See *Reference for Oracle GoldenGate*.

You must configure the following properties in the handler properties file to define the mapping of different error codes for the target database.

gg.error.duplicateErrorCodes
A comma-separated list of error codes defined in the target database that indicate a duplicate key violation error. Most of the drivers of the JDBC drivers return a valid error code so, REPERROR actions can be configured based on the error code. For example:

```
gg.error.duplicateErrorCodes=1062,1088,1092,1291,1330,1331,1332,1333
```

gg.error.notFoundErrorCodes
A comma-separated list of error codes that indicate missed DELETE or UPDATE operations on the target database.
In some cases, the JDBC driver errors occur when an UPDATE or DELETE operation does not modify any rows in the target database so, no additional handling is required by the JDBC Handler.
Most JDBC drivers do not return an error when a DELETE or UPDATE is affecting zero rows so, the JDBC Handler automatically detects a missed UPDATE or DELETE operation and triggers an error to indicate a not-found error to the Replicat process. The Replicat process can then execute the specified REPERROR action.
The default error code used by the handler is zero. When you configure this property to a non-zero value, the configured error code value is used when the handler triggers a not-found error. For example:

```
gg.error.notFoundErrorCodes=1222
```

gg.error.deadlockErrorCodes
A comma-separated list of error codes that indicate a deadlock error in the target database. For example:

```
gg.error.deadlockErrorCodes=1213
```

Setting Codes
Oracle recommends that you set a non-zero error code for the gg.error.duplicateErrorCodes, gg.error.notFoundErrorCodes, and gg.error.deadlockErrorCodes properties because Replicat does not respond to REPERROR and HANDLECOLLISIONS configuration when the error code is set to zero.

Sample Oracle Database Target Error Codes

```
gg.error.duplicateErrorCodes=1
gg.error.notFoundErrorCodes=0
gg.error.deadlockErrorCodes=60
```

Sample MySQL Database Target Error Codes

```
gg.error.duplicateErrorCodes=1022,1062
gg.error.notFoundErrorCodes=1329
gg.error.deadlockErrorCodes=1213,1614
```

9.4 Sample Configurations

The following sections contain sample configurations for the databases supported by the JDBC Handler from the Java Adapter properties file.

Topics:

- Sample Oracle Database Target (page 9-7)
- Sample Oracle Database Target with JDBC Metadata Provider (page 9-7)
- Sample MySQL Database Target (page 9-8)
- Sample MySQL Database Target with JDBC Metadata Provider (page 9-8)

9.4.1 Sample Oracle Database Target

```
gg.handlerlist=jdbcwriter
gg.handler.jdbcwriter.type=jdbc

#Handler properties for Oracle database target
gg.handler.jdbcwriter.DriverClass=oracle.jdbc.driver.OracleDriver
gg.handler.jdbcwriter.connectionURL=jdbc:oracle:thin:@<DBServer address>:
1521:<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/oracle/jdbc/driver/ojdbc5.jar
goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./
dirprm
```

9.4.2 Sample Oracle Database Target with JDBC Metadata Provider

```
gg.handlerlist=jdbcwriter
gg.handler.jdbcwriter.type=jdbc

#Handler properties for Oracle database target with JDBC Metadata provider
gg.handler.jdbcwriter.DriverClass=oracle.jdbc.driver.OracleDriver
gg.handler.jdbcwriter.connectionURL=jdbc:oracle:thin:@<DBServer address>:
1521:<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/oracle/jdbc/driver/ojdbc5.jar
#JDBC Metadata provider for Oracle target
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:oracle:thin:@<DBServer address>:1521:<database name>
gg.mdp.DriverClassName=oracle.jdbc.driver.OracleDriver
```

```
gg.mdp.UserName=<dbuser>
gg.mdp.Password=<dbpassword>
goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./
dirprm
```

9.4.3 Sample MySQL Database Target

```
gg.handlerlist=jdbcwriter
gg.handler.jdbcwriter.type=jdbc

#Handler properties for MySQL database target
gg.handler.jdbcwriter.DriverClass=com.mysql.jdbc.Driver
gg.handler.jdbcwriter.connectionURL=jdbc:<a target="_blank"
href="mysql://">mysql://</a><DBServer address>:3306/<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/mysql/jdbc/driver//mysql-connector-java-5.1.39-bin.jar

goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./
dirprm
```

9.4.4 Sample MySQL Database Target with JDBC Metadata Provider

```
gg.handlerlist=jdbcwriter
gg.handler.jdbcwriter.type=jdbc

#Handler properties for MySQL database target with JDBC Metadata provider
gg.handler.jdbcwriter.DriverClass=com.mysql.jdbc.Driver
gg.handler.jdbcwriter.connectionURL=jdbc:<a target="_blank"
href="mysql://">mysql://</a><DBServer address>:3306/<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/mysql/jdbc/driver//mysql-connector-java-5.1.39-bin.jar
#JDBC Metadata provider for MySQL target
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:<a target="_blank" href="mysql://">mysql://</a><DBServer
address>:3306/<database name>
gg.mdp.DriverClassName=com.mysql.jdbc.Driver
gg.mdp.UserName=<dbuser>
gg.mdp.Password=<dbpassword>

goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
```

```
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./
dirprm
```

10
Using the Kafka Handler

Learn how to use the Kafka Handler, which is designed to stream change capture data from an Oracle GoldenGate trail to a Kafka topic.

Topics:

- Overview (page 10-1)
- Detailed Functionality (page 10-1)
- Setting Up and Running the Kafka Handler (page 10-3)
- Schema Propagation (page 10-13)
- Performance Considerations (page 10-13)
- About Security (page 10-14)
- Metadata Change Events (page 10-14)
- Snappy Considerations (page 10-15)
- Troubleshooting (page 10-15)

10.1 Overview

The Oracle GoldenGate for Big Data Kafka Handler streams change capture data from an Oracle GoldenGate trail to a Kafka topic. Additionally, the Kafka Handler provides functionality to publish messages to a separate schema topic. Schema publication for Avro and JSON is supported.

Apache Kafka is an open source, distributed, partitioned, and replicated messaging service, see http://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.

In Kafka, when the topic name corresponds to the fully-qualified source table name, the Kafka Handler implements a Kafka producer. The Kafka producer writes serialized change data capture, from multiple source tables to either a single configured topic or separating source operations, to different Kafka topics.

10.2 Detailed Functionality

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

The following configuration sets the Kafka Handler to transaction mode:

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

In transaction mode, the serialized data is concatenated for every operation in a transaction from the source Oracle GoldenGate trail files. 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 data from 1 to *N* operations, where *N* is the number of operations in the transaction.

For grouped transactions, all the data for all the operations are concatenated into a single Kafka message. Therefore, grouped transactions may result in very large Kafka messages that contain data for a large number of operations.

Operation Mode

The following configuration sets the Kafka Handler to operation mode:

```
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 that there is a 1 to 1 relationship between the incoming operations and the number of Kafka messages produced.

Blocking Versus Non-Blocking Mode

The Kafka Handler can send messages to Kafka in either blocking mode (synchronous) or non-blocking mode (asynchronous).

Blocking Mode

The following configuration property sets the Kafka Handler to blocking mode:

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

Messages are delivered to Kafka on a synchronous basis. The Kafka Handler does 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 but at the cost of reduced performance.

You must *never* set the Kafka Producer `linger.ms` variable when in blocking mode, as this causes the Kafka producer to wait for the entire timeout period before sending the message to the Kafka broker. When this happens, the Kafka Handler waits for acknowledgement that the message has been sent while at the same time the Kafka Producer buffers messages to be sent to the Kafka brokers.

Non-Blocking Mode

The following configuration property sets the Kafka Handler to non-blocking mode:

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

Messages are delivered to Kafka asynchronously. 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, the Kafka producer flush call is invoked to ensure that all outstanding messages are transferred to the Kafka cluster. This allows the Kafka Handler to safely checkpoint, ensuring zero data loss. Invocation of the Kafka producer flush call is not affected by the `linger.ms` duration. This allows the Kafka Handler to safely checkpoint ensuring zero data loss.

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. 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 is sent to Kafka once the `batch.size` is reached or when the `linger.ms` period expires, whichever comes first. Setting the `batch.size` variable only ensures that messages are sent immediately to Kafka.

Topic Name Selection

The topic is resolved at runtime using this configuration parameter:

```
gg.handler.topicMappingTemplate
```

You can configure a static string, keywords, or a combination of static strings and keywords to dynamically resolve the topic name at runtime based on the context of the current operation, see Using Templates to Resolve the Topic Name and Message Key (page 11-9).

Kafka Broker Settings

To configure topics to be created automatically, set the `auto.create.topics.enable` property to `true`. This is the default setting.

If you set the `auto.create.topics.enable` property to `false`, then you must manually create topics before you start the Replicat process.

Schema Propagation

The schema data for all tables is delivered to the schema topic that is configured with the `schemaTopicName` property. For more information , see Schema Propagation (page 10-13).

10.3 Setting Up and Running the Kafka Handler

Instructions for configuring the Kafka Handler components and running the handler are described in this section.

You must install and correctly configure Kafka either as a single node or a clustered instance, see http://kafka.apache.org/documentation.html.

If you are using a Kafka distribution other than Apache Kafka, then consult the documentation for your Kafka distribution for installation and configuration instructions.

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

Oracle recommends and considers it best practice that the data topic and the schema topic (if applicable) are preconfigured on the running Kafka brokers. You can 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 Kafka Handler process, then the remote host port must be reachable from the machine running the Kafka Handler.

Topics:

- Classpath Configuration (page 10-4)
- Kafka Handler Configuration (page 10-4)
- Java Adapter Properties File (page 10-6)
- Kafka Producer Configuration File (page 10-7)
- Using Templates to Resolve the Topic Name and Message Key (page 10-7)
- Kafka Configuring with Kerberos on a Hadoop Platform (page 10-9)

10.3.1 Classpath Configuration

For the Kafka Handler to connect to Kafka and run, the Kafka Producer properties file and the Kafka client JARs must be configured in the `gg.classpath` configuration variable. The Kafka client JARs must match the version of Kafka that the Kafka Handler is connecting to. For a list of the required client JAR files by version, see Kafka Handler Client Dependencies (page G-1).

The recommended 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 precisely. The path of the Kafka Producer Properties file must contain the path with no wildcard appended. If the `*` wildcard is included in the path to the Kafka Producer Properties file, the file is not picked up. Conversely, path to the dependency JARs must include the `*` wild card character in order to include all 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={kafka install dir}/libs/*
```

10.3.2 Kafka Handler Configuration

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

To enable the selection of the Kafka Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=kafka` and the other Kafka properties as follows:

Table 10-1 Configuration Properties for Kafka Handler

Property Name	Required / Optional	Property Value	Default	Description
`gg.handlerlist`	Required	*name* (choice of any name)	None	List of handlers to be used.

Table 10-1 (Cont.) Configuration Properties for Kafka Handler

Property Name	Required / Optional	Property Value	Default	Description
`gg.handler.name.type`	Required	`kafka`	None	Type of handler to use.
`gg.handler.name.KafkaProducerConfigFile`	Optional	Any custom file name	`kafka-producer-default.properties`	Filename in classpath that holds Apache Kafka properties to configure the Apache Kafka producer.
`gg.handler.name.Format`	Optional	Formatter class or short code.	`delimitedtext`	Formatter to use to format payload. Can be one of `xml`, `delimitedtext`, `json`, `json_row`, `avro_row`, `avro_op`
`gg.handler.name.SchemaTopicName`	Required when schema delivery is required.	Name of the schema topic.	None	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.name.SchemaPrClassName`	Optional	Fully qualified class name of a custom class that implements Oracle GoldenGate for Big Data Kafka Handler's `CreateProducerRecord` Java Interface.	Provided this implementation class: `oracle.goldengate.handler.kafka` `ProducerRecord`	Schema is also propagated as a `ProducerRecord`. The default key 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.name.BlockingSend`	Optional	`true` | `false`	`false`	If this property is set to true, then delivery to Kafka works in a completely synchronous model. The next payload is sent only after the current payload has been written 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 10-1 (Cont.) Configuration Properties for Kafka Handler

Property Name	Required / Optional	Property Value	Default	Description
`gg.handler.name.mode`	Optional	`tx/op`	`tx`	With Kafka Handler operation mode, each change capture data record (Insert, Update, Delete, and so on) payload is represented as a Kafka Producer Record and is flushed one at a time. With Kafka Handler in transaction mode, all operations within a source transaction are represented as a single Kafka Producer record. This combined byte payload is flushed on a transaction Commit event.
`gg.handler.name.topicMappingTemplate`	Required	A template string value to resolve the Kafka topic name at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key (page 11-9).
`gg.handler.name.keyMappingTemplate`	Required	A template string value to resolve the Kafka message key at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key (page 11-9).
`gg.hander.name.logSuccessfullySentMessages`	Optional	`true | false`	`true`	Set to `true`, the Kafka Handler will log at the `INFO` level message that have been successfully sent to Kafka. Enabling this property has negative impact onnnperformance.

10.3.3 Java Adapter Properties File

The following is a sample configuration for the Kafka Handler from the Adapter properties file:

```
gg.handlerlist = kafkahandler
gg.handler.kafkahandler.Type = kafka
gg.handler.kafkahandler.KafkaProducerConfigFile = custom_kafka_producer.properties
gg.handler.kafkahandler.topicMappingTemplate=oggtopic
gg.handler.kafkahandler.keyMappingTemplate=${currentTimestamp}
gg.handler.kafkahandler.Format = avro_op
gg.handler.kafkahandler.SchemaTopicName = oggSchemaTopic
gg.handler.kafkahandler.SchemaPrClassName = com.company.kafkaProdRec.SchemaRecord
gg.handler.kafkahandler.Mode = tx
gg.handler.kafkahandler.BlockingSend = true
```

You can find a sample Replicat configuration and a Java Adapter Properties file for a Kafka integration in the following directory:

GoldenGate_install_directory/AdapterExamples/big-data/kafka

10.3.4 Kafka Producer Configuration File

The Kafka Handler must access a Kafka producer configuration file in order to 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 attempts to locate and load the Kafka producer configuration file by 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 uses 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 = 0
max.request.size = 1048576
send.buffer.bytes = 131072
```

10.3.5 Using Templates to Resolve the Topic Name and Message Key

The Kafka Handler provides functionality to resolve the topic name and the message key at runtime using a template configuration value. Templates allow you to configure static values and keywords. Keywords are used to dynamically replace the keyword with the context of the current processing. The templates use the following configuration properties:

```
gg.handler.name.topicMappingTemplate
gg.handler.name.keyMappingTemplate
```

Template Modes

Source database transactions are made up of one or more individual operations that are the individual inserts, updates, and deletes. The Kafka Handler can be configured to send one message per operation (insert, update, delete), or alternatively can be configured to group operations into messages at the transaction level. Many template keywords resolve data based on the context of an individual source database operation. Therefore, many of the keywords do *not* work when sending messages at the transaction level. For example, using `${fullyQualifiedTableName}` does not work when sending messages at the transaction level rather it resolves to the qualified source table name for an operation. However, transactions can contain multiple

operations for many source tables. Resolving the fully qualified table name for messages at the transaction level is non-deterministic so abends at runtime.

Template Keywords

This table includes a column if the keyword is supported for transaction level messages.

Keyword	Explanation	Transaction Message Support
`${fullyQualifiedTableName}`	Resolves to the fully qualified table name including the period (.) delimiter between the catalog, schema, and table names. For example, `test.dbo.table1`.	No
`${catalogName}`	Resolves to the catalog name.	No
`${schemaName}`	Resolves to the schema name.	No
`${tableName}`	Resolves to the short table name.	No
`${opType}`	Resolves to the type of the operation: (`INSERT`, `UPDATE`, `DELETE`, or `TRUNCATE`)	No
`${primaryKeys}`	Resolves to the concatenated primary key values delimited by an underscore (_) character.	No
`${position}`	The sequence number of the source trail file followed by the offset (RBA).	Yes
`${opTimestamp}`	The operation timestamp from the source trail file.	Yes
`${emptyString}`	Resolves to “”.	Yes
`${groupName}`	Resolves to the name of the Replicat process. If using coordinated delivery, it resolves to the name of the Replicat process with the Replicate thread number appended.	Yes
`${staticMap[]}`	Resolves to a static value where the key is the fully-qualified table name. The keys and values are designated inside of the square brace in the following format: `${staticMap[dbo.table1=value1,dbo.table2=value2]}`	No

Keyword	Explanation	Transaction Message Support
`${columnValue[]}`	Resolves to a column value where the key is the fully-qualified table name and the value is the column name to be resolved. For example: `${staticMap[dbo.table1=col1, dbo.table2=col2]}`	No

`${currentTimestamp}`
Or
`${currentTimestamp[]}`
	Resolves to the current timestamp. You can control the format of the current timestamp using the Java based formatting as described in the `SimpleDateFormat` class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
	Examples:
	`${currentDate}`
	`${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}`
		Yes

`${null}`	Resolves to a NULL string.	Yes
`${custom[]}`	It is possible to write a custom value resolver. If required, contact Oracle Support.	Implementation dependent

Example Templates

The following describes example template configuration values and the resolved values.

Example Template	Resolved Value
`${groupName}_{fullyQualfiedTableName}`	`KAFKA001_dbo.table1`
`prefix_${schemaName}_${tableName}_suffix`	`prefix_dbo_table1_suffix`
`${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}`	`2017-05-17 11:45:34.254`

10.3.6 Kafka Configuring with Kerberos on a Hadoop Platform

Use these steps to configure a Kafka Handler Replicat with Kerberos to enable a Cloudera instance to process an Oracle GoldenGate for Big Data trail to a Kafka topic:

1. In GGSCI, add a Kafka Replicat:

   ```
   GGSCI> add replicat kafka, exttrail dirdat/gg
   ```

2. Configure a `prm` file with these properties:

```
replicat kafka
discardfile ./dirrpt/kafkax.dsc, purge
SETENV (TZ=PST8PDT)
GETTRUNCATES
GETUPDATEBEFORES
ReportCount Every 1000 Records, Rate
MAP qasource.*, target qatarget.*;
```

3. Configure a Replicat properties file as follows:

```
###KAFKA Properties file ###
gg.log=log4j
gg.log.level=info
gg.report.time=30sec

###Kafka Classpath settings ###
gg.classpath=/opt/cloudera/parcels/KAFKA-2.1.0-1.2.1.0.p0.115/lib/kafka/libs/*
jvm.bootoptions=-Xmx64m -Xms64m -Djava.class.path=./ggjava/ggjava.jar -
Dlog4j.configuration=log4j.properties -Djava.security.auth.login.config=/scratch/
ydama/ogg/v123211/dirprm/jaas.conf -Djava.security.krb5.conf=/etc/krb5.conf

javawriter.stats.full=TRUE
javawriter.stats.display=TRUE

### native library config ###
goldengate.userexit.nochkpt=TRUE
goldengate.userexit.timestamp=utc

### Kafka handler properties ###
gg.handlerlist = kafkahandler
gg.handler.kafkahandler.type=kafka
gg.handler.kafkahandler.KafkaProducerConfigFile=kafka-producer.properties
gg.handler.kafkahandler.format=delimitedtext
gg.handler.kafkahandler.format.PkUpdateHandling=update
gg.handler.kafkahandler.mode=tx
gg.handler.kafkahandler.format.includeCurrentTimestamp=false
#gg.handler.kafkahandler.maxGroupSize=100
#gg.handler.kafkahandler.minGroupSize=50
gg.handler.kafkahandler.format.fieldDelimiter=|
gg.handler.kafkahandler.format.lineDelimiter=CDATA[\n]
gg.handler.kafkahandler.topicMappingTemplate=myoggtopic
gg.handler.kafkahandler.keyMappingTemplate=${position}
```

4. Configure a Kafka Producer file with these properties:

```
bootstrap.servers=10.245.172.52:9092
acks=1
#compression.type=snappy
reconnect.backoff.ms=1000
value.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
batch.size=1024
linger.ms=2000

security.protocol=SASL_PLAINTEXT

sasl.kerberos.service.name=kafka
sasl.mechanism=GSSAPI
```

5. Configure a `jaas.conf` file with these properties:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
```

```
useKeyTab=true
storeKey=true
keyTab="/scratch/ydama/ogg/v123211/dirtmp/keytabs/slc06unm/kafka.keytab"
principal="kafka/slc06unm.us.oracle.com@HADOOPTEST.ORACLE.COM";
};
```

6. Ensure that you have the latest `key.tab` files from the Cloudera instance to connect secured Kafka topics.
7. Start the Replicat from GGSCI and make sure that it is running with `INFO ALL`.
8. Review the Replicat report to see the total number of records processed. The report is similar to:

```
Oracle GoldenGate for Big Data, 12.3.2.1.1.005

Copyright (c) 2007, 2018. Oracle and/or its affiliates. All rights reserved

Built with Java 1.8.0_161 (class version: 52.0)

2018-08-05 22:15:28 INFO OGG-01815 Virtual Memory Facilities for: COM
anon alloc: mmap(MAP_ANON) anon free: munmap
file alloc: mmap(MAP_SHARED) file free: munmap
target directories:
/scratch/ydama/ogg/v123211/dirtmp.

Database Version:

Database Language and Character Set:

***********************************************************************
** Run Time Messages **
***********************************************************************


2018-08-05 22:15:28 INFO OGG-02243 Opened trail file /scratch/ydama/ogg/v123211/
dirdat/kfkCustR/gg000000 at 2018-08-05 22:15:28.258810.

2018-08-05 22:15:28 INFO OGG-03506 The source database character set, as
determined from the trail file, is UTF-8.

2018-08-05 22:15:28 INFO OGG-06506 Wildcard MAP resolved (entry qasource.*): MAP
"QASOURCE"."BDCUSTMER1", target qatarget."BDCUSTMER1".

2018-08-05 22:15:28 INFO OGG-02756 The definition for table QASOURCE.BDCUSTMER1
is obtained from the trail file.

2018-08-05 22:15:28 INFO OGG-06511 Using following columns in default map by
name: CUST_CODE, NAME, CITY, STATE.

2018-08-05 22:15:28 INFO OGG-06510 Using the following key columns for target
table qatarget.BDCUSTMER1: CUST_CODE.

2018-08-05 22:15:29 INFO OGG-06506 Wildcard MAP resolved (entry qasource.*): MAP
"QASOURCE"."BDCUSTORD1", target qatarget."BDCUSTORD1".

2018-08-05 22:15:29 INFO OGG-02756 The definition for table QASOURCE.BDCUSTORD1
is obtained from the trail file.

2018-08-05 22:15:29 INFO OGG-06511 Using following columns in default map by
name: CUST_CODE, ORDER_DATE, PRODUCT_CODE, ORDER_ID, PRODUCT_PRICE,
PRODUCT_AMOUNT, TRANSACTION_ID.
```

```
2018-08-05 22:15:29 INFO OGG-06510 Using the following key columns for target
table qatarget.BDCUSTORD1: CUST_CODE, ORDER_DATE, PRODUCT_CODE, ORDER_ID.

2018-08-05 22:15:33 INFO OGG-01021 Command received from GGSCI: STATS.

2018-08-05 22:16:03 INFO OGG-01971 The previous message, 'INFO OGG-01021',
repeated 1 times.

2018-08-05 22:43:27 INFO OGG-01021 Command received from GGSCI: STOP.

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

Last record for the last committed transaction is the following:
_______________________________________________________________________
Trail name : /scratch/ydama/ogg/v123211/dirdat/kfkCustR/gg000000
Hdr-Ind : E (x45) Partition : . (x0c)
UndoFlag : . (x00) BeforeAfter: A (x41)
RecLength : 0 (x0000) IO Time : 2015-08-14 12:02:20.022027
IOType : 100 (x64) OrigNode : 255 (xff)
TransInd : . (x03) FormatType : R (x52)
SyskeyLen : 0 (x00) Incomplete : . (x00)
AuditRBA : 78233 AuditPos : 23968384
Continued : N (x00) RecCount : 1 (x01)

2015-08-14 12:02:20.022027 GGSPurgedata Len 0 RBA 6473
TDR Index: 2
_______________________________________________________________________

Reading /scratch/ydama/ogg/v123211/dirdat/kfkCustR/gg000000, current RBA 6556,
20 records, m_file_seqno = 0, m_file_rba = 6556

Report at 2018-08-05 22:43:27 (activity since 2018-08-05 22:15:28)

From Table QASOURCE.BDCUSTMER1 to qatarget.BDCUSTMER1:
# inserts: 5
# updates: 1
# deletes: 0
# discards: 0
From Table QASOURCE.BDCUSTORD1 to qatarget.BDCUSTORD1:
# inserts: 5
# updates: 3
# deletes: 5
# truncates: 1
# discards: 0
```

9. Ensure that the secure Kafka topic is created:

   ```
   /kafka/bin/kafka-topics.sh --zookeeper slc06unm:2181 --list
   myoggtopic
   ```

10. Review the contents of the secure Kafka topic:

    a. Create a `consumer.properties` file containing:

       ```
       security.protocol=SASL_PLAINTEXT
       sasl.kerberos.service.name=kafka
       ```

    b. Set this environment variable:

```
export KAFKA_OPTS="-Djava.security.auth.login.config="/scratch/ogg/v123211/
dirprm/jaas.conf"
```

c. Run the consumer utility to check the records:

```
/kafka/bin/kafka-console-consumer.sh --bootstrap-server sys06:9092 --topic
myoggtopic --new-consumer --consumer.config consumer.properties
```

10.4 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 that are enabled for schema publishing. If the Kafka Handler `schemaTopicName` property is set, then the schema is published for the following events:

- The Avro schema for a specific table is published the first time an operation for that table is encountered.
- If the Kafka Handler receives a metadata change event, the schema is flushed. The regenerated Avro schema for a specific table is published the next time an operation for that table is encountered.
- If the Avro wrapping functionality is enabled, then the generic wrapper Avro schema is published the first time that any operation is encountered. To enable the generic wrapper, Avro schema functionality is enabled in the Avro formatter configuration, see Avro Row Formatter (page 17-1) and The Avro Operation Formatter (page 17-10).

The Kafka `ProducerRecord` value is the schema, and the key is the fully qualified table name.

Because Avro messages directly depend on an Avro schema, user of Avro over Kafka may encounter issues. Avro messages are not human readable because they are binary. To deserialize an Avro message, the receiver must first have the correct Avro schema, but because each table from the source database results in a separate Avro schema, this can be difficult. The receiver of a Kafka message cannot determine which Avro schema to use to deserialize individual messages when the source Oracle GoldenGate trail file includes operations from multiple tables. To solve this problem, you can 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.

10.5 Performance Considerations

Oracle recommends that you do *not* use the `linger.ms` setting in the Kafka producer `config` file when `gg.handler.name.BlockingSend`.is set to `true`. This causes each send to block for at least the value of `linger.ms`, leading to major performance issues because the Kafka Handler configuration and the Kafka Producer configuration are in conflict with each other. This configuration results in a temporary deadlock scenario, where the Kafka Handler is waits to received a send acknowledgement while the Kafka producer waits for more messages before sending. The deadlock resolves when the `linger.ms` period expires. This behavior repeats for every message sent.

For the best performance, Oracle recommends that you set the Kafka Handler to operate in operation mode using non-blocking (asynchronous) calls to the Kafka producer. Use the following configuration in your Java Adapter properties file:

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

Additionally, Oracle recommends that you set the `batch.size` and linger.ms values in the Kafka Producer properties file. These values are highly dependent upon the use case scenario. Typically, higher values result in better throughput, but latency is increased. Smaller values in these properties reduces latency but overall throughput decreases. If you have a high volume of input data from the source trial files, then set the `batch.size` and `linger.ms` size as high as possible.

Use of the Replicat variable `GROUPTRANSOPS` also improves performance. The recommended setting is `10000`.

If the serialized operations from the source trail file must be delivered in individual Kafka messages, then the Kafka Handler must be set to operation mode.

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

However, the result is many more Kafka messages and adversely affected performance.

10.6 About Security

Kafka version 0.9.0.0 introduced security through SSL/TLS and SASL (Kerberos). You can secure the Kafka Handler using one or both of the SSL/TLS and SASL security offerings. The Kafka producer client libraries provide an abstraction of security functionality from the integrations that use those libraries. The 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 with the required security properties. For detailed instructions about securing the Kafka cluster, see the Kafka documentation at

You may encounter the inability to decrypt the Kerberos password from the `keytab` file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html.

10.7 Metadata Change Events

Metadata change events are now handled in the Kafka Handler. This is relevant only if you have 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 is published to the schema topic.

To support metadata change events, the Oracle GoldenGate process capturing changes in the source database must support the Oracle GoldenGate metadata in trail feature, which was introduced in Oracle GoldenGate 12*c* (12.2).

10.8 Snappy Considerations

The Kafka Producer Configuration file supports the use of compression. One of the configurable options is Snappy, an open source compression and decompression (`codec`) library that provides better performance than other `codec` libraries. The Snappy JAR does not run on all platforms. Snappy may work on Linux systems though may or may not work on other UNIX and Windows implementations. If you want to use Snappy compression, test Snappy on all required systems before implementing compression using Snappy. If Snappy does not port to all required systems, then Oracle recommends using an alternate `codec` library.

10.9 Troubleshooting

Topics:

- Verify the Kafka Setup (page 10-15)
- Classpath Issues (page 10-15)
- Invalid Kafka Version (page 10-15)
- Kafka Producer Properties File Not Found (page 10-15)
- Kafka Connection Problem (page 10-16)

10.9.1 Verify the Kafka Setup

You can use the command line Kafka producer to write dummy data to a Kafka topic, and you can use a Kafka consumer to read this data from the Kafka topic. Use this method to verify the setup and read/write permissions to Kafka topics on disk, see http://kafka.apache.org/documentation.html#quickstart.

10.9.2 Classpath Issues

Java classpath problems are common. Such problems may include a `ClassNotFoundException` problem in the `log4j` log file or may be an error resolving the classpath because of a typographic error in the `gg.classpath` variable. The Kafka client libraries do *not* ship with the Oracle GoldenGate for Big Data product. You must obtain the correct version of the Kafka client libraries and properly configure the `gg.classpath` property in the Java Adapter Properties file to correctly resolve the Java the Kafka client libraries as described in Classpath Configuration (page 10-4).

10.9.3 Invalid Kafka Version

The Kafka Handler does *not* support Kafka versions 0.8.2.2 or older. If you run an unsupported version of Kafka, a runtime Java exception, `java.lang.NoSuchMethodError`, occurs. It implies that the `org.apache.kafka.clients.producer.KafkaProducer.flush()` method cannot be found. If you encounter this error, migrate to Kafka version 0.9.0.0 or later.

10.9.4 Kafka Producer Properties File Not Found

This problem typically results in the following exception:

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

Check the `gg.handler.kafkahandler.KafkaProducerConfigFile` configuration variable to ensure that the Kafka Producer Configuration file name is set correctly. Check the `gg.classpath` variable 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.

10.9.5 Kafka Connection Problem

This problem occurs when the Kafka Handler is unable to connect to Kafka. You receive 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
```

The connection retry interval expires, and the Kafka Handler process abends. Ensure that the Kafka Broker is running and that the host and port provided in the Kafka Producer Properties file are correct. You can use network shell commands (such as `netstat -l`) on the machine hosting the Kafka broker to verify that Kafka is listening on the expected port.

11

Using the Kafka Connect Handler

Learn how to use the Kafka Connect Handler, which is an extension of the standard Kafka messaging functionality.

Topics:

- Overview (page 11-1)
- Detailed Functionality (page 11-1)
- Setting Up and Running the Kafka Connect Handler (page 11-3)
- Kafka Connect Handler Performance Considerations (page 11-11)
- Troubleshooting the Kafka Connect Handler (page 11-11)

11.1 Overview

The Oracle GoldenGate Kafka Connect is an extension of the standard Kafka messaging functionality. Kafka Connect is a functional layer on top of the standard Kafka Producer and Consumer interfaces. It provides standardization for messaging to make it easier to add new source and target systems into your topology.

Confluent is a primary adopter of Kafka Connect and their Confluent Platform offering includes extensions over the standard Kafka Connect functionality. This includes Avro serialization and deserialization, and an Avro schema registry. Much of the Kafka Connect functionality is available in Apache Kafka. A number of open source Kafka Connect integrations are found at:

https://www.confluent.io/product/connectors/

The Kafka Connect Handler is a Kafka Connect source connector. You can capture database changes from any database supported by Oracle GoldenGate and stream that change of data through the Kafka Connect layer to Kafka. You can also connect to Oracle Event Hub Cloud Services (EHCS) with this handler.

Kafka Connect uses proprietary objects to define the schemas (`org.apache.kafka.connect.data.Schema`) and the messages (`org.apache.kafka.connect.data.Struct`). The Kafka Connect Handler can be configured to manage what data is published and the structure of the published data.

The Kafka Connect Handler does *not* support any of the pluggable formatters that are supported by the Kafka Handler.

Topics:

11.2 Detailed Functionality

The Kafka Connect framework provides converters to convert in-memory Kafka Connect messages to a serialized format suitable for transmission over a network. These converters are selected using configuration in the Kafka Producer properties file.

JSON Converter

Kafka Connect and the JSON converter is available as part of the Apache Kafka download. The JSON Converter converts the Kafka keys and values to JSONs which are then sent to a Kafka topic. You identify the JSON Converters with the following configuration in the Kafka Producer properties file:

```
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true
```

The format of the messages is the message schema information followed by the payload information. JSON is a self describing format so you should not include the schema information in each message published to Kafka.

To omit the JSON schema information from the messages set the following:

```
key.converter.schemas.enable=false
value.converter.schemas.enable=false
```

Avro Converter

Confluent provides Kafka installations, support for Kafka, and extended functionality built on top of Kafka to help realize the full potential of Kafka. Confluent provides both open source versions of Kafka (Confluent Open Source) and an enterprise edition (Confluent Enterprise), which is available for purchase.

A common Kafka use case is to send Avro messages over Kafka. This can create a problem on the receiving end as there is a dependency for the Avro schema in order to deserialize an Avro message. Schema evolution can increase the problem because received messages must be matched up with the exact Avro schema used to generate the message on the producer side. Deserializing Avro messages with an incorrect Avro schema can cause runtime failure, incomplete data, or incorrect data. Confluent has solved this problem by using a schema registry and the Confluent schema converters.

The following shows the configuration of the Kafka Producer properties file.

```
key.converter=io.confluent.connect.avro.AvroConverter
value.converter=io.confluent.connect.avro.AvroConverter
key.converter.schema.registry.url=http://localhost:8081
value.converter.schema.registry.url=http://localhost:8081
```

When messages are published to Kafka, the Avro schema is registered and stored in the schema registry. When messages are consumed from Kafka, the exact Avro schema used to create the message can be retrieved from the schema registry to deserialize the Avro message. This creates matching of Avro messages to corresponding Avro schemas on the receiving side, which solves this problem.

Following are the requirements to use the Avro Converters:

- This functionality is available in both versions of Confluent Kafka (open source or enterprise).
- The Confluent schema registry service must be running.
- Source database tables must have an associated Avro schema. Messages associated with different Avro schemas must be sent to different Kafka topics.

- The Confluent Avro converters and the schema registry client must be available in the classpath.

The schema registry keeps track of Avro schemas by topic. Messages must be sent to a topic that has the same schema or evolving versions of the same schema. Source messages have Avro schemas based on the source database table schema so Avro schemas are unique for each source table. Publishing messages to a single topic for multiple source tables will appear to the schema registry that the schema is evolving every time the message sent from a source table that is different from the previous message.

11.3 Setting Up and Running the Kafka Connect Handler

Instructions for configuring the Kafka Connect Handler components and running the handler are described in this section.

Classpath Configuration

Two things must be configured in the `gg.classpath` configuration variable so that the Kafka Connect Handler can 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 Connect Handler is connecting to. For a listing of the required client JAR files by version, see Kafka Handler Client Dependencies Kafka Connect Handler Client Dependencies (page H-1). The recommended storage location for the Kafka Producer properties file is the Oracle GoldenGate `dirprm` directory.

The default location of the Kafka Connect client JARs is the `Kafka_Home/libs/*` directory.

The `gg.classpath` variable must be configured precisely. Pathing to the Kafka Producer properties file should contain the path with no wildcard appended. The inclusion of the asterisk (*) wildcard in the path to the Kafka Producer properties file causes it to be discarded. Pathing to the dependency JARs should include the * wildcard character to include all of the JAR files in that directory in the associated classpath. Do not use `*.jar`.

Following is an example of a correctly configured Apache Kafka classpath:

```
gg.classpath=dirprm:{kafka_install_dir}/libs/*
```

Following is an example of a correctly configured Confluent Kafka classpath:

```
gg.classpath={confluent_install_dir}/share/java/kafka-serde-tools/*:
{confluent_install_dir}/share/java/kafka/*:{confluent_install_dir}/share/java/
confluent-common/*
```

Topics:

- Kafka Connect Handler Configuration (page 11-4)
- Using Templates to Resolve the Topic Name and Message Key (page 11-9)
- Configuring Security in the Kafka Connect Handler (page 11-10)

11.3.1 Kafka Connect Handler Configuration

The following are the configurable values for the Kafka Connect Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Kafka Connect Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=kafkaconnect` and the other Kafka Connect properties as follows:

Table 11-1 Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.type`	Required	`kafkaconnect`	None	The configuration to select the Kafka Connect Handler.
`gg.handler.name.kafkaProducerConfigFile`	Required	string	None	A path to a properties file containing the properties of the Kafka and Kafka Connect configuration properties.
`gg.handler.name.topicMappingTemplate`	Required	A template string value to resolve the Kafka topic name at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key (page 11-9).
`gg.handler.name.keyMappingTemplate`	Required	A template string value to resolve the Kafka message key at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key (page 11-9).
`gg.handler.name.includeTableName`	Optional	`true` | `false`	`true`	Set to `true` to create a field in the output messages called “table” for which the value is the fully qualified table name. Set to `false` to omit this field in the output.

Table 11-1 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.includeOpType`	Optional	`true` | `false`	`true`	Set to `true` to create a field in the output messages called `op_type` for which the value is is an indicator of the type of source database operation (for example, `I` for insert, `U` for update, and `D`for delete). Set to `false` to omit this field in the output.
`gg.handler.name.includeOpTimestamp`	Optional	`true` | `false`	`true`	Set to `true` to create a field in the output messages called `op_ts` for which the value is the operation timestamp (commit timestamp) from the source trail file. Set to `false` to omit this field in the output.
`gg.handler.name.includeCurrentTimestamp`	Optional	`true` | `false`	`true`	Set to `true` to create a field in the output messages called `current_ts` for which the value is the current timestamp of when the handler processes the operation. Set to `false` to omit this field in the output.

Table 11-1 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.includePosition`	Optional	`true` | `false`	`true`	Set to `true` to create a field in the output messages called `pos` for which the value is the position (sequence number + offset) of the operation from the source trail file. Set to `false` to omit this field in the output.
`gg.handler.name.includePrimaryKeys`	Optional	`true` | `false`	`false`	Set to `true` to include a field in the message called `primary_keys` and the value of which is an array of the column names of the primary key columns. Set to `false` to suppress this field.
`gg.handler.name.includeTokens`	Optional	`true` | `false`	`false`	Set to `true` to include a map field in output messages. The key is **tokens** and the value is a map where the keys and values are the token keys and values from the Oracle GoldenGate source trail file. Set to `false` to suppress this field.

Table 11-1 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.messageFormatting`	Optional	`row` | `op`	`row`	Controls how output messages are modeled. Selecting row and the output messages will be modeled as row. Set to op and the output messages will be modeled as operations messages.
`gg.handler.name.insertOpKey`	Optional	any string	`I`	The value of the field `op_type` to indicate an insert operation.
`gg.handler.name.updateOpKey`	Optional	any string	`U`	The value of the field `op_type` to indicate an insert operation.
`gg.handler.name.deleteOpKey`	Optional	any string	`D`	The value of the field `op_type` to indicate a delete operation.
`gg.handler.name.truncateOpKey`	Optional	any string	`T`	The value of the field `op_type` to indicate a truncate operation.
`gg.handler.name.treatAllColumnsAsStrings`	Optional	`true` | `false`	`false`	Set to true to treat all output fields as strings. Set to false and the Handler will map the corresponding field type from the source trail file to the best corresponding Kafka Connect data type.

Table 11-1 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.mapLargeNumbersAsStrings`	Optional	`true | false`	`false`	Large numbers are mapping to number fields as Doubles. It is possible to lose precision in certain scenarios. If set to `true` these fields will be mapped as Strings in order to preserve precision.
`gg.handler.name.iso8601Format`	Optional	`True | False`	`false`	Set to true to output the current date in the ISO8601 format.
`gg.handler.name.pkUpdateHandling`	Optional	`insert | abend | update | delete`	`abend`	Only applicable if modeling row messages `gg.handler.name.messageFormatting=row`. Not applicable if modeling operations messages as the before and after images are propagated to the message in the case of an update.

See Using Templates to Resolve the Stream Name and Partition Name (page 13-8) for more information.

Review a Sample Configuration

```
gg.handlerlist=kafkaconnect

#The handler properties
gg.handler.kafkaconnect.type=kafkaconnect
gg.handler.kafkaconnect.kafkaProducerConfigFile=kafkaconnect.properties
gg.handler.kafkaconnect.mode=op
#The following selects the topic name based on the fully qualified table name
gg.handler.kafkaconnect.topicMappingTemplate=$

{fullyQualifiedTableName}
#The following selects the message key using the concatenated primary keys
gg.handler.kafkaconnect.keyMappingTemplate=$

{primaryKeys}
```

```
#The formatter properties
gg.handler.kafkaconnect.messageFormatting=row
gg.handler.kafkaconnect.insertOpKey=I
gg.handler.kafkaconnect.updateOpKey=U
gg.handler.kafkaconnect.deleteOpKey=D
gg.handler.kafkaconnect.truncateOpKey=T
gg.handler.kafkaconnect.treatAllColumnsAsStrings=false
gg.handler.kafkaconnect.iso8601Format=false
gg.handler.kafkaconnect.pkUpdateHandling=abend
gg.handler.kafkaconnect.includeTableName=true
gg.handler.kafkaconnect.includeOpType=true
gg.handler.kafkaconnect.includeOpTimestamp=true
gg.handler.kafkaconnect.includeCurrentTimestamp=true
gg.handler.kafkaconnect.includePosition=true
gg.handler.kafkaconnect.includePrimaryKeys=false
gg.handler.kafkaconnect.includeTokens=false
```

11.3.2 Using Templates to Resolve the Topic Name and Message Key

The Kafka Connect Handler provides functionality to resolve the topic name and the message key at runtime using a template configuration value. Templates allow you to configure static values and keywords. Keywords are used to dynamically replace the keyword with the context of the current processing. Templates are applicable to the following configuration parameters:

```
gg.handler.name.topicMappingTemplate
gg.handler.name.keyMappingTemplate
```

Template Modes

The Kafka Connect Handler can only send operation messages. The Kafka Connect Handler cannot group operation messages into a larger transaction message.

Template Keywords

Keyword	Explanation
${*fullyQualifiedTableName*}	Resolves to the fully qualified table name including the period (.) delimiter between the catalog, schema, and table names. For example, `test.dbo.table1`.
${*catalogName*}	Resolves to the catalog name.
${*schemaName*}	Resolves to the schema name.
${*tableName*}	Resolves to the short table name.
${opType}	Resolves to the type of the operation: (`INSERT`, `UPDATE`, `DELETE`, or `TRUNCATE`)
${primaryKeys}	Resolves to the concatenated primary key values delimited by an underscore (_) character.
${position}	The sequence number of the source trail file followed by the offset (RBA).
${opTimestamp}	The operation timestamp from the source trail file.
${emptyString}	Resolves to “”.

Keyword	Explanation
${*groupName*}	Resolves to the name of the Replicat process. If using coordinated delivery, it resolves to the name of the Replicat process with the Replicate thread number appended.
${staticMap[]}	Resolves to a static value where the key is the fully-qualified table name. The keys and values are designated inside of the square brace in the following format: ${staticMap[dbo.table1=value1,dbo.table2=value2]}
${columnValue[]}	Resolves to a column value where the key is the fully-qualified table name and the value is the column name to be resolved. For example: ${staticMap[dbo.table1=col1,dbo.table2=col2]}
${currentTimestamp} Or ${currentTimestamp[]}	Resolves to the current timestamp. You can control the format of the current timestamp using the Java based formatting as described in the SimpleDateFormat class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html. Examples: ${currentDate} ${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}
${null}	Resolves to a NULL string.
${custom[]}	It is possible to write a custom value resolver. If required, contact Oracle Support.

Example Templates

The following describes example template configuration values and the resolved values.

Example Template	Resolved Value
${groupName}_{*fullyQualfiedTableName*}	KAFKA001_dbo.table1
prefix_${*schemaName*}_${*tableName*}_suffix	prefix_dbo_table1_suffix
${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}	2017-05-17 11:45:34.254

11.3.3 Configuring Security in the Kafka Connect Handler

Kafka version 0.9.0.0 introduced security through SSL/TLS or Kerberos. The Kafka Connect 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 Kafka Connect 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 Kafka Handler uses for processing, with the required security properties.

You may encounter the inability to decrypt the Kerberos password from the `keytab` file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html.

11.4 Kafka Connect Handler Performance Considerations

There are multiple configuration settings both for the Oracle GoldenGate for Big Data configuration and in the Kafka producer which affect performance.

The Oracle GoldenGate parameter have the greatest affect on performance is the Replicat `GROUPTRANSOPS` parameter. The `GROUPTRANSOPS` parameter allows Replicat to group multiple source transactions into a single target transaction. At transaction commit, the Kafka Connect Handler calls flush on the Kafka Producer to push the messages to Kafka for write durability followed by a checkpoint. The flush call is an expensive call and setting the Replicat `GROUPTRANSOPS` setting to larger amount allows the replicat to call the flush call less frequently thereby improving performance.

The default setting for `GROUPTRANSOPS` is `1000` and performance improvements can be obtained by increasing the value to 2500, 5000, or even 10000.

The Op mode `gg.handler.kafkaconnect.mode=op` parameter can also improve performance than the Tx mode `gg.handler.kafkaconnect.mode=tx`.

A number of Kafka Producer properties can affect performance. The following are the parameters with significant impact:

- `linger.ms`
- `batch.size`
- `acks`
- `buffer.memory`
- `compression.type`

Oracle recommends that you start with the default values for these parameters and perform performance testing to obtain a base line for performance. Review the Kafka documentation for each of these parameters to understand its role and adjust the parameters and perform additional performance testing to ascertain the performance effect of each parameter.

11.5 Troubleshooting the Kafka Connect Handler

Topics:

- Java Classpath for Kafka Connect Handler (page 11-12)
- Invalid Kafka Version (page 11-12)
- Kafka Producer Properties File Not Found (page 11-12)
- Kafka Connection Problem (page 11-12)

11.5.1 Java Classpath for Kafka Connect Handler

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 or and error while resolving the classpath if there is a typographic error in the `gg.classpath` variable.

The Kafka client libraries do not ship with the Oracle GoldenGate for Big Data product. You are required 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 as described in Setting Up and Running the Kafka Connect Handler (page 11-3).

11.5.2 Invalid Kafka Version

Kafka Connect was introduced in Kafka 0.9.0.0 version. The Kafka Connect Handler does not work with Kafka versions 0.8.2.2 and older. Attempting to use Kafka Connect with Kafka 0.8.2.2 version typically results in a `ClassNotFoundException` error at runtime.

11.5.3 Kafka Producer Properties File Not Found

Typically, the following exception message occurs:

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

Verify that the `gg.handler.kafkahandler.KafkaProducerConfigFile` configuration property for the Kafka Producer Configuration file name is set correctly.

Ensure that the `gg.classpath` variable 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.

11.5.4 Kafka Connection Problem

Typically, the following exception message appears:

```
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
```

When this occurs, the connection retry interval expires and the Kafka Connection Handler process abends. Ensure that the Kafka Brokers are 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.

12

Using the Kafka REST Proxy Handler

Learn how to use the Kafka REST Proxy Handler to stream messages to the Kafka REST Proxy distributed by Confluent.

Topics:

- Overview (page 12-1)
- Setting Up and Starting the Kafka REST Proxy Handler Services (page 12-1)
- Consuming the Records (page 12-14)
- Performance Considerations (page 12-15)
- Kafka REST Proxy Handler Metacolumns Template Property (page 12-15)

12.1 Overview

The Kafka REST Proxy Handler allows Kafka messages to be streamed using an HTTPS protocol. The use case for this functionality is to stream Kafka messages from an Oracle GoldenGate On Premises installation to cloud or alternately from cloud to cloud.

The Kafka REST proxy provides a RESTful interface to a Kafka cluster. It makes it easy for you to:

- produce and consume messages,
- view the state of the cluster,
- and perform administrative actions without using the native Kafka protocol or clients.

Kafka REST Proxy is part of the Confluent Open Source and Confluent Enterprise distributions. It is not available in the Apache Kafka distribution. To access Kafka through the REST proxy, you have to install the Confluent Kafka version see https://docs.confluent.io/current/kafka-rest/docs/index.html.

12.2 Setting Up and Starting the Kafka REST Proxy Handler Services

You must download and install the Confluent Open Source or Confluent Enterprise Distribution. You have several installation formats to choose from including ZIP or tar archives, Docker, and Packages.

- Using the Kafka REST Proxy Handler (page 12-2)
- Kafka REST Proxy Handler Configuration (page 12-2)
- Security (page 12-4)
- Generating a Keystore (page 12-5)

- Using Templates to Resolve the Topic Name and Message Key (page 12-5)
- Kafka REST Proxy Handler Formatter Properties (page 12-8)
- Setting Metacolumn Output (page 12-11)

12.2.1 Using the Kafka REST Proxy Handler

You must download and install the Confluent Open Source or Confluent Enterprise Distribution because the Kafka REST Proxy is not included in Apache, Cloudera, or Hortonworks. You have several installation formats to choose from including ZIP or TAR archives, Docker, and Packages.

The Kafka REST Proxy has dependency on ZooKeeper, Kafka, and the Schema Registry

12.2.2 Kafka REST Proxy Handler Configuration

The following are the configurable values for the Kafka REST Proxy Handler. Oracle recommend that you store the Kafka REST Proxy properties file in the Oracle GoldenGate `dirprm` directory.

To enable the selection of the Kafka REST Proxy Handler, you must first configure the handler type by specifying `gg.handler.name.type=kafkarestproxy` and the other Kafka REST Proxy Handler properties as follows:

Table 12-1 Kafka REST Proxy Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.type`	Required	`kafkarestproxy`	None	The configuration to select the Kafka REST Proxy Handler.
`gg.handler.name.topicMappingTemplate`	Required	A template string value to resolve the Kafka topic name at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key (page 11-9).
`gg.handler.name.keyMappingTemplate`	Required	A template string value to resolve the Kafka message key at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key (page 11-9).
`gg.handler.name.postDataUrl`	Required	The Listener address of the Rest Proxy.	None	Set to the URL of the Kafka REST proxy.
`gg.handler.name.format`	Required	`avro` | `json`	None	Set to the REST proxy payload data format

Table 12-1 (Cont.) Kafka REST Proxy Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.payloadsize`	Optional	A value representing the payload size in mega bytes.	`5MB`	Set to the maximum size of the payload of the HTTP messages.
`gg.handler.name.circularRedirectsAllowed`	Optional	`true` | `false`	`false`	Set to allow or disallow circular redirects.
`gg.handler.name.connectionRequestTimeout`	Optional	A value representing milliseconds.	`–1`	Set the maximum time to wait for the connection manager to return a connection. The connection manager may not be able to return a connection if the max number of connections in the pool are used.
`gg.handler.name.connectTimeout`	Optional	`true` | `false`	`-1`	Set the timeout in milliseconds until a connection is established.
`gg.handler.name.contentCompressionEnabled`	Optional	`true` | `false`	`true`	Sets content compression to on or off.
`gg.handler.name.maxRedirects`	Optional	Integer value representing the redirect count.	`50`	Sets the maximum number of redirects.
`gg.handler.name.proxy`	Optional	`host:port`	None	Sets the proxy.
`gg.handler.name.userName`	Optional	Any string.	None	Sets the username for the proxy authentication.
`gg.handler.name.password`	Optional	Any string.	None	Sets the password for the proxy authentication.
`gg.handler.name.redirectsEnabled`	Optional	`true` | `false`	`true`	Set to check if redirects using relative naming is enabled.
`gg.handler.name.relativeRedirectsAllowed`	Optional	`true` | `false`	`true`	Set to check if redirects is enabled.

Table 12-1 (Cont.) Kafka REST Proxy Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.socketTimeout`	Optional	A value representing milliseconds.	`–1`	Set the maximum time allowable between data packets on a read.
`gg.handler.name.httpClientResetInterval`	Optional	A value representing milliseconds.	`0`	Sets the wait interval between when the HTTP client is destroyed and when it is recreated.
`gg.handler.name.apiVersion`	Optional	`v1` | `v2`	`v2`	Sets the API version to use.
`gg.handler.name.mode`	Optional	`op` | `tx`	`op`	Sets how operations are processed. In `op` mode, operations are processed as they are received. In `tx` mode, operations are cached and processed at the transaction commit.

See Using Templates to Resolve the Stream Name and Partition Name (page 13-8) for more information.

12.2.3 Security

REST Proxy supports SSL for securing communication between clients and the Kafka REST Proxy Handler. To configure SSL:

1. Generate a keystore using the keytool for the server. This is also the client-side truststore. Follow the instructions to generate KeyStore in session Procedure to generate KeyStore.
2. Update the Kafka REST Proxy server configuration, `kafka-rest.properties` file with these properties:

```
listeners=https://server.domain.com:8082
ssl.keystore.location= path_of_serverkeystore.jks file
ssl.keystore.password=kafkarest
ssl.key.password=kafkarest
ssl.client.auth=false
```

3. Restart your server.
4. Append the client-side boot options, in the handler properties file, with:

```
-Djavax.net.ssl.trustStore=path_of_serverkeystore.jks file
-Djavax.net.ssl.trustStorePassword=pwd
```

For example, in the `krp.properties` file:

```
javawriter.bootoptions=-Xmx512m-Xms32m
-Djava.class.path=.:ggjava/ggjava.jar:./dirprm
-Djavax.net.ssl.trustStore=/scratch/sabbabu/view_storage/serverkeystore.jks
-Djavax.net.ssl.trustStorePassword=kafkarest
```

12.2.4 Generating a Keystore

You generate the `keystore.jks` keystore file by executing this statement from the command line:

```
keytool -genkey -keyalg RSA -alias fun -keystore serverkeystore.jks -validity 365 -
keysize 2048
```

The first name and last name should be the server machine name.

For example:

```
$ keytool -genkey -keyalg RSA -alias fun –keystore ~/serverkeystore.jks -validity
365 -keysize 2048
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  kkm00cfb.in.oracle.com
What is the name of your organizational unit?
  [Unknown]:  goldengate
What is the name of your organization?
  [Unknown]:  oracle
What is the name of your City or Locality?
  [Unknown]:  bangalore
What is the name of your State or Province?
  [Unknown]:  karnataka
What is the two-letter country code for this unit?
  [Unknown]:  in
Is CN=kkm00cfb.in.oracle.com , OU=goldengate, O=oracle, L=bangalore, ST= karnataka,
C=in correct?
  [no]:  yes

Enter key password for fun
(RETURN if same as keystore password):
```

This creates a file called `serverkeystore.jks`. There are two passwords that you need to provide. The keystore password and the key password. These passwords can be the same or different.

You update the Kafka HTTP proxy server configuration file, `kafka-rest.properties`, and then restart your REST server

12.2.5 Using Templates to Resolve the Topic Name and Message Key

The Kafka REST Proxy Handler provides functionality to resolve the topic name and the message key at runtime using a template configuration value. Templates allow you to configure static values and keywords. Keywords are used to dynamically replace the keyword with the context of the current processing. The templates use the following configuration properties:

```
gg.handler.name.topicMappingTemplate
gg.handler.name.keyMappingTemplate
```

Template Modes

The Kafka REST Proxy Handler can be configured to send one message per operation (insert, update, delete). Alternatively, it can be configured to group operations into messages at the transaction level.

Template Keywords

This table includes a column if the keyword is supported for transaction level messages.

Keyword	Explanation	Transaction Message Support
`${fullyQualifiedTableName}`	Resolves to the fully qualified table name including the period (.) delimiter between the catalog, schema, and table names. For example, `test.dbo.table1`.	No
`${catalogName}`	Resolves to the catalog name.	No
`${schemaName}`	Resolves to the schema name.	No
`${tableName}`	Resolves to the short table name.	No
`${opType}`	Resolves to the type of the operation: (`INSERT`, `UPDATE`, `DELETE`, or `TRUNCATE`)	No
`${primaryKeys}`	Resolves to the concatenated primary key values delimited by an underscore (_) character.	No
`${position}`	The sequence number of the source trail file followed by the offset (RBA).	Yes
`${opTimestamp}`	The operation timestamp from the source trail file.	Yes
`${emptyString}`	Resolves to “”.	Yes
`${groupName}`	Resolves to the name of the Replicat process. If using coordinated delivery, it resolves to the name of the Replicat process with the Replicate thread number appended.	Yes

Keyword	Explanation	Transaction Message Support
`${staticMap[]}`	Resolves to a static value where the key is the fully-qualified table name. The keys and values are designated inside of the square brace in the following format: `${staticMap[dbo.table1=value1,dbo.table2=value2]}`	No
`${columnValue[]}`	Resolves to a column value where the key is the fully-qualified table name and the value is the column name to be resolved. For example: `${staticMap[dbo.table1=col1,dbo.table2=col2]}`	No

`${currentTimestamp}`
Or
`${currentTimestamp[]}`
	Resolves to the current timestamp. You can control the format of the current timestamp using the Java based formatting as described in the `SimpleDateFormat` class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
	Examples:
	`${currentDate}`
	`${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}`
		Yes

`${null}`	Resolves to a NULL string.	Yes
`${custom[]}`	It is possible to write a custom value resolver. If required, contact Oracle Support.	Implementation dependent

Example Templates

The following describes example template configuration values and the resolved values.

Example Template	Resolved Value
`${groupName}_${fullyQualfiedTableName}`	`KAFKA001_dbo.table1`
`prefix_${schemaName}_${tableName}_suffix`	`prefix_dbo_table1_suffix`
`${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}`	`2017-05-17 11:45:34.254`

12.2.6 Kafka REST Proxy Handler Formatter Properties

The following are the configurable values for the Kafka REST Proxy Handler Formatter.

Table 12-2 Kafka REST Proxy Handler Formatter Properties

Properties	Optional/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.includeOpType`	Optional	`true | false`	`true`	Set to `true` to create a field in the output messages called `op_ts`. The value is an indicator of the type of source database operation (for example, `I`for insert, `U`for update, `D`for delete). Set to `false` to omit this field in the output.
`gg.handler.name.format.includeOpTimestamp`	Optional	`true | false`	`true`	Set to `true` to create a field in the output messages called `op_type`. The value is the operation timestamp (commit timestamp) from the source trail file. Set to `false` to omit this field in the output.
`gg.handler.name.format.includeCurrentTimestamp`	Optional	`true | false`	`true`	Set to `true` to create a field in the output messages called `current_ts`. The value is the current timestamp of when the handler processes the operation. Set to `false` to omit this field in the output.

Table 12-2 (Cont.) Kafka REST Proxy Handler Formatter Properties

Properties	Optional/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.includePosition`	Optional	`true | false`	`true`	Set to `true` to create a field in the output messages called `pos`. The value is the position (sequence number + offset) of the operation from the source trail file. Set to `false` to omit this field in the output.
`gg.handler.name.format.includePrimaryKeys`	Optional	`true | false`	`true`	Set to `true` to create a field in the output messages called `primary_keys`. The value is an array of the column names of the primary key columns. Set to `false` to omit this field in the output.
`gg.handler.name.format.includeTokens`	Optional	`true | false`	`true`	Set to `true` to include a map field in output messages. The key is `tokens` and the value is a map where the keys and values are the token keys and values from the Oracle GoldenGate source trail file. Set to `false` to suppress this field.
`gg.handler.name.format.insertOpKey`	Optional	Any string.	`I`	The value of the field `op_type` that indicates an insert operation.
`gg.handler.name.format.updateOpKey`	Optional	Any string.	`U`	The value of the field `op_type` that indicates an update operation.

Table 12-2 (Cont.) Kafka REST Proxy Handler Formatter Properties

Properties	Optional/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.deleteOpKey`	Optional	Any string.	`D`	The value of the field `op_type` that indicates an delete operation.
`gg.handler.name.format.truncateOpKey`	Optional	Any string.	`T`	The value of the field `op_type` that indicates an truncate operation.
`gg.handler.name.format.treatAllColumnsAsStrings`	Optional	`true | false`	`false`	Set to `true` treat all output fields as strings. Set to `false` and the handler maps the corresponding field type from the source trail file to the best corresponding Kafka data type.
`gg.handler.name.format.mapLargeNumbersAsStrings`	Optional	`true | false`	`false`	Set to `true` and these fields are mapped as strings to preserve precision. This property is specific to the Avro Formatter; it cannot be used with other formatters.
`gg.handler.name.format.iso8601Format`	Optional	`true | false`	`false`	Set to `true` to output the current date in the ISO8601 format.

Table 12-2 (Cont.) Kafka REST Proxy Handler Formatter Properties

Properties	Optional/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.pkUpdateHandling`	Optional	`abend | update | delete-insert`	`abend`	It is only applicable if you are modeling row messages with the `.(gg.handler.name.format.messageFormatting=row` property. It is not applicable if you are modeling operations messages as the before and after images are propagated to the message with an update.

12.2.7 Setting Metacolumn Output

The following are the configurable values for the Kafka REST Proxy Handler metacolumns template property that controls metacolumn output.

Table 12-3 Metacolumns Template Property

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.metaColumnsTemplate`	Optional	`${alltokens} | ${token} | ${env} | ${sys} | ${javaprop} | ${optype} | ${position} | ${timestamp} | ${catalog} | ${schema} | ${table} | ${objectname} | ${csn} | ${xid} | ${currenttimestamp} | ${opseqno} | ${timestampmicro} | ${currenttimestampmicro}`	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: `insertOpKey | updateOpKey | deleteOpKey | truncateOpKey | includeTableName | includeOpTimestamp | includeOpType | includePosition | includeCurrentTimestamp, useIso8601Format` It is a comma-delimited string consisting of one or more templated values that represent the template.

This is an example that would produce a list of metacolumns:

```
${optype}, ${token.ROWID}, ${sys.username}, ${currenttimestamp}
```

Explanation of the Metacolumn Keywords

`${alltokens}`
All of the Oracle GoldenGate tokens.

`${token}`
The value of a specific Oracle GoldenGate token. The token key should follow token key should follow the token using the period (`.`) operator. For example:

```
${token.MYTOKEN}
```

`${sys}`
A system environmental variable. The variable name should follow `sys` using the period (`.`) operator. For example:

`${sys.MYVAR}`

`${env}`
An Oracle GoldenGate environment variable. The variable name should follow `env` using the period (`.`) operator. For example:

`${env.someVariable}`

`${javaprop}`
A Java JVM variable. The variable name should follow `javaprop` using the period (`.`) operator. For example:

`${javaprop.MYVAR}`

`${optype}`
Operation Type

`${position}`
Record Position

`${timestamp}`
Record Timestamp

`${catalog}`
Catalog Name

`${schema}`
Schema Name

`${table}`
Table Name

`${objectname}`
The fully qualified table name.

`${csn}`
Source Commit Sequence Number

`${xid}`
Source Transaction ID

`${currenttimestamp}`
Current Timestamp

`${opseqno}`
Record sequence number within the transaction.

`${timestampmicro}`
Record timestamp (in microseconds after epoch).

`${currenttimestampmicro}`
Current timestamp (in microseconds after epoch).

Sample Configuration:

```
gg.handlerlist=kafkarestproxy

#The handler properties
gg.handler.kafkarestproxy.type=kafkarestproxy
```

```
gg.handler.kafkarestproxy.mode=tx
#The following selects the topic name based on the fully qualified table name
gg.handler.kafkarestproxy.topicMappingTemplate=${fullyQualifiedTableName}
#The following selects the message key using the concatenated primary keys
gg.handler.kafkarestproxy.keyMappingTemplate=${primaryKeys}
gg.handler.kafkarestproxy.postDataUrl=https://kkm00cfb.in.oracle.com:8082
gg.handler.kafkarestproxy.format=avro
gg.handler.kafkarestproxy.payloadsize=1

gg.handler.kafkarestproxy.socketTimeout=1000
gg.handler.kafkarestproxy.connectTimeout=1000
gg.handler.kafkarestproxy.proxy=host:port
gg.handler.kafkarestproxy.username=username
gg.handler.kafkarestproxy.password=pwd

#The MetaColumnTemplate formatter properties
gg.handler.kafkarestproxy.format.metaColumnsTemplate=${optype},${timestampmicro},$
{currenttimestampmicro}
```

12.3 Consuming the Records

A simple way to consume data from Kafka topics using the Kafka REST Proxy Handler is Curl.

Consume JSON Data

1. Create a consumer for JSON data.

   ```
   curl -k -X POST -H "Content-Type: application/vnd.kafka.v2+json"
   ```

   https://localhost:8082/consumers/my_json_consumer

2. Subscribe to a topic.

   ```
   curl -k -X POST -H "Content-Type: application/vnd.kafka.v2+json"    --data
   '{"topics":["topicname"]}' \
   ```

   https://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/subscription

3. Consume records.

   ```
   curl –k -X GET -H "Accept: application/vnd.kafka.json.v2+json" \
   ```

   https://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/records

Consume Avro Data

1. Create a consumer for Avro data.

   ```
   curl -k -X POST  -H "Content-Type: application/vnd.kafka.v2+json" \
    --data '{"name": "my_consumer_instance", "format": "avro", "auto.offset.reset":
   "earliest"}' \
   ```

   https://localhost:8082/consumers/my_avro_consumer

2. Subscribe to a topic.

   ```
   curl –k -X POST -H "Content-Type: application/vnd.kafka.v2+json"      --data
   '{"topics":["topicname"]}' \
   ```

   https://localhost:8082/consumers/my_avro_consumer/instances/my_consumer_instance/subscription

3. Consume records.

```
curl -X GET -H "Accept: application/vnd.kafka.avro.v2+json" \
https://localhost:8082/consumers/my_avro_consumer/instances/my_consumer_instance/records
```

12.4 Performance Considerations

There are several configuration settings both for the Oracle GoldenGate for Big Data configuration and in the Kafka producer that affects performance.

The Oracle GoldenGate parameter that has the greatest affect on performance is the Replicat `GROUPTRANSOPS` parameter. It allows Replicat to group multiple source transactions into a single target transaction. At transaction commit, the Kafka REST Proxy Handler `POST`'s the data to the Kafka Producer.

Setting the Replicat `GROUPTRANSOPS` to a larger number allows the Replicat to call the `POST` less frequently improving performance. The default value for `GROUPTRANSOPS` is 1000 and performance can be improved by increasing the value to 2500, 5000, or even 10000.

12.5 Kafka REST Proxy Handler Metacolumns Template Property

Problems Starting Kafka REST Proxy server

Sometimes, Flume is set in the classpath, which may stop your REST Proxy server from starting up. Reset the `CLASSPATH` to `“”` to overcome the problem.

Problems with Consuming Records

Your proxy could block the calls while consuming the records from Kafka. Disabling the `http_proxy` variable resolves the issue.

13
Using the Kinesis Streams Handler

Learn how to use the Kinesis Streams Handler, which streams data to applications hosted on the Amazon Cloud or in your environment.

Topics:

- Overview (page 13-1)
- Detailed Functionality (page 13-1)
- Setting Up and Running the Kinesis Streams Handler (page 13-2)
- Kinesis Handler Performance Considerations (page 13-12)
- Troubleshooting (page 13-13)

13.1 Overview

Amazon Kinesis is a messaging system that is hosted in the Amazon Cloud. Kinesis streams can be used to stream data to other Amazon Cloud applications such as Amazon S3 and Amazon Redshift. Using the Kinesis Streams Handler, you can also stream data to applications hosted on the Amazon Cloud or at your site. Amazon Kinesis streams provides functionality similar to Apache Kafka.

The logical concepts map is as follows:

- Kafka Topics = Kinesis Streams
- Kafka Partitions = Kinesis Shards

A Kinesis stream must have at least one shard.

13.2 Detailed Functionality

Topics:

- Amazon Kinesis Java SDK (page 13-1)
- Kinesis Streams Input Limits (page 13-2)

13.2.1 Amazon Kinesis Java SDK

The Oracle GoldenGate Kinesis Streams Handler uses the AWS Kinesis Java SDK to push data to Amazon Kinesis, see *Amazon Kinesis Streams Developer Guide* at:

http://docs.aws.amazon.com/streams/latest/dev/developing-producers-with-sdk.html.

The Kinesis Steams Handler was designed and tested with the latest AWS Kinesis Java SDK version 1.11.107. These are the dependencies:

- Group ID: `com.amazonaws`

- Artifact ID: `aws-java-sdk-kinesis`
- Version: `1.11.107`

Oracle GoldenGate for Big Data does not ship with the AWS Kinesis Java SDK. Oracle recommends that you use the AWS Kinesis Java SDK identified in the Certification Matrix, see Verifying Certification and System Requirements (page 1-1).

> **Note:**
>
> It is assumed by moving to the latest AWS Kinesis Java SDK that there are no changes to the interface, which can break compatibility with the Kinesis Streams Handler.

You can download the AWS Java SDK, including Kinesis from:

https://aws.amazon.com/sdk-for-java/

13.2.2 Kinesis Streams Input Limits

The upper input limit for a Kinesis stream with a single shard is 1000 messages per second up to a total data size of 1MB per second. Adding streams or shards can increase the potential throughput such as the following:

- 1 stream with 2 shards = 2000 messages per second up to a total data size of 2MB per second
- 3 streams of 1 shard each = 3000 messages per second up to a total data size of 3MB per second

The scaling that you can achieve with the Kinesis Streams Handler depends on how you configure the handler. Kinesis stream names are resolved at runtime based on the configuration of the Kinesis Streams Handler.

Shards are selected by the hash the partition key. The partition key for a Kinesis message cannot be null or an empty string (`""`). A null or empty string partition key results in a Kinesis error that results in an abend of the Replicat process.

Maximizing throughput requires that the Kinesis Streams Handler configuration evenly distributes messages across streams and shards.

13.3 Setting Up and Running the Kinesis Streams Handler

Instructions for configuring the Kinesis Streams Handler components and running the handler are described in the following sections.

Use the following steps to set up the Kinesis Streams Handler:

1. Create an Amazon AWS account at https://aws.amazon.com/.
2. Log into Amazon AWS.
3. From the main page, select **Kinesis** (under the Analytics subsection).
4. Select Amazon Kinesis Streams **Go to Streams** to create Amazon Kinesis streams and shards within streams.
5. Create a client ID and secret to access Kinesis.

The Kinesis Streams Handler requires these credentials at runtime to successfully connect to Kinesis.

6. Create the client ID and secret:
   a. Select your name in AWS (upper right), and then in the list select **My Security Credentials**.
   b. Select **Access Keys** to create and manage access keys.

      Note your client ID and secret upon creation.

      The client ID and secret can only be accessed upon creation. If lost, you have to delete the access key, and then recreate it.

Topics:

- Set the Classpath in Kinesis Streams Handler (page 13-3)
- Kinesis Streams Handler Configuration (page 13-3)
- Using Templates to Resolve the Stream Name and Partition Name (page 13-8)
- Configuring the Client ID and Secret in Kinesis Handler (page 13-10)
- Configuring the Proxy Server for Kinesis Streams Handler (page 13-11)
- Configuring Security in Kinesis Streams Handler (page 13-11)

13.3.1 Set the Classpath in Kinesis Streams Handler

You must configure the `gg.classpath` property in the Java Adapter properties file to specify the JARs for the AWS Kinesis Java SDK as follows:

```
gg.classpath={download_dir}/aws-java-sdk-1.11.107/lib/*:{download_dir}/aws-java-sdk-1.11.107/third-party/lib/*
```

13.3.2 Kinesis Streams Handler Configuration

You configure the Kinesis Streams Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Kinesis Streams Handler, you must first configure the handler type by specifying `gg.handler.name.type=kinesis_streams` and the other Kinesis Streams properties as follows:

Table 13-1 Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.type`	Required	`kinesis_streams`	None	Selects the Kinesis Streams Handler for streaming change data capture into Kinesis.

Table 13-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.region`	Required	The Amazon region name which is hosting your Kinesis instance.	None	Setting of the Amazon AWS region name is required.
`gg.handler.name.proxyServer`	Optional	The host name of the proxy server.	None	Set the host name of the proxy server if connectivity to AWS is required to go through a proxy server.
`gg.handler.name.proxyPort`	Optional	The port number of the proxy server.	None	Set the port name of the proxy server if connectivity to AWS is required to go through a proxy server.
`gg.handler.name.proxyUsername`	Optional	The username of the proxy server (if credentials are required).	None	Set the username of the proxy server if connectivity to AWS is required to go through a proxy server and the proxy server requires credentials.
`gg.handler.name.proxyPassword`	Optional	The password of the proxy server (if credentials are required).	None	Set the password of the proxy server if connectivity to AWS is required to go through a proxy server and the proxy server requires credentials.

Table 13-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.deferFlushAtTxCommit`	Optional	`true` | `false`	`false`	When set to false, the Kinesis Streams Handler will flush data to Kinesis at transaction commit for write durability. However, it may be preferable to defer the flush beyond the transaction commit for performance purposes, see Kinesis Handler Performance Considerations (page 13-12).
`gg.handler.name.deferFlushOpCount`	Optional	Integer	None	Only applicable if `gg.handler.name.deferFlushAtTxCommit` is set to `true`. This parameter marks the minimum number of operations that must be received before triggering a flush to Kinesis. Once this number of operations are received, a flush will occur on the next transaction commit and all outstanding operations will be moved from the Kinesis Streams Handler to AWS Kinesis.

Table 13-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.formatPerOp`	Optional	`true` | `false`	`true`	When set to `true`, it will send messages to Kinesis, once per operation (insert, delete, update). When set to `false`, operations messages will be concatenated for all the operations and a single message will be sent at the transaction level. Kinesis has a limitation of 1MB max massage size. If 1MB is exceeded then transaction level message will be broken up into multiple messages.

Table 13-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.customMessageGrouper`	Optional	`oracle.goldengate.handler.kinesis.KinesisJsonTxMessageGrouper`	None	This configuration parameter provides the ability to group Kinesis messages using custom logic. Only one implementation is included in the distribution at this time. The `oracle.goldengate.handler.kinesis.KinesisJsonTxMessageGrouper` is a custom message which groups JSON operation messages representing operations into a wrapper JSON message that encompasses the transaction. Setting of this value overrides the setting of the `gg.handler.formatPerOp` setting. Using this feature assumes that the customer is using the JSON formatter (that is `gg.handler.name.format=json`).
`gg.handler.name.streamMappingTemplate`	Required	A template string value to resolve the Kinesis message partition key (message key) at runtime.	None	See Using Templates to Resolve the Stream Name and Partition Name (page 13-8) for more information.

Table 13-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.partitionMappingTemplate`	Required	A template string value to resolve the Kinesis message partition key (message key) at runtime.	None	See Using Templates to Resolve the Stream Name and Partition Name (page 13-8) for more information.
`gg.hander.name.format`	Required	Any supported pluggable formatter.	`delimitedtext` | `json` | `json_row` | `xml` | `avro_row` | `avro_opt`	Selects the operations message formatter. JSON is likely the best fit for Kinesis.
`gg.hander.name.enableStreamCreation`	Optional	`true`	`true` | `false`	By default, the Kinesis Handler automatically creates Kinesis streams if they do not already exist. Set to `false` to disable to automatic creation of Kinesis streams.
`gg.hander.name.shardCount`	Optional	Positive integer.	`1`	A Kinesis stream contains 1 or more shards. Controls the number of shards on Kinesis streams that the Kinesis Handler creates. Multiple shards can help improve the ingest performance to a Kinesis stream. Use only when `gg.handler.name.enableStreamCreation` is set to `true`.

13.3.3 Using Templates to Resolve the Stream Name and Partition Name

The Kinesis Streams Handler provides the functionality to resolve the stream name and the partition key at runtime using a template configuration value. Templates allow

you to configure static values and keywords. Keywords are used to dynamically replace the keyword with the context of the current processing. Templates are applicable to the following configuration parameters:

```
gg.handler.name.streamMappingTemplate
gg.handler.name.partitionMappingTemplate
```

Template Modes

Source database transactions are made up of 1 or more individual operations which are the individual inserts, updates, and deletes. The Kinesis Handler can be configured to send one message per operation (insert, update, delete, Alternatively, it can be configured to group operations into messages at the transaction level. Many of the template keywords resolve data based on the context of an individual source database operation. Therefore, many of the keywords *do not work* when sending messages at the transaction level. For example `${fullyQualifiedTableName}` does not work when sending messages at the transaction level. The `${fullyQualifiedTableName}` property resolves to the qualified source table name for an operation. Transactions can contain multiple operations for many source tables. Resolving the fully-qualified table name for messages at the transaction level is non-deterministic and so abends at runtime.

Template Keywords

The following table lists the currently supported keyword templates and includes a column if the keyword is supported for transaction level messages:

Keyword	Explanation	Transaction Message Support
`${fullyQualifiedTableName}`	Resolves to the fully qualified table name including the period (`.`) Delimiter between the catalog, schema, and table names. For example, `test.dbo.table1`.	No
`${catalogName}`	Resolves to the catalog name.	No
`${schemaName}`	Resolves to the schema name	No
`${tableName}`	Resolves to the short table name.	No
`${opType}`	Resolves to the type of the operation: (INSERT, UPDATE, DELETE, or TRUNCATE)	No
`${primaryKeys}`	Resolves to the concatenated primary key values delimited by an underscore (_) character.	No
`${position}`	The sequence number of the source trail file followed by the offset (RBA).	Yes
`${opTimestamp}`	The operation timestamp from the source trail file.	Yes
`${emptyString}`	Resolves to “”.	Yes
`${groupName}`	Resolves to the name of the replicat process. If using coordinated delivery it resolves to the name of the Replicat process with the replicate thread number appended.	Yes

Keyword	Explanation	Transaction Message Support
`${staticMap[]}`	Resolves to a static value where the key is the fully qualified table name. The keys and values are designated inside of the square brace in the following format: `${staticMap[dbo.table1=value1,dbo.table2=value2]}`	No
`${columnValue[]}`	Resolves to a column value where the key is the fully qualified table name and the value is the column name to be resolved. For example: `${staticMap[dbo.table1=col1,dbo.table2=col2]}`	No

`${currentTimestamp}`
Or
`${currentTimestamp[]}`
	Resolves to the current timestamp. You can control the format of the current timestamp using the Java based formatting as described in the `SimpleDateFormat` class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html.
	Examples:
	`${currentDate}`
	`${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}`
		Yes

`${null}`	Resolves to a null string.	Yes
`${custom[]}`	It is possible to write a custom value resolver.	Depends on the implementation.

Example Templates

The following describes example template configuration values and the resolved values.

Example Template	Resolved Value
`${groupName}_{fullyQualifiedTableName}`	`KINESIS001_dbo.table1`
`prefix_${schemaName}_${tableName}_suffix`	`prefix_dbo_table1_suffix`
`${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}`	`2017-05-17 11:45:34.254`

13.3.4 Configuring the Client ID and Secret in Kinesis Handler

A client ID and secret are required credentials for the Kinesis Streams Handler to interact with Amazon Kinesis. A client ID and secret are generated through the Amazon AWS website. The retrieval of these credentials and presentation to the Kinesis server are performed on the client side by the AWS Kinesis Java SDK. The AWS Kinesis Java SDK provides multiple ways that the client ID and secret can be resolved at runtime.

The client ID and secret can be set

- as Java properties so configured in the Java Adapter properties file as follows:

```
javawriter.bootoptions=-Xmx512m -Xms32m
-Djava.class.path=ggjava/ggjava.jar
```

```
-Daws.accessKeyId=your_access_key
-Daws.secretKey=your_secret_key
```

- as environmental variables using the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` variables.
- in the E2C environment on the local machine.

13.3.5 Configuring the Proxy Server for Kinesis Streams Handler

Oracle GoldenGate can be used with a proxy server using the following parameters to enable the proxy server:

- `gg.handler.name.proxyServer=`
- `gg.handler.name.proxyPort=80`

Access to the proxy servers can be secured using credentials and the following configuration parameters:

- `gg.handler.name.proxyUsername=username`
- `gg.handler.name.proxyPassword=password`

Sample configurations:

```
gg.handlerlist=kinesis
gg.handler.kinesis.type=kinesis_streams
gg.handler.kinesis.mode=op
gg.handler.kinesis.format=json
gg.handler.kinesis.region=us-west-2
gg.handler.kinesis.partitionMappingTemplate=TestPartitionName
gg.handler.kinesis.streamMappingTemplate=TestStream
gg.handler.kinesis.deferFlushAtTxCommit=true
gg.handler.kinesis.deferFlushOpCount=1000
gg.handler.kinesis.formatPerOp=true
#gg.handler.kinesis.customMessageGrouper=oracle.goldengate.handler.kinesis.KinesisJso
nTxMessageGrouper
gg.handler.kinesis.proxyServer=www-proxy.myhost.com
gg.handler.kinesis.proxyPort=80
```

13.3.6 Configuring Security in Kinesis Streams Handler

The AWS Kinesis Java SDK uses HTTPS to communicate with Kinesis. The Kinesis Streams Handler is authenticated by presenting the client ID and secret credentials at runtime using a trusted certificate.

The Kinesis Streams Handler can also be configured to authenticate the server providing mutual authentication. You can do this by generating a certificate from the Amazon AWS website and configuring server authentication. A trust store must be generated on the machine hosting Oracle GoldenGate for Big Data. The trust store and trust store password must be configured in the Kinesis Streams Handler Java Adapter properties file.

The following is an example configuration:

```
javawriter.bootoptions=-Xmx512m -Xms32m
-Djava.class.path=ggjava/ggjava.jar
–Djavax.net.ssl.trustStore=path_to_trust_store_file
–Djavax.net.ssl.trustStorePassword=trust_store_password
```

13.4 Kinesis Handler Performance Considerations

Topics:

- Kinesis Streams Input Limitations (page 13-12)
- Transaction Batching (page 13-12)
- Deferring Flush at Transaction Commit (page 13-13)

13.4.1 Kinesis Streams Input Limitations

The maximum write rate to a Kinesis stream with a single shard to be 1000 messages per second up to a maximum of 1MB of data per second. You can scale input to Kinesis by adding additional Kinesis streams or adding shards to streams. Both adding streams and adding shards can linearly increase the Kinesis input capacity and thereby improve performance of the Oracle GoldenGate Kinesis Streams Handler.

Adding streams or shards can linearly increase the potential throughput such as follows:

- 1 stream with 2 shards = 2000 messages per second up to a total data size of 2MB per second.
- 3 streams of 1 shard each = 3000 messages per second up to a total data size of 3MB per second.

To fully take advantage of streams and shards, you must configure the Oracle GoldenGate Kinesis Streams Handler to distribute messages as evenly as possible across streams and shards.

Adding additional Kinesis streams or shards does nothing to scale Kinesis input if all data is sent to using a static partition key into a single Kinesis stream. Kinesis streams are resolved at runtime using the selected mapping methodology. For example, mapping the source table name as the Kinesis stream name may provide good distribution of messages across Kinesis streams if operations from the source trail file are evenly distributed across tables. Shards are selected by a hash of the partition key. Partition keys are resolved at runtime using the selected mapping methodology. Therefore, it is best to choose a mapping methodology to a partition key that rapidly changes to ensure a good distribution of messages across shards.

13.4.2 Transaction Batching

The Oracle GoldenGate Kinesis Streams Handler receives messages and then batches together messages by Kinesis stream before sending them via synchronous HTTPS calls to Kinesis. At transaction commit all outstanding messages are flushed to Kinesis. The flush call to Kinesis impacts performance. Therefore, deferring the flush call can dramatically improve performance.

The recommended way to defer the flush call is to use the `GROUPTRANSOPS` configuration in the replicat configuration. The `GROUPTRANSOPS` groups multiple small transactions into a single larger transaction deferring the transaction commit call until the larger transaction is completed. The `GROUPTRANSOPS` parameter works by counting the database operations (inserts, updates, and deletes) and only commits the transaction group when the number of operations equals or exceeds the `GROUPTRANSOPS` configuration setting. The default `GROUPTRANSOPS` setting for replicat is 1000.

Interim flushes to Kinesis may be required with the `GROUPTRANSOPS` setting set to a large amount. An individual call to send batch messages for a Kinesis stream cannot exceed 500 individual messages or 5MB. If the count of pending messages exceeds 500 messages or 5MB on a per stream basis then the Kinesis Handler is required to perform an interim flush.

13.4.3 Deferring Flush at Transaction Commit

The messages are by default flushed to Kinesis at transaction commit to ensure write durability. However, it is possible to defer the flush beyond transaction commit. This is only advisable when messages are being grouped and sent to Kinesis at the transaction level (that is one transaction = one Kinesis message or chunked into a small number of Kinesis messages), when the user is trying to capture the transaction as a single messaging unit.

This may require setting the `GROUPTRANSOPS` replication parameter to 1 so as not to group multiple smaller transactions from the source trail file into a larger output transaction. This can impact performance as only one or few messages are sent per transaction and then the transaction commit call is invoked which in turn triggers the flush call to Kinesis.

In order to maintain good performance the Oracle GoldenGate Kinesis Streams Handler allows the user to defer the Kinesis flush call beyond the transaction commit call. The Oracle GoldenGate replicat process maintains the checkpoint in the `.cpr` file in the `{GoldenGate Home}/dirchk` directory. The Java Adapter also maintains a checkpoint file in this directory named `.cpj`. The Replicat checkpoint is moved beyond the checkpoint for which the Oracle GoldenGate Kinesis Handler can guarantee message loss will not occur. However, in this mode of operation the GoldenGate Kinesis Streams Handler maintains the correct checkpoint in the `.cpj` file. Running in this mode will not result in message loss even with a crash as on restart the checkpoint in the `.cpj` file is parsed if it is before the checkpoint in the `.cpr` file.

13.5 Troubleshooting

Topics:

- Java Classpath (page 13-13)
- Kinesis Handler Connectivity Issues (page 13-14)
- Logging (page 13-14)

13.5.1 Java Classpath

The most common initial error is an incorrect classpath to include all the required AWS Kinesis Java SDK client libraries and creates a `ClassNotFound` exception in the log file.

You can troubleshoot by setting the Java Adapter logging to `DEBUG`, and then rerun the process. At the debug level, the logging includes information about which JARs were added to the classpath from the `gg.classpath` configuration variable.

The `gg.classpath` variable supports the wildcard asterisk (`*`) character to select all JARs in a configured directory. For example, `/usr/kinesis/sdk/*`, see Setting Up and Running the Kinesis Streams Handler (page 13-2).

13.5.2 Kinesis Handler Connectivity Issues

If the Kinesis Streams Handler is unable to connect to Kinesis when running on premise, the problem can be the connectivity to the public Internet is protected by a proxy server. Proxy servers act a gateway between the private network of a company and the public Internet. Contact your network administrator to get the URLs of your proxy server, and then follow the directions in Configuring the Proxy Server for Kinesis Streams Handler (page 13-11).

13.5.3 Logging

The Kinesis Streams Handler logs the state of its configuration to the Java log file.

This is helpful because you can review the configuration values for the handler. Following is a sample of the logging of the state of the configuration:

```
**** Begin Kinesis Streams Handler - Configuration Summary ****
Mode of operation is set to op.
   The AWS region name is set to [us-west-2].
   A proxy server has been set to [www-proxy.us.oracle.com] using port [80].
   The Kinesis Streams Handler will flush to Kinesis at transaction commit.
    Messages from the GoldenGate source trail file will be sent at the operation
level.
   One operation = One Kinesis Message
The stream mapping template of [${fullyQualifiedTableName}] resolves to [fully
qualified table name].
 The partition mapping template of [${primaryKeys}] resolves to [primary keys].
**** End Kinesis Streams Handler - Configuration Summary ****
```

14
Using the MongoDB Handler

Learn how to use the MongoDB Handler, which can replicate transactional data from Oracle GoldenGate to a target MongoDB database.

Topics:

- Overview (page 14-1)
- Detailed Functionality (page 14-1)
- Setting Up and Running the MongoDB Handler (page 14-2)
- Review a Sample Configuration (page 14-7)

14.1 Overview

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling, see https://www.mongodb.com/.

14.2 Detailed Functionality

The MongoDB Handler takes operations from the source trail file and creates corresponding documents in the target MongoDB database.

A record in MongoDB is a Binary JSON (BSON) document, which is a data structure composed of field and value pairs. A BSON data structure is a binary representation of JSON documents. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

A collection is a grouping of MongoDB documents and is the equivalent of an RDBMS table. In MongoDB, databases hold collections of documents. Collections do not enforce a schema. MongoDB documents within a collection can have different fields.

Topics:

- Document Key Column (page 14-1)
- Primary Key Update Operation (page 14-2)
- MongoDB Trail Data Types (page 14-2)

14.2.1 Document Key Column

MongoDB databases require every document (row) to have a column named `_id` whose value should be unique in a collection (table). This is similar to a primary key for RDBMS tables. If a document does not contain a top-level `_id` column during an insert, the MongoDB driver adds this column.

The MongoDB Handler builds custom `_id` field values for every document based on the primary key column values in the trail record. This custom `_id` is built using all the key column values concatenated by a `:` (colon) separator. For example:

`KeyColValue1:KeyColValue2:KeyColValue3`
The MongoDB Handler enforces uniqueness based on these custom `_id` values. This means that every record in the trail must be unique based on the primary key columns values. Existence of non-unique records for the same table results in a MongoDB Handler failure and in Replicat abending with a duplicate key error.

The behavior of the `_id` field is:

- By default, MongoDB creates a unique index on the column during the creation of a collection.
- It is always the first column in a document.
- It may contain values of any BSON data type except an array.

14.2.2 Primary Key Update Operation

MongoDB databases do not allow the `_id` column to be modified. This means a primary key update operation record in the trail needs special handling. The MongoDB Handler converts a primary key update operation into a combination of a `DELETE` (with old key) and an `INSERT` (with new key). To perform the `INSERT`, a complete before-image of the update operation in trail is recommended. You can generate the trail to populate a complete before image for update operations by enabling the Oracle GoldenGate `GETUPDATEBEFORES` and `NOCOMPRESSUPDATES` parameters, see *Reference for Oracle GoldenGate*.

14.2.3 MongoDB Trail Data Types

The MongoDB Handler supports delivery to the BSON data types as follows:

- 32-bit integer
- 64-bit integer
- Double
- Date
- String
- Binary data

14.3 Setting Up and Running the MongoDB Handler

The following sections provide instructions for configuring the MongoDB Handler components and running the handler.

Topics:

- Classpath Configuration (page 14-3)
- MongoDB Handler Configuration (page 14-3)
- Connecting and Authenticating (page 14-5)
- Using Bulk Write (page 14-6)
- Using Write Concern (page 14-6)
- Using Three-Part Table Names (page 14-6)

- Using Undo Handling (page 14-7)

14.3.1 Classpath Configuration

The MongoDB Java Driver is required for Oracle GoldenGate for Big Data to connect and stream data to MongoDB. The minimum required version of the MongoDB Java Driver is 3.4.3. The MongoDB Java Driver is not included in the Oracle GoldenGate for Big Data product. You must download the driver from:

https://docs.mongodb.com/ecosystem/drivers/java/#download-upgrade

Select **mongo-java-driver** and the **3.4.3** version to download the recommended driver JAR file.

You must configure the `gg.classpath` variable to load the MongoDB Java Driver JAR at runtime. For example: `gg.classpath=/home/mongodb/mongo-java-driver-3.4.3.jar`

Oracle GoldenGate for Big Data supports the MongoDB Decimal 128 data type that was added in MongoDB 3.4. Use of a MongoDB Java Driver prior to 3.4.3 results in a `ClassNotFound` exception. You can disable the use of the Decimal128 data type to support MongoDB server versions older than 3.4 by setting this configuration parameter:

```
gg.handler.name.enableDecimal128=false
```

14.3.2 MongoDB Handler Configuration

You configure the MongoDB Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the MongoDB Handler, you must first configure the handler type by specifying `gg.handler.name.type=mongodb` and the other MongoDB properties as follows:

Table 14-1 MongoDB Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.type`	Required	`mongodb`	None	Selects the MongoDB Handler for use with Replicat.
`gg.handler.`*`name`*`.bulkWrite`	Optional	`true` | `false`	`true`	Set to `true`, the handler caches operations until a commit transaction event is received. When committing the transaction event, all the cached operations are written out to the target MongoDB database, which provides improved throughput. Set to `false`, there is no caching within the handler and operations are immediately written to the MongoDB database.

Table 14-1 (Cont.) MongoDB Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.WriteConcern`	Optional	`{"w": "value" , "wtimeout": "number" }`	None	Sets the required write concern for all the operations performed by the MongoDB Handler. The property value is in JSON format and can only accept keys as `w` and `wtimeout`, see https://docs.name.com/manual/reference/write-concern/.
`gg.handler.name.username`	Optional	A legal username string.	None	Sets the authentication username to be used. Use with the `AuthenticationMechanism` property.
`gg.handler.name.password`	Optional	A legal password string.	None	Sets the authentication password to be used. Use with the `AuthenticationMechanism` property.
`gg.handler.name.ServerAddressList`	Optional	`IP:PORT` with multiple port values delimited by a comma	None	Enables the connection to a list of Replicat set members or a list of MongoDB databases. This property accepts a comma separated list of `[hostnames:port]`. For example, `localhost1:27017,localhost2:27018,localhost3:27019`, see http://api.name.com/java/3.0/com/mongodb/MongoClient.html#MongoClient-java.util.List-java.util.List-com.name.MongoClientOptions-.
`gg.handler.name.AuthenticationMechanism`	Optional	Comma separated list of authentication mechanism	None	Sets the authentication mechanism which is a process of verifying the identity of a client. The input would be a comma separated list of various authentication options. For example, `GSSAPI,MONGODB_CR,MONGODB_X509,PLAIN,SCRAM_SHA_1`. see http://api.name.com/java/3.0/com/mongodb/MongoCredential.html.
`gg.handler.name.source`	Optional	Valid authentication source	None	Sets the source of the user name, typically the name of the database where the user is defined. Use with the `AuthenticationMechanism` property.
`gg.handler.name.clientURI`	Optional	Valid MongoDB client URI	None	Sets the MongoDB client URI. A client URI can also be used to set other MongoDB connection properties, such as authentication and `WriteConcern`. For example, `mongodb://localhost:27017/`, see: http://api.name.com/java/3.0/com/mongodb/MongoClientURI.html.

Table 14-1 (Cont.) MongoDB Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.Host`	Optional	Valid MongoDB server name or IP address	None	Sets the MongoDB database hostname to connect to based on a (single) MongoDB node, see http://api.name.com/java/3.0/com/mongodb/MongoClient.html#MongoClient-java.lang.String-.
`gg.handler.name.Port`	Optional	Valid MongoDB port	None	Sets the MongoDB database instance port number. Use with the `Host` property.
`gg.handler.name.CheckMaxRowSizeLimit`	Optional	`true` | `false`	`false`	When set to `true`, the handler verifies that the size of the BSON document inserted or modified is within the limits defined by the MongoDB database. Calculating the size involves the use of a default codec to generate a `RawBsonDocument`, leading to a small degradation in the throughput of the MongoDB Handler. If the size of the document exceeds the MongoDB limit, an exception occurs and Replicat abends.
`gg.handler.name.upsert`	Optional	`true` | `false`	`false`	When set to `true`, a new Mongo document is inserted if there are no matches to the query filter when performing an `UPDATE` operation.
`gg.handler.name.enableDecimal128`	Optional	`true` | `false`	`true`	MongoDB version 3.4 added support for a 128 bit decimal data type called Decimal128. This data type was needed since supports both integer and decimal data types that do not fit into a 64 bit Long or Double. Setting this property to `true` enables mapping into the Double128 data type for source data types that require it. Set to `false` to process these source data types as 64 bit Doubles.

14.3.3 Connecting and Authenticating

In the handler properties file, you can configure various connection and authentication properties. When multiple connection properties are specified, the MongoDB Handler chooses the properties according to the following priority:

Priority 1:

```
ServerAddressList
AuthentictionMechanism
UserName
Password
Source
Write Concern
```

Priority 2:

```
ServerAddressList
AuthentictionMechanism
UserName
Password
Source
```

Priority 3:

```
clientURI
```

Priority 4:

```
Host
Port
```

Priority 5:

```
Host
```

If none of the connection and authentication properties are specified, the handler tries to connect to `localhost` on port `27017`.

14.3.4 Using Bulk Write

Bulk write is enabled by default. For better throughput, Oracle recommends that you use bulk write.

You can also enable bulk write by using the `BulkWrite` handler property. To enable or disable bulk write use the `gg.handler.`*`handler`*`.BulkWrite=`*`true`* `|` *`false`*. The MongoDB Handler does *not* use the `gg.handler.`*`handler`*`.mode=op | tx` property that is used by Oracle GoldenGate for Big Data.

With bulk write, the MongoDB Handler uses the `GROUPTRANSOPS` parameter to retrieve the batch size. The handler converts a batch of trail records to MongoDB documents, which are then written to the database in one request.

14.3.5 Using Write Concern

Write concern describes the level of acknowledgement that is requested from MongoDB for write operations to a standalone MongoDB, replica sets, and sharded-clusters. With sharded-clusters, Mongo instances pass the write concern on to the shards, see https://docs.mongodb.com/manual/reference/write-concern/.

Use the following configuration:

```
w: value
wtimeout: number
```

14.3.6 Using Three-Part Table Names

An Oracle GoldenGate trail may have data for sources that support three-part table names, such as *`Catalog.Schema.Table`*. MongoDB only supports two-part names, such as *`DBName.Collection`*. To support the mapping of source three-part names to MongoDB two-part names, the source *`Catalog`* and *`Schema`* is concatenated with an underscore delimiter to construct the Mongo *`DBName`*.

For example, *`Catalog.Schema.Table`* would become *`catalog1_schema1.table1`*.

14.3.7 Using Undo Handling

The MongoDB Handler can recover from bulk write errors using a lightweight undo engine. This engine works differently from typical RDBMS undo engines, rather the best effort to assist you in error recovery. Error recovery works well when there are primary violations or any other bulk write error where the MongoDB database provides information about the point of failure through `BulkWriteException`.

Table 14-2 (page 14-7)Table 1 lists the requirements to make the best use of this functionality.

Table 14-2 Undo Handling Requirements

Operation to Undo	Require Full Before Image in the Trail?
`INSERT`	No
`DELETE`	Yes
`UPDATE`	No (before image of fields in the `SET` clause.)

If there are errors during undo operations, it may be not possible to get the MongoDB collections to a consistent state. In this case, you must manually reconcile the data.

14.4 Review a Sample Configuration

The following is a sample configuration for the MongoDB Handler from the Java adapter properties file:

```
gg.handlerlist=mongodb
gg.handler.mongodb.type=mongodb

#The following handler properties are optional.
#Please refer to the Oracle GoldenGate for BigData documentation
#for details about the configuration.
#gg.handler.mongodb.clientURI=mongodb://localhost:27017/
#gg.handler.mongodb.Host=<MongoDBServer address>
#gg.handler.mongodb.Port=<MongoDBServer port>
#gg.handler.mongodb.WriteConcern={ w: <value>, wtimeout: <number> }
#gg.handler.mongodb.AuthenticationMechanism=GSSAPI,MONGODB_CR,MONGODB_X509,PLAIN,SCRA
M_SHA_1
#gg.handler.mongodb.UserName=<Authentication username>
#gg.handler.mongodb.Password=<Authentication password>
#gg.handler.mongodb.Source=<Authentication source>
#gg.handler.mongodb.ServerAddressList=localhost1:27017,localhost2:27018,localhost3:27
019,...
#gg.handler.mongodb.BulkWrite=<false|true>
#gg.handler.mongodb.CheckMaxRowSizeLimit=<true|false>

goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
```

```
#Path to MongoDB Java driver.
# maven co-ordinates
# <dependency>
# <groupId>org.mongodb</groupId>
# <artifactId>mongo-java-driver</artifactId>
# <version>3.2.2</version>
# </dependency>
gg.classpath=/path/to/mongodb/java/driver/mongo-java-driver-3.2.2.jar
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./
dirprm
```

15
Using the Metadata Providers

Learn how to use the Metadata Providers, which can replicate from a source to a target using a Replicat parameter file.

Topics:

- About the Metadata Providers (page 15-1)
- Avro Metadata Provider (page 15-2)
- Java Database Connectivity Metadata Provider (page 15-7)
- Hive Metadata Provider (page 15-10)

15.1 About the Metadata Providers

Metadata Providers work only if handlers are configured to run with a Replicat process.

The Replicat process maps source table to target table and source column to target column mapping using syntax in the Replicat configuration file. The source metadata definitions are included in the Oracle GoldenGate trail file (or by source definitions files in Oracle GoldenGate releases 12.2 and later). 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 during Java delivery in general. Typically, Big Data applications provide no target metadata, so Replicat mapping is not possible. The metadata providers exist to address this deficiency. You can use a metadata provider to define target metadata using either Avro or Hive, which enables Replicat mapping of source table to target table and source column to target column.

The use of the metadata provider is optional and is enabled if the `gg.mdp.type` property is specified in the Java Adapter Properties file. If the metadata included in the source Oracle GoldenGate trail file is acceptable for output, then do not use the metadata provider. Use a metadata provider should be used in the following cases:

- You need to map source table names into target table names that do not match.
- You need to map source column names into target column name that do not match.
- You need to include certain columns from the source trail file and omit other columns.

A limitation of Replicat mapping is that the mapping defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 and later provides functionality for DDL propagation when using an Oracle database as the source. The proper handling of schema evolution can be problematic when the Metadata Provider and Replicat mapping are used. Consider your use cases for schema evolution and plan for how you want to update the Metadata Provider and the Replicat mapping syntax for required changes.

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

Only the Hive and Avro Metadata Providers are supported and you must choose one or the other to use in your metadata provider implementation.

Scenarios - When to use a metadata provider

1. The following scenarios do *not* require a metadata provider to be configured:

   A mapping in which the source schema named `GG` is mapped to the target schema named `GGADP`.*

   A mapping in which the 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 a metadata provider to be configured:

   A mapping in which the source column name does not match the target column name. For example, a source column of `CUST_CODE` mapped to a target column of `CUST_CODE_NEW`.

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

15.2 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 is retrieved from Avro Schema. Retrieved metadata is then used by Replicat for column mapping.

This section contains the following:

Topics:

- Detailed Functionality (page 15-2)
- Runtime Prerequisites (page 15-3)
- Classpath Configuration (page 15-4)
- Avro Metadata Provider Configuration (page 15-4)
- Review a Sample Configuration (page 15-4)
- Metadata Change Events (page 15-5)
- Limitations (page 15-6)
- Troubleshooting (page 15-6)

15.2.1 Detailed Functionality

The Avro Metadata Provider uses Avro schema definition files to retrieve metadata. Avro schemas are defined using JSON. For each table mapped in the *`process_name.prm`* file, you must create a corresponding Avro schema definition file. For information about how to define Avro schemas, see 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 names of schema files that are 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

See https://avro.apache.org/docs/1.7.5/spec.html#schema_primitive.

15.2.2 Runtime Prerequisites

Before you start the Replicat process, create Avro schema definitions for all tables mapped in Replicat's parameter file.

15.2.3 Classpath Configuration

The Avro Metadata Provider requires no additional classpath setting.

15.2.4 Avro Metadata Provider Configuration

Property	Required/ Optional	Legal Values	Default	Explanation
`gg.mdp.type`	Required	`avro`	-	Selects the Avro Metadata Provider
`gg.mdp.schemaFilesPath`	Required	Example:`/home/user/ggadp/avroschema/`	-	The path to the Avro schema files directory
`gg.mdp.charset`	Optional	Valid character set	`UTF-8`	Specifies the character set of the column with character data type. Used to convert the source data from the trail file to the correct target character set.
`gg.mdp.nationalCharset`	Optional	Valid character set	`UTF-8`	Specifies the character set of the column with character data type. Used to convert the source data from the trail file to the correct target character set. Example: Used to indicate character set of columns, such as `NCHAR`, `NVARCHAR` in an Oracle database.

15.2.5 Review a Sample Configuration

This is an example for configuring the Avro Metadata Provider. Consider a source that includes the following table:

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

This table maps the`(CUST_CODE (GG.TCUSTMER)` in the source to `CUST_CODE2 (GG_AVRO.TCUSTMER_AVRO)` on the target and the column `CITY (GG.TCUSTMER)` in source to `CITY2 (GG_AVRO.TCUSTMER_AVRO)` on the target. Therefore, the mapping in the *`process_name.`* `prm` file is:

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

In this example the mapping definition is as follows:

- Source schema `GG` is mapped to target schema `GG_AVRO`.
- 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).

This example uses the following Avro schema definition file:

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 includes the following:

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

The following sample output uses a delimited text formatter with a semi-colon as the delimiter:

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

Oracle GoldenGate for Big Data includes a sample Replicat configuration file, a sample Java Adapter properties file, and sample Avro schemas at the following location:

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

15.2.6 Metadata Change Events

If the DDL changes in the source database tables, you may need to modify the Avro schema definitions and the mappings in the Replicat configuration file. You may also want to stop or suspend the Replicat process in the case of a metadata change event. You can stop the Replicat process by adding the following line to the Replicat configuration file (*process_name.* `prm`):

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

Alternatively, you can suspend the Replicat process by adding the following line to the Replication configuration file:

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

15.2.7 Limitations

Avro bytes data type cannot be used as primary key.

The source-to-target mapping that is defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 and later support DDL propagation and source schema evolution for Oracle Databases as replication source. If you use DDL propagation and source schema evolution, you lose the ability to seamlessly handle changes to the source metadata.

15.2.8 Troubleshooting

This section contains the information about how to troubleshoot the following issues:

Topics:

- Invalid Schema Files Location (page 15-6)
- Invalid Schema File Name (page 15-6)
- Invalid Namespace in Schema File (page 15-7)
- Invalid Table Name in Schema File (page 15-7)

15.2.8.1 Invalid Schema Files Location

The Avro schema files directory specified in the `gg.mdp.schemaFilesPath` configuration property must be a valid directory.If the path is not valid, you encounter following exception:

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

15.2.8.2 Invalid Schema File Name

For every table that is mapped in the *`process_name`*`.prm` file, you must create a corresponding Avro schema file in the directory that is 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/
```

In this scenario, you must create a file called `GG_AVRO.TCUSTMER_AVRO.mdp.avsc` in the `/home/usr/avro/` directory.

If you do not create the `/home/usr/avro/GG_AVRO.TCUSTMER_AVRO.mdp.avsc` file, you encounter the following exception:

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

15.2.8.3 Invalid Namespace in Schema File

The target schema name specified in REPLICAT mapping must be same as the 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:

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

15.2.8.4 Invalid Table Name in Schema File

The target table name that is specified in Replicat mapping must be same as the 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, if the target table name specified in Replicat mapping does not match with the Avro schema name, then REPLICAT abends with following exception:

```
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}
```

15.3 Java Database Connectivity Metadata Provider

The Java Database Connectivity (JDBC) Metadata Provider is used to retrieve the table metadata from any target database that supports a JDBC connection and has a database schema. The JDBC Metadata Provider is the preferred metadata provider for

any target database that is an RDBMS, although various other non-RDBMS targets also provide a JDBC driver.

Topics:

- JDBC Detailed Functionality (page 15-8)
- Java Classpath (page 15-8)
- JDBC Metadata Provider Configuration (page 15-9)
- Review a Sample Configuration (page 15-9)

15.3.1 JDBC Detailed Functionality

The JDBC Metadata Provider uses the JDBC driver that is provided with your target database. The DBC driver retrieves the metadata for every target table that is mapped in the Replicat properties file. Replicat processes use the retrieved target metadata to map columns.

You can enable this feature for JDBC Handler by configuring the `REPERROR` property in your Replicat parameter file. In addition, you need to define the error codes specific to your RDBMS JDBC target in the JDBC Handler properties file as follows:

Table 15-1 JDBC `REPERROR` Codes

Property	Value	Required
`gg.error.duplicateErrorCodes`	Comma-separated integer values of error codes that indicate duplicate errors	No
`gg.error.notFoundErrorCodes`	Comma-separated integer values of error codes that indicate Not Found errors	No
`gg.error.deadlockErrorCodes`	Comma-separated integer values of error codes that indicate deadlock errors	No

For example:

```
#ErrorCode
gg.error.duplicateErrorCodes=1062,1088,1092,1291,1330,1331,1332,1333
gg.error.notFoundErrorCodes=0
gg.error.deadlockErrorCodes=1213
```

To understand how the various JDBC types are mapped to database-specific SQL types, see https://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/getstart/mapping.html#table1.

15.3.2 Java Classpath

The JDBC Java Driver location must be included in the class path of the handler using the `gg.classpath` property.

For example, the configuration for a MySQL database might be:

```
gg.classpath= /path/to/jdbc/driver/jar/mysql-connector-java-5.1.39-bin.jar
```

15.3.3 JDBC Metadata Provider Configuration

The following are the configurable values for the JDBC Metadata Provider. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

Table 15-2 JDBC Metadata Provider Properties

Properties	Required / Optional	Legal Values	Default	Explanation
`gg.mdp.type`	Required	`jdbc`	None	Entering `jdbc` at a command prompt activates the use of the JDBC Metadata Provider.
`gg.mdp.ConnectionUrl`	Required	`jdbc:`*`subprotocol`*`:`*`subname`	None	The target database JDBC URL.
`gg.mdp.DriverClassName`	Required	Java class name of the JDBC driver	None	The fully qualified Java class name of the JDBC driver.
`gg.mdp.userName`	Optional	A legal username string.	None	The user name for the JDBC connection. Alternatively, you can provide the user name using the `ConnectionURL` property.
`gg.mdp.password`	Optional	A legal password string	None	The password for the JDBC connection. Alternatively, you can provide the password using the `ConnectionURL` property.

15.3.4 Review a Sample Configuration

MySQL Driver Configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:oracle:thin:@myhost:1521:orcl
gg.mdp.DriverClassName=oracle.jdbc.driver.OracleDriver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Netezza Driver Configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:netezza://hostname:port/databaseName
gg.mdp.DriverClassName=org.netezza.Driver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Oracle OCI Driver configuration

```
ggg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:oracle:oci:@myhost:1521:orcl
gg.mdp.DriverClassName=oracle.jdbc.driver.OracleDriver
```

```
gg.mdp.UserName=username
gg.mdp.Password=password
```

Oracle Teradata Driver configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:teradata://10.111.11.111/USER=username,PASSWORD=password
gg.mdp.DriverClassName=com.teradata.jdbc.TeraDriver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Oracle Thin Driver Configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:mysql://localhost/databaseName?
user=username&password=password
gg.mdp.DriverClassName=com.mysql.jdbc.Driver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Redshift Driver Configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:redshift://hostname:port/databaseName
gg.mdp.DriverClassName=com.amazon.redshift.jdbc42.Driver
gg.mdp.UserName=username
gg.mdp.Password=password
```

15.4 Hive Metadata Provider

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

Topics:

- Detailed Functionality (page 15-11)
- Configuring Hive with a Remote Metastore Database (page 15-12)
- Classpath Configuration (page 15-13)
- Hive Metadata Provider Configuration Properties (page 15-14)
- Review a Sample Configuration (page 15-15)
- Security (page 15-17)
- Metadata Change Event (page 15-17)
- Limitations (page 15-18)
- Additional Considerations (page 15-18)
- Troubleshooting (page 15-18)

15.4.1 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 the *process_name*.prm file, a corresponding table is created in Hive.

The default Hive configuration starts an embedded, local metastore Derby database. Because, Apache Derby is designed to be an embedded database, it allows only a single connection. The limitation of the Derby Database means that it cannot function when working with the Hive Metadata Provider. To workaround this limitation this, you must configure Hive with a remote metastore database. For more information about how to configure Hive with a remote metastore database, see 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 does not include a primary key definition. When you use the Hive Metadata Provider, use Replicat's KEYCOLS parameter to define primary keys.

KEYCOLS

Use the KEYCOLS parameter must be used to define primary keys in the target schema. The Oracle GoldenGate HBase Handler requires primary keys. Therefore, you must set primary keys in the target schema when you use Replicat mapping with HBase as the target.

The output of the Avro formatters includes an Array field to hold the primary column names. If you use Replicat mapping with the Avro formatters, consider using KEYCOLS to identify the primary key columns.

For example configurations of KEYCOLS, see Review a Sample Configuration (page 15-15).

Supported Hive Data types

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

See https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types.

15.4.2 Configuring Hive with a Remote Metastore Database

You can find a list of supported databases that you can use to configure remote Hive metastore can be found at https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-SupportedBackendDatabasesforMetastore.

The following example shows a MySQL database is configured as the Hive metastore using 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. If you use a database other than MySQL, then change the values to fit your configuration.

```
<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>
```

To see a list of parameters to configure in the hive-site.xml file for a remote metastore, see https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-RemoteMetastoreDatabase.

> **Note:**
>
> Follow these steps to add the MySQL JDBC connector JAR in the Hive classpath:
>
> 1. In *HIVE_HOME*/lib/ directory. *DB_NAME* should be replaced by a valid database name created in MySQL.
> 2. Start the Hive Server:
>
> *HIVE_HOME*/bin/hiveserver2/bin/hiveserver2
>
> 3. Start the Hive Remote Metastore Server:
>
> *HIVE_HOME*/bin/hive --service metastore

15.4.3 Classpath Configuration

For the Hive Metadata Provider to connect to Hive, you must configure the`hive-site.xml` file and the Hive and HDFS client jars in the `gg.classpath` variable. The client JARs must match the version of Hive to which the Hive Metadata Provider is connecting.

For example, if the `hive-site.xml` file is created in the `/home/user/oggadp/dirprm` directory, then `gg.classpath` entry is `gg.classpath=/home/user/oggadp/dirprm/`

1. Create a `hive-site.xml` file that has 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>
```

2. By default, the following directories contain the Hive and HDFS client jars:

```
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 step 1. The path to the `hive-site.xml` file must be the path with no wildcard appended. If you include the `*` wildcard in the path to the `hive-site.xml` file, it will not be located. The path to the dependency JARs must include the `*` wildcard character to include all of the JAR files in that directory in the associated classpath. Do *not* use `*.jar`.

15.4.4 Hive Metadata Provider Configuration Properties

Property	Required/ Optional	Legal Values	Default	Explanation
`gg.mdp.type`	Required	`hive`	-	Selects the 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`
			-	The JDBC connection URL of the Hive server

`gg.mdp.driverClassName`	Required	`org.apache.hive.jdbc.HiveDriver`	-	The fully qualified Hive JDBC driver class name
`gg.mdp.userName`	Optional	Valid username	`""`	The user name for connecting to the Hive database. The `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	`""`	The password for connecting to the Hive database
`gg.mdp.charset`	Optional	Valid character set	`UTF-8`	The character set of the column with the character data type. Used to convert the source data from the trail file to the correct target character set.

Property	Required/ Optional	Legal Values	Default	Explanation
gg.mdp.nationalCharset	Optional	Valid character set	UTF-8	The character set of the column with the national character data type. Used to convert the source data from the trail file to the correct target character set. For example, this property may indicate the character set of columns, such as NCHAR and NVARCHAR in an Oracle database.
gg.mdp.authType	Optional	Kerberos	none	Allows you to designate Kerberos authentication to Hive.
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 the 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.

15.4.5 Review a Sample Configuration

This is 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 *process_name.* prm file includes the following configuration:

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

In this example:

- The source schema GG is mapped to the target schema GG_HIVE.
- The source column CUST_CODE is mapped to the target column CUST_CODE2.
- The source column CITY is mapped to the target column CITY2.

- `USEDEFAULTS` specifies that rest of the column 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.

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

> **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, update the connection URL (`gg.mdp.connectionUrl`) and mapping in the `Replicat.prm` file.

You can create the schema and tables for this example in Hive by using the following commands. You can create the schema and tables for this example in Hive by using the following commands. To start the Hive CLI use the following command:

```
HIVE_HOME/bin/hive
```

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

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

To create the `TCUSTMER_HIVE` table in the `GG_HIVE` database, use 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
```

Configure the `.properties` file in a way that resembles 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
```

The following sample output uses the delimited text formatter, with a comma as the delimiter:

```
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 Hive create table SQL script are included with the installation at the following location:

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

15.4.6 Security

You can secure the Hive server using Kerberos authentication. For information about how to secure the Hive server, see the Hive documentation for the specific Hive release. The Hive Metadata Provider can connect to a Kerberos secured Hive server.

Make sure that the paths to the HDFS `core-site.xml` file and the `hive-site.xml` file are in the handler's classpath.

Enable the following properties in the `core-site.xml` file:

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

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

Enable the following properties in the `hive-site.xml` file:

```
<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>
```

15.4.7 Metadata Change Event

Tables in Hive metastore should be updated, altered, or created manually if the source database tables change. In the case of a metadata change event, you may wish to

terminate or suspend the Replicat process. You can terminate the Replicat process by adding the following to the Replicat configuration file (*process_name.* prm):

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

You can suspend, the Replicat process by adding the following to the Replication configuration file:

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

15.4.8 Limitations

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

The source-to-target mapping that is defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 and later versions supports DDL propagation and source schema evolution for Oracle databases as replication sources. If you use DDL propagation and source schema evolution, you lose the ability to seamlessly handle changes to the source metadata.

15.4.9 Additional Considerations

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

The required client JAR directories are listed in Classpath Configuration (page 15-13). Hive and HDFS client JARs do not ship with Oracle GoldenGate for Big Data. The client JARs should be of the same version as the Hive version to which the Hive Metadata Provider is connecting.

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

15.4.10 Troubleshooting

If the mapped target table is not present in Hive, the Replicat process will terminate with a "Table metadata resolution exception".

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);
```

This mapping requires a table called `TCUSTMER_HIVE` to be created in the schema `GG_HIVE` in the Hive metastore. If this table is not present in Hive, then the following exception occurs:

```
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)
```

16
Using the Oracle NoSQL Handler

Learn how to use the Oracle NoSQL Handler, which can replicate transactional data from Oracle GoldenGate to a target Oracle NoSQL Database.

Topics:

- Overview (page 16-1)
- Detailed Functionality (page 16-1)
- Oracle NoSQL Handler Configuration (page 16-5)
- Review a Sample Configuration (page 16-7)
- Performance Considerations (page 16-8)
- Full Image Data Requirements (page 16-8)

16.1 Overview

Oracle NoSQL Database is a NoSQL-type distributed key-value database. It provides a powerful and flexible transaction model that greatly simplifies the process of developing a NoSQL-based application. It scales horizontally with high availability and transparent load balancing even when dynamically adding new capacity.

Oracle NoSQL Database provides a very simple data model to the application developer. Each row is identified by a unique key, and also has a value, of arbitrary length, which is interpreted by the application. The application can manipulate (insert, delete, update, read) a single row in a transaction. The application can also perform an iterative, non-transactional scan of all the rows in the database, see https://www.oracle.com/database/nosql and https://docs.oracle.com/cd/NOSQL/docs.htm.

The Oracle NoSQL Handler streams change data capture into Oracle NoSQL using the Table API. The Table API provides some of the functionality of an RDBMS, including tables, schemas, data types, and primary keys. Oracle NoSQL also supports a Key Value API. The Key Value API stores raw data in Oracle NoSQL based on a key. The NoSQL Handler does not support the Key Value API.

16.2 Detailed Functionality

Topics:

- Oracle NoSQL Data Types (page 16-2)
- Performance Considerations (page 16-2)
- Operation Processing Support (page 16-2)
- Column Processing (page 16-3)
- Table Check and Reconciliation Process (page 16-3)
- Security (page 16-4)

16.2.1 Oracle NoSQL Data Types

Oracle NoSQL provides a number of column data types and most of these data types are supported by the Oracle NoSQL Handler. A data type conversion from the column value in the trail file to the corresponding Java type representing the Oracle NoSQL column type in the Oracle NoSQL Handler is required.

The Oracle NoSQL Handler does not support Array, Map and Record data types by default. To support them, you can implement a custom data converter and override the default data type conversion logic to override it with your own custom logic to support your use case. Contact Oracle Support for guidance.

The following Oracle NoSQL data types are supported:

- Binary
- Boolean
- Double
- Float
- Integer
- Long
- Java String

16.2.2 Performance Considerations

Configuring the Oracle NoSQL Handler for batch mode provides better performance than the interactive mode. The batch processing mode provides an efficient and transactional mechanism for executing a sequence of operations associated with tables that share the same shard key portion of their primary keys. The efficiency results from the use of a single network interaction to accomplish the entire sequence of operations. All the operations specified in a batch are executed within the scope of a single transaction that effectively provides serializable isolation.

16.2.3 Operation Processing Support

The Oracle NoSQL Handler moves operations to Oracle NoSQL using synchronous API. The Insert, update, and delete operations are processed differently in Oracle NoSQL databases rather than in a traditional RDBMS:

The following explains how insert, update, and delete operations are interpreted by the handler depending on the mode of operation:

- `insert` – If the row does not exist in your database, then an insert operation is processed as an insert. If the row exists, then an insert operation is processed as an update.
- `update` – If a row does not exist in your database, then an update operation is processed as an insert. If the row exists, then an update operation is processed as update.
- `delete` – If the row does not exist in your database, then a delete operation has no effect. If the row exists, then a delete operation is processed as a delete.

The state of the data in Oracle NoSQL databases is eventually idempotent. You can replay the source trail files or replay sections of the trail files. Ultimately, the state of an Oracle NoSQL database is the same regardless of the number of times the trail data was written into Oracle NoSQL.

Primary key values for a row in Oracle NoSQL databases are immutable. An update operation that changes any primary key value for a Oracle NoSQL row must be treated as a delete and insert. The Oracle NoSQL Handler can process update operations that result in the change of a primary key in an Oracle NoSQL database only as a delete and insert. To successfully process this operation, the source trail file *must* contain the complete before and after change data images for all columns.

16.2.4 Column Processing

Add Column Functionality

You can configure the Oracle NoSQL Handler to add columns that exist in the source trail file table definition though are missing in the Oracle NoSQL table definition. The Oracle NoSQL Handler can accommodate metadata change events of adding a column. A reconciliation process occurs that reconciles the source table definition to the Oracle NoSQL table definition. When configured to add columns, any columns found in the source table definition that do not exist in the Oracle NoSQL table definition are added. The reconciliation process for a table occurs after application start up the first time an operation for the table is encountered. The reconciliation process reoccurs after a metadata change event on a source table, when the first operation for the source table is encountered after the change event.

Drop Column Functionality

Similar to adding, you can configure the Oracle NoSQL Handler to drop columns. The Oracle NoSQL Handler can accommodate metadata change events of dropping a column. A reconciliation process occurs that reconciles the source table definition to the Oracle NoSQL table definition. When configured to drop columns, any columns found in the Oracle NoSQL table definition that are not in the source table definition are dropped.

> **Caution:**
>
> Dropping a column is potentially dangerous because it is permanently removing data from an Oracle NoSQL Database. Carefully consider your use case before configuring dropping.

Primary key columns cannot be dropped.

Column name changes are not handled well because there is no DDL-processing. The Oracle NoSQL Handler can handle any case change for the column name. A column name change event on the source database appears to the handler like dropping an existing column and adding a new column.

16.2.5 Table Check and Reconciliation Process

First, the Oracle NoSQL Handler interrogates the target Oracle NoSQL database for the table definition. If the table does not exist, the Oracle NoSQL Handler does one of

two things. If `gg.handler.name.ddlHandling` includes `CREATE`, then a table is created in the database. Otherwise, the process abends and a message is logged that tells you the table that does not exist. If the table exists in the Oracle NoSQL database, then the Oracle NoSQL Handler performs a reconciliation between the table definition from the source trail file and the table definition in the database. This reconciliation process searches for columns that exist in the source table definition and not in the corresponding database table definition. If it locates columns fitting this criteria and the `gg.handler.name.ddlHandling` property includes `ADD`, then the Oracle NoSQL Handler alters the target table in the database to add the new columns. Otherwise, those columns are ignored.

Next, the reconciliation process search for columns that exist in the target Oracle NoSQL table though do not exist in the source table definition. If it locates columns fitting this criteria and the `gg.handler.name.ddlHandling` property includes `DROP` then the Oracle NoSQL Handler alters the target table in Oracle NoSQL to drop these columns. Otherwise, those columns are ignored.

16.2.6 Security

The Oracle NoSQL Handler supports two authentication methods, Basic and Kerberos

Both of these authentication methods uses SSL as the transport mechanism to the KV Store. You *must* specify the relative or absolute path of the public trust file for SSL as a part of the Oracle NoSQL Handler configuration in the Adapter properties file.

The basic authentication mechanism tries to login into the Oracle NoSQL database using the username and password specified as configuration parameters in the properties file. You can create a credential store for your Big Data environment in Oracle GoldenGate. After you create a credential store for your Big Data environment, you can add users to the store.

To create a user, run this command in GGSCI:

```
ALTER CREDENTIALSTORE ADD USER userid PASSWORD password [ALIAS alias] [DOMAIN domain]
```

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 you don’t use this option, then you are prompted for the password. The password is obfuscated as you type (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 you don’t use the `ALIAS` option, the alias defaults to the user name.

The user created should have the access to read-write from the Oracle NoSQL database. For details about Oracle NoSQL user management, see

https://docs.oracle.com/cd/NOSQL/html/SecurityGuide/config_auth.html.

The only supported external login mechanism to the Oracle NoSQL is Kerberos. The Kerberos authentication mechanism tries to log in to the Oracle NoSQL database using the Kerberos principal, realm, and the `keytab` file. You specify these values as configuration parameters in the properties file.

The handler first tries to check if the security properties file is available to the handler for logging in to the Oracle NoSQL database as an administrator. If the `user.security` file is available to the handler, it logs in as an administrator into the database. If the security properties file is not available to the handler, it checks the `AuthType`, which can be basic or Kerberos. If the Oracle NoSQL store is configured to run with security disabled, then the handler allows access to the NoSQL store with `authType` set to `none`.

16.3 Oracle NoSQL Handler Configuration

You configure the Oracle NoSQL Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Oracle NoSQL Handler, you must first configure the handler type by specifying `gg.handler.name.type=nosql` and the other Oracle NoSQL properties as follows:

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handlerlist`	Required	Any String.	None	Provides a name for the Oracle NoSQL Handler. The Oracle NoSQL Handler name becomes part of the property names listed in the table.
`gg.handler.`*`name`*`.type`	Required	`nosql`	None	Selects the Oracle NoSQL Handler for streaming change data capture into an Oracle NoSQL Database.
`gg.handler.`*`name`*`.fullyQualifiedTableName`	Optional	`true` | `false`	`false`	The Oracle NoSQL Handler adds the schema name to the table name and stores it as a fully qualified table name in the NoSQL store.
`gg.handler.`*`name`*`.mode`	Optional	`op` | `tx`	`op`	The default is recommended. In `op` mode, operations are processed as received. In `tx` mode, operations are cached and processed at transaction commit. The `tx` mode is slower and creates a larger memory footprint.
`gg.handler.`*`name`*`.nosqlStore`	Required	Any String.	None	The name of the store. The name you specify must be identical to the name used when you installed the store.
`gg.handler.name.nosqlURL`	Required	Any String.	None	The network name and the port information for the node currently belonging to the store.
`gg.handler.`*`name`*`.interactiveMode`	Optional	`true` | `false`	`true`	The Oracle NoSQL Handler can operate in either interactive mode where one operation is processed each time or batch mode where a group of operations are processed together.

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.ddlHandling`	Optional	`CREATE | ADD| DROP` in any combination with values delimited by a comma.	None	Configure the Oracle NoSQL Handler for the DDL functionality to provide. Options include `CREATE`, `ADD` and `DROP`. When `CREATE` is enabled, the handler creates tables in Oracle NoSQL if a corresponding table does not exist. When `ADD` is enabled, the handler adds columns that exist in the source table definition, but do not exist in the corresponding target Oracle NoSQL table definition. When `DROP` is enabled, the handler drops columns that exist in the Oracle NoSQL table definition, but do not exist in the corresponding source table definition.
`gg.handler.name.retries`	Optional	Any number.	3	The number of retries on any read or write exception that the Oracle NoSQL Handler encounters.
`gg.handler.name.username`	Optional	A legal username string.	None	A username for the connection to Oracle NoSQL store. It is required if the `AuthType` is set to `basic`.
`gg.handler.name.password`	Optional	A legal password string.	None	A password for the connection to Oracle NoSQL store. It is required if the `AuthType` is set to `basic`.
`gg.handler.name.authType`	Optional	`basic| kerberos| none`	None	The authentication type to login into the Oracle NoSQL store. If `authType` is set to `basic`, it needs a username and password to login. If `authType` is set to Kerberos, it needs a Kerberos principal, Kerberos realm, and a Kerberos key tab file location to login.
`gg.handler.name.securityPropertiesFile`	Optional	Relative or absolute path to the security file.	None	The security file enables the Oracle NoSQL Handler to have administrator access into the KV Store.
`gg.handler.name.publicTrustFile`	Optional	Relative or absolute path to the trust file.	None	The public trust file to enable SSL transport.

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.kerberosKeyTabFile`	Optional	Relative or absolute path to the Kerberos key tab file	None	The key tab file allows the Oracle NoSQL Handler to access a password to perform `kinit` operation for Kerberos security.
`gg.handler.name.kerberosPrincipal`	Optional	A legal Kerberos principal name like `user/FQDN@MY.REALM`	None	The Kerberos principal name for Kerberos authentication.
`gg.handler.name.kerberosRealm`	Optional	A Kerberos Realm name	None	The Kerberos realm name for Kerberos authentication.
`gg.handler.name.dataConverterClass`	Optional	The fully qualified data converter class name.	`DefaultDataConverter`	The custom data converter can be implemented to override the default data conversion logic to support your specific use case.

16.4 Review a Sample Configuration

The following excerpt shows a sample configuration for the Oracle NoSQL Handler as it appears in the Java adapter properties file:

```
gg.handlerlist=nosql

#The handler properties
gg.handler.nosql.type= nosql
gg.handler.nosql.mode= op
gg.handler.nosql.nosqlStore= kvstore
gg.handler.nosql.nosqlURL= localhost:5000
gg.handler.nosql.ddlHandling= CREATE,ADD,DROP
gg.handler.nosql.interactiveMode=true
gg.handler.nosql.retries= 2
gg.handler.nosql.authType=basic
```

```
gg.handler.nosql.username= ORACLEWALLETUSERNAME[myalias mydomain]
gg.handler.nosql.password= ORACLEWALLETPASSWORD[myalias mydomain]
```

16.5 Performance Considerations

Configuring the Oracle NoSQL Handler for batch mode provides better performance than the interactive mode. The batch processing mode provides an efficient and transactional mechanism for executing a sequence of operations associated with tables that share the same shard key portion of their primary keys. The efficiency results from the use of a single network interaction to accomplish the entire sequence of operations. All the operations specified in a batch are executed within the scope of a single transaction that effectively provides serializable isolation.

16.6 Full Image Data Requirements

In Oracle NoSQL, update operations perform a complete reinsertion of the data for the entire row. This Oracle NoSQL feature improves ingest performance, but in turn levies a critical requirement. Updates must include data for all columns, also known as **full image updates**. Partial image updates are not supported (updates with just the primary key information and data for the columns that changed). Using the Oracle NoSQL Handler with partial image update information results in incomplete data in the target NoSQL table.

17
Using the Pluggable Formatters

Learn how to use the pluggable formatters to convert operations from the Oracle GoldenGate trail file into formatted messages that you can send to Big Data targets using one of the Oracle GoldenGate for Big Data Handlers.

Topics:

- Using the Avro Formatter (page 17-1)
- Using the Delimited Text Formatter (page 17-25)
- Using the JSON Formatter (page 17-34)
- Using the Length Delimited Value Formatter (page 17-52)
- Using Operation-Based versus Row-Based Formatting (page 17-57)
- Using the XML Formatter (page 17-59)

17.1 Using the Avro Formatter

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

Topics:

- Avro Row Formatter (page 17-1)
- The Avro Operation Formatter (page 17-10)
- Avro Object Container File Formatter (page 17-18)
- Setting Metacolumn Output (page 17-23)

17.1.1 Avro Row Formatter

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 is formatted into an individual Avro message. The source trail file contains the before and after images of the operation data. The Avro Row Formatter takes the before-image and after-image data and formats it 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 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.

This section contains the following topics:

- Operation Metadata Formatting Details (page 17-2)

- Operation Data Formatting Details (page 17-2)
- Sample Avro Row Messages (page 17-3)
- Avro Schemas (page 17-4)
- Avro Row Configuration Properties (page 17-5)
- Review a Sample Configuration (page 17-7)
- Metadata Change Events (page 17-8)
- Special Considerations (page 17-8)

17.1.1.1 Operation Metadata Formatting Details

In Avro messages generated by the Avro Row Formatter, the following seven metadata fields begin each message:

Table 17-1 Avro Formatter Metadata

Value	Description
`table`	The fully qualified table in the format is: *`CATALOG_NAME.SCHEMA_NAME.TABLE_NAME`*
`op_type`	The type of database operation from the source trail file. Default values are `I` for insert, `U` for update, `D` for delete, and `T` for truncate.
`op_ts`	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 results in the same timestamp for the same operation.
`current_ts`	The 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 concatenated sequence number and the RBA number from the source trail file. This trail position lets you trace 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 that holds the column names of the primary keys of the source table.
`tokens`	A map variable that holds the token key value pairs from the source trail file.

17.1.1.2 Operation Data Formatting Details

The operation data follows the operation metadata. 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: the column has a value, the column value is null, or the column value is missing. Avro attributes only support two states, the column has a value or the column value is null. Missing column values are handled the same as null values. Oracle recommends that when you use the Avro Row Formatter, you configure the Oracle GoldenGate capture process to provide full image data for all columns in the source trail file.

By default, the setting of the Avro Row Formatter maps the data types from the source trail file to the associated Avro data type. Because Avro provides limited support for data types, source columns map into Avro long, double, float, binary, or string data types. You can also configure data type mapping to handle all data as strings.

17.1.1.3 Sample Avro Row Messages

Because Avro messages are binary, they are not human readable. The following sample messages show the JSON representation of the messages.

- Sample Insert Message (page 17-3)
- Sample Update Message (page 17-3)
- Sample Delete Message (page 17-3)
- Sample Truncate Message (page 17-4)

17.1.1.3.1 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"}
```

17.1.1.3.2 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"}
```

17.1.1.3.3 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}
```

17.1.1.3.4 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}
```

17.1.1.4 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 just-in-time basis when the first operation for a table is encountered. Because generated Avro schemas are specific to a table definition, a separate Avro schema is generated for every table encountered for processed operations. By default, Avro schemas are written to the *GoldenGate_Home*/dirdef directory, although the write location is configurable. Avro schema file names adhere to the following naming convention: *Fully_Qualified_Table_Name*.avsc.

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

```
{
  "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
  } ]
}
```

17.1.1.5 Avro Row Configuration Properties

Table 17-2 Avro Row Configuration Properties

Properties	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.

Table 17-2 (Cont.) Avro Row Configuration Properties

Properties	Optional / Required	Legal Values	Default	Explanation
`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 JSON Avro schema. 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.
`gg.handler.name.format.pkUpdateHandlingformat.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Specifies how the formatter handles update operations that change a primary key. Primary key operations for the Avro Row formatter require special consideration. • `abend`: the process terminates. • `update`: the process handles the update as a normal update. • `delete` or `insert`: the process handles the update as a delete and an insert. Full supplemental logging must be enabled. Without full before and after row images, the insert data will be incomplete.
`gg.handler.name.format.lineDelimiter`	Optional	Any string	no value	Inserts a delimiter after each Avro message. This is not a best practice, but in certain cases you may want to parse a stream of data and extract individual Avro messages from the stream. Select a unique delimiter that cannot occur in any Avro message. This property supports `CDATA[]` wrapping.

Table 17-2 (Cont.) Avro Row Configuration Properties

Properties	Optional / Required	Legal Values	Default	Explanation
`gg.handler.name.format.versionSchemas`	Optional	`true|false`	`false`	Avro schemas always follow the*`fully_qualified_table_name`*`.avsc` convention. Setting this property to `true` creates an additional Avro schema named *`fully_qualified_table_name_current_timestamp`*`.avsc` in the schema directory. Because the additional Avro schema is not destroyed or removed, provides a history of schema evolution.
`gg.handler.name.format.wrapMessageInGenericAvroMessage`	Optional	`true|false`	`false`	Wraps the Avro messages for operations from the source trail file in a generic Avro wrapper message. For more information, see Generic Wrapper Functionality (page 17-9).
`gg.handler.name.format.schemaDirectory`	Optional	Any legal, existing file system path.	`./dirdef`	The output location of generated Avro schemas.
`gg.handler.name.schemaFilePath=`	Optional	Any legal encoding name or alias supported by Java.	`./dirdef`	The directory in the HDFS where schemas are output. A metadata change overwrites the schema during the next operation for the associated table. 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`	The format of the current timestamp. The default is the ISO 8601 format. A setting of `false` removes the `T` between the date and time in the current timestamp, which outputs a space instead.

17.1.1.6 Review a Sample Configuration

The following is a sample configuration for the Avro Row Formatter in 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
```

17.1.1.7 Metadata Change Events

If the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events, the Avro Row Formatter can take action when metadata changes. Because Avro messages depend closely on their corresponding schema, metadata changes are important when you use Avro formatting.

An updated Avro schema is generated as soon as a table operation occurs after a metadata change event. You must 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. For more information, consult the Apache Avro documentation.

17.1.1.8 Special Considerations

This sections describes these special considerations:

- Troubleshooting (page 17-8)
- Primary Key Updates (page 17-8)
- Generic Wrapper Functionality (page 17-9)

17.1.1.8.1 Troubleshooting

Because Avro is a binary format, it is not human readable. Since Avro messages are in binary format, it is difficult to debug any issue, the Avro Row Formatter provides a special feature to help debug issues. When the `log4j` Java logging level is set to `TRACE` , Avro messages are deserialized and displayed in the log file as a JSON object, letting you view the structure and contents of the created Avro messages. Do not enable `TRACE` in a production environment as it has substantial negative impact on performance. To troubleshoot content, you may want to consider switching to use a formatter that produces human-readable content. The XML or JSON formatters both produce content in human-readable format.

17.1.1.8.2 Primary Key Updates

In Big Data integrations, primary key update operations require special consideration and planning. Primary key updates modify one or more of the primary keys of a given row in the source database. Because data is appended in Big Data applications, a primary key update operation looks more like a new insert than like an update without special handling. You can use the following properties to configure the Avro Row Formatter to handle primary keys:

Table 17-3 Configurable behavior

Value	Description
`abend`	The formatter terminates. This behavior is the default behavior.
`update`	With this configuration the primary key update is treated like any other update operation. Use this configuration only if you can guarantee that the primary key is not used as selection criteria row data from a Big Data system.
`delete-insert`	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 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.

17.1.1.8.3 Generic Wrapper Functionality

Because Avro messages are not self describing, 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 troublesome when messages are interlaced into a single stream of data such as Kafka.

The Avro formatter provides a special feature to wrap the Avro message in a generic Avro message. You can enable this functionality by setting the following configuration property.

```
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_fingerprint` : The fingerprint of the Avro schema of the wrapped message. The fingerprint is generated using the Avro `SchemaNormalization.parsingFingerprint64(schema)` call.
- `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_fingerprint",
    "type" : "long"
  }, {
    "name" : "payload",
    "type" : "bytes"
  } ]
}
```

17.1.2 The Avro Operation Formatter

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 is formatted into an individual Avro message. The source trail file contains the before and after images of the operation data. The Avro Operation Formatter formats this data into an Avro binary representation of the operation data.

This format is more verbose than the output of the Avro Row Formatter for which the Avro messages model the row data.

This section contains the following topics:

- Operation Metadata Formatting Details (page 17-10)
- Operation Data Formatting Details (page 17-11)
- Sample Avro Operation Messages (page 17-11)
- Avro Schema (page 17-13)
- Avro Operation Formatter Configuration Properties (page 17-15)
- Review a Sample Configuration (page 17-16)
- Metadata Change Events (page 17-17)
- Special Considerations (page 17-17)

17.1.2.1 Operation Metadata Formatting Details

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

Table 17-4 Avro Messages and its Metadata

Fields	Description
`table`	The fully qualified table name, in the format: *`CATALOG_NAME.SCHEMA NAME.TABLE NAME`*
`op_type`	The type of database operation from the source trail file. Default values are `I` for insert, `U` for update, `D` for delete, and `T` for truncate.
`op_ts`	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 results in the same timestamp for the same operation.
`current_ts`	The 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 17-4 (Cont.) Avro Messages and its Metadata

Fields	Description
`pos`	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 that holds the column names of the primary keys of the source table.
`tokens`	A map variable that holds the token key value pairs from the source trail file.

17.1.2.2 Operation Data Formatting Details

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: the column has a value, the column value is null, or the column value is missing. Avro attributes only support two states: the column has a value or the column value is null. The Avro Operation Formatter contains an additional Boolean field *`COLUMN_NAME`*`_isMissing` for each column to indicate whether the column value is missing or not. Using *`COLUMN_NAME`* field together with the *`COLUMN_NAME`*`_isMissing` field, all three states can be defined.

- State 1: The column has a value

  `COLUMN_NAME` field has a value

  `COLUMN_NAME``_isMissing` field is false
- State 2: The column value is null

  `COLUMN_NAME` field value is null

  `COLUMN_NAME``_isMissing` field is false
- State 3: The column value is missing

  `COLUMN_NAME` field value is null

  `COLUMN_NAME``_isMissing` field is true

By default the Avro Row Formatter maps the data types from the source trail file to the associated Avro data type. Because Avro supports few data types, this functionality usually results in the mapping of numeric fields from the source trail file to members typed as numbers. You can also configure this data type mapping to handle all data as strings.

17.1.2.3 Sample Avro Operation Messages

Because Avro messages are binary, they are not human readable. The following topics show example Avro messages in JSON format:

- Sample Insert Message (page 17-12)
- Sample Update Message (page 17-12)
- Sample Delete Message (page 17-13)

- Sample Truncate Message (page 17-13)

17.1.2.3.1 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}}
```


17.1.2.3.2 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}}
```

17.1.2.3.3 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}
```

17.1.2.3.4 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}
```

17.1.2.4 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 just-in-time basis when the first operation for a table is encountered. Because Avro schemas are specific to a table definition, a separate Avro schema is generated for every table encountered for processed operations. By default, Avro schemas are written to the *GoldenGate_Home*/dirdef directory, although the write location is configurable. Avro schema file names adhere to the following naming convention: *Fully_Qualified_Table_Name*.avsc .

The following is a sample Avro schema for the Avro Operation Format for the samples in the preceding sections:

```
{
  "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
  } ]
}
```

17.1.2.5 Avro Operation Formatter Configuration Properties

Table 17-5 Configuration Properties

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 17-5 (Cont.) Configuration Properties

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 JSON Avro schema. 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 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.
`gg.handler.name.format.lineDelimiter`	Optional	Any string	no value	Inserts delimiter after each Avro message. This is not a best practice, but in certain cases you may want to parse a stream of data and extract individual Avro messages from the stream, use this property to help. 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`	The output location of generated Avro schemas.
`gg.handler.name.format.wrapMessageInGenericAvroMessage`	Optional	`true`|`false`	`false`	Wraps Avro messages for operations from the source trail file in a generic Avro wrapper message. For more information, see Generic Wrapper Functionality (page 17-9).
`gg.handler.name.format.iso8601Format`	Optional	`true` | `false`	`true`	The format of the current timestamp. By default the ISO 8601 is set to `false`, removes the `T` between the date and time in the current timestamp, which outputs a space instead.

17.1.2.6 Review a Sample Configuration

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

```
gg.hdfs.format=avro_op
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.wrapMessageInGenericAvroMessage=false
```

17.1.2.7 Metadata Change Events

If the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events, the Avro Operation Formatter can take action when metadata changes. Because Avro messages depend closely on their corresponding schema, metadata changes are important when you use Avro formatting.

An updated Avro schema is generated as soon as a table operation occurs after a metadata change event.

You must 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

For more information, consult the Apache Avro documentation.

17.1.2.8 Special Considerations

This section describes these special considerations:

- Troubleshooting (page 17-17)
- Primary Key Updates (page 17-17)
- Generic Wrapper Message (page 17-17)

17.1.2.8.1 Troubleshooting

Because Avro is a binary format, it is not human readable. However, when the `log4j` Java logging level is set to `TRACE`, Avro messages are deserialized and displayed in the log file as a JSON object, letting you view the structure and contents of the created Avro messages. Do not enable `TRACE` in a production environment, as it has a substantial impact on performance.

17.1.2.8.2 Primary Key Updates

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

17.1.2.8.3 Generic Wrapper Message

Because Avro messages are not self describing, 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 troublesome when messages are interlaced into a single stream of data such as Kafka.

The Avro formatter provides a special feature to wrap the Avro message in a generic Avro message. You can enable this functionality by setting the following configuration property:

```
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_fingerprint` : The fingerprint of the of the Avro schema generating the messages. The fingerprint is generated using the `parsingFingerprint64(Schema s)` method on the `org.apache.avro.SchemaNormalization` class.
- `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_fingerprint",
    "type" : "long"
  }, {
    "name" : "payload",
    "type" : "bytes"
  } ]
}
```

17.1.3 Avro Object Container File Formatter

Oracle GoldenGate for Big Data can write to HDFS in Avro Object Container File (OCF) format. Avro OCF handles schema evolution more efficiently than other formats. The Avro OCF Formatter also supports compression and decompression to allow more efficient use of disk space.

The HDFS Handler integrates with the Avro formatters to write files to HDFS in Avro OCF format. The Avro OCF format is required for Hive to read Avro data in HDFS. The Avro OCF format is detailed in the Avro specification, see http://avro.apache.org/docs/current/spec.html#Object+Container+Files.

You can configure the HDFS Handler 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.

- Avro OCF Formatter Configuration Properties (page 17-19)

17.1.3.1 Avro OCF Formatter Configuration Properties

Properties	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 JSON Avro schema. The JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding.

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

Properties	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` : the process will terminates. • `update` : the process handles this as a normal update • `delete` and `insert`: the process handles thins operation 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.

Properties	Optional / Required	Legal Values	Default	Explanation
`gg.handler.name.format.generateSchema`	Optional	`true` | `false`	`true`	Because schemas must be generated for Avro serialization 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`	The directory where generated Avro schemas are saved 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`	By default, the value of this property is true, and the 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.
`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 schema uses the following format: *`fully_qualified table_name_time stamp`*`.avsc`

17.1.4 Setting Metacolumn Output

The following are the configurable values for all Avro Formatter metacolumns template property that controls metacolumn output.

Table 17-6 Metacolumns Template Property

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.metaColumnsTemplate`	Optional	`${alltokens} | ${token} | ${env} | ${sys} | ${javaprop} | ${optype} | ${position} | ${timestamp} | ${catalog} | ${schema} | ${table} | ${objectname} | ${csn} | ${xid} | ${currenttimestamp} | ${opseqno} | ${timestampmicro} | ${currenttimestampmicro}`	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: `insertOpKey | updateOpKey | deleteOpKey | truncateOpKey | includeTableName | includeOpTimestamp | includeOpType | includePosition | includeCurrentTimestamp, useIso8601Format` It is a comma-delimited string consisting of one or more templated values that represent the template.

Explanation of the Metacolumn Keywords

`${alltokens}`
All of the Oracle GoldenGate tokens.

`${token}`
The value of a specific Oracle GoldenGate token. The token key should follow token key should follow the token using the period (`.`) operator. For example:

```
${token.MYTOKEN}
```

`${sys}`
A system environmental variable. The variable name should follow `sys` using the period (`.`) operator. For example:

```
${sys.MYVAR}
```

`${env}`
An Oracle GoldenGate environment variable. The variable name should follow `env` using the period (`.`) operator. For example:

```
${env.someVariable}
```

`${javaprop}`
A Java JVM variable. The variable name should follow `javaprop` using the period (`.`) operator. For example:

```
${javaprop.MYVAR}
```

`${optype}`
Operation Type

`${position}`
Record Position

`${timestamp}`
Record Timestamp

`${catalog}`
Catalog Name

`${schema}`
Schema Name

`${table}`
Table Name

`${objectname}`
The fully qualified table name.

`${csn}`
Source Commit Sequence Number

`${xid}`
Source Transaction ID

`${currenttimestamp}`
Current Timestamp

`${opseqno}`
Record sequence number within the transaction.

`${timestampmicro}`
Record timestamp (in microseconds after epoch).

`${currenttimestampmicro}`
Current timestamp (in microseconds after epoch).

17.2 Using the 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 is formatted into an individual delimited message. Delimited text output includes 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 that contain delimited text.

Column values for an operation from the source trail file can have one of three states: the column has a value, the column value is null, or the 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 an empty string ("). The output for the case of a missing column value is configurable.

Topics:

- Message Formatting Details (page 17-25)
- Sample Formatted Messages (page 17-26)
- Output Format Summary Log (page 17-27)
- Delimited Text Formatter Configuration Properties (page 17-27)
- Review a Sample Configuration (page 17-29)
- Metadata Change Events (page 17-30)
- Setting Metacolumn Output (page 17-30)
- Additional Considerations (page 17-32)

17.2.1 Message Formatting Details

The default output format uses a semicolon as the delimiter and resembles the following:

First is the row metadata:

```
operation_type;fully_qualified_table_name;operation_timestamp;current_timestamp;trail
_position;tokens;
```

Next is the row data:

```
column_1_value;column_n_value_then_line_delimeter
```

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

```
column_1_name;column_1_value;column_n_name;column_n_value_then_line_delimeter
```

Formatting details:

- **Operation Type** : Indicates 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 including the catalog name, and the schema name. The format of the fully qualified table name is *`catalog_name.schema_name.table_name`*. The output of this field is suppressible.
- **Operation Timestamp** : The commit record timestamp from the source system. All operations in a transaction (unbatched transaction) will have the same operation timestamp. This timestamp is fixed, and the operation timestamp is the same if the trail file is replayed. The output of this field is suppressible.
- **Current Timestamp** : The timestamp of the current time when the delimited text formatter processes the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file does *not* result in the same timestamp for the same operation. The output of this field is suppressible.
- **Trail Position** :The concatenated sequence number and RBA number from the source trail file. The trail position lets you trace 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. The output of this field is suppressible.
- **Tokens** : The token key value pairs from the source trail file. The output of this field in the delimited text output is suppressed unless the `includeTokens` configuration property on the corresponding handler is explicitly set to `true`.

17.2.2 Sample Formatted Messages

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

- Sample Insert Message (page 17-26)
- Sample Update Message (page 17-26)
- Sample Delete Message (page 17-27)
- Sample Truncate Message (page 17-27)

17.2.2.1 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
```

17.2.2.2 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
```

17.2.2.3 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|||
```

17.2.2.4 Sample Truncate Message

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

17.2.3 Output Format Summary Log

If `INFO` level logging is enabled, the Java `log4j` logging logs a summary of the delimited text output format . A summary of the delimited fields is logged for each source table encountered and occurs when the first operation for that table is received by the Delimited Text formatter. This detailed explanation of the fields of the delimited text output may be useful when you perform an initial setup. When a metadata change event occurs, the summary of the delimited fields is regenerated and logged again at the first subsequent operation for that table.

17.2.4 Delimited Text Formatter Configuration Properties

Table 17-7 Delimited Text Formatter Configuration Properties

Properties	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. When true, the output resembles: *`COL1_Name`*`|`*`COL1_Value`*`|`*`COL2_Name`*`|`*`COL2_Value`* When false, the output resembles: *`COL1_Value`*`|I`
`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.

Table 17-7 (Cont.) Delimited Text Formatter Configuration Properties

Properties	Optional / Required	Legal Values	Default	Explanation
`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 encoding name or alias supported by Java.	The native system encoding of the machine hosting the Oracle GoldenGate 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	`=`	Specifies 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	`,`	Specifies a delimiter between key value pairs in a map. *`Key1=Value1,Key2=Value2`*. Tokens are mapped values. Configuration value supports `CDATA[]` wrapping.
`gg.handler.name.format.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Specifies how the formatter handles 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 handles this as a delete and an insert. Full supplemental logging must be enabled for this to work. Without full before and after row images, the insert data will be incomplete.

Table 17-7 (Cont.) Delimited Text Formatter Configuration Properties

Properties	Optional / Required	Legal Values	Default	Explanation
`gg.handler.name.format.nullValueRepresentation`	Optional	Any string	NULL	Specifies what is 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)	Specifies what is 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`	When true, suppresses the output of the operation position from the source trail file.
`gg.handler.name.format.iso8601Format`	Optional	`true` | `false`	`true`	Controls the format of the current timestamp. The default is the ISO 8601 format. When `false`, removes the `T` between the date and time in the current timestamp, which outputs a space instead.
`gg.handler.name.format.includeMetaColumnNames`	Optional	`true` | `false`	`false`	Set to `true`, a field is included prior to each metadata column value, which is the column name of the metadata column. You can use it to make delimited messages more self-describing.
`gg.handler.name.format.wrapStringsInQuotes`	Optional	`true` | `false`	`false`	Set to `true` to wrap string value output in the delimited text format in double quotes (`"`).

17.2.5 Review a Sample Configuration

The following is a sample configuration for the Delimited Text formatter in 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.keyValuePairDelimiter=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
```

17.2.6 Metadata Change Events

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

Note that a metadata change may affect downstream applications. Delimited text formats include 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 results in a change in the total number of fields, and potentially changes the positional relevance of some fields. Adding an additional column or columns is probably the least impactful metadata change event, assuming that the new column is added to the end. Consider the impact of a metadata change event before executing the event. When metadata change events are frequent, Oracle recommends that you consider a more flexible and self-describing format, such as JSON or XML.

17.2.7 Setting Metacolumn Output

The following are the configurable values for the Delimited Text Format metacolumns template property that controls metacolumn output.

Table 17-8 Metacolumns Template Property

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.metaColumnsTemplate`	Optional	`${alltokens} | ${token} | ${env} | ${sys} | ${javaprop} | ${optype} | ${position} | ${timestamp} | ${catalog} | ${schema} | ${table} | ${objectname} | ${csn} | ${xid} | ${currenttimestamp} | ${opseqno} | ${timestampmicro} | ${currenttimestampmicro}`	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: `insertOpKey | updateOpKey | deleteOpKey | truncateOpKey | includeTableName | includeOpTimestamp | includeOpType | includePosition | includeCurrentTimestamp, useIso8601Format` It is a comma-delimited string consisting of one or more templated values that represent the template.

Explanation of the Metacolumn Keywords

`${alltokens}`
All of the Oracle GoldenGate tokens.

`${token}`
The value of a specific Oracle GoldenGate token. The token key should follow token key should follow the token using the period (`.`) operator. For example:

```
${token.MYTOKEN}
```

`${sys}`
A system environmental variable. The variable name should follow `sys` using the period (`.`) operator. For example:

```
${sys.MYVAR}
```

`${env}`
An Oracle GoldenGate environment variable. The variable name should follow `env` using the period (`.`) operator. For example:

```
${env.someVariable}
```

`${javaprop}`
A Java JVM variable. The variable name should follow `javaprop` using the period (`.`) operator. For example:

```
${javaprop.MYVAR}
```

`${optype}`
Operation Type

`${position}`
Record Position

`${timestamp}`
Record Timestamp

`${catalog}`
Catalog Name

`${schema}`
Schema Name

`${table}`
Table Name

`${objectname}`
The fully qualified table name.

`${csn}`
Source Commit Sequence Number

`${xid}`
Source Transaction ID

`${currenttimestamp}`
Current Timestamp

`${opseqno}`
Record sequence number within the transaction.

`${timestampmicro}`
Record timestamp (in microseconds after epoch).

`${currenttimestampmicro}`
Current timestamp (in microseconds after epoch).

17.2.8 Additional Considerations

Exercise care when you choose field and line delimiters. It is important to choose delimiter values that will not occur in the content of the data.

The Java Adapter configuration trims leading and trailing characters from configuration values when they are determined to be whitespace. However, you may want to choose

field delimiters, line delimiters, null value representations, and missing value representations that include or are fully considered to be whitespace . In these cases, you must employ specialized syntax in the Java Adapter configuration file to preserve the whitespace. To preserve the whitespace, when your configuration values contain leading or trailing characters that are considered whitespace, wrap the configuration value in a `CDATA[]` wrapper. For example, a configuration value of `\n` should be configured as `CDATA[\n]`.

You can use regular expressions to search column values then replace matches with a specified value. You can use this search and replace functionality together 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 (page 1-11).

Big Data applications sore data differently from RDBMSs. Update and delete operations in an RDBMS result in a change to the existing data. However, in Big Data applications, data is appended instead of changed. Therefore, the current state of a given row consolidates all of the existing operations for that row in the HDFS system. This leads to some special scenarios as described in the following sections.

- Primary Key Updates (page 17-33)
- Data Consolidation (page 17-34)

17.2.8.1 Primary Key Updates

In Big Data integrations, primary key update operations require special consideration and planning. Primary key updates modify one or more of the primary keys for the given row from the source database. Because data is appended in Big Data applications, a primary key update operation looks more like an insert than an update without any special handling. You can configure how the Delimited Text formatter handles primary key updates. These are the configurable behaviors:

Table 17-9 Configurable Behavior

Value	Description
`abend`	By default the delimited text formatter terminates in the case of a primary key update.
`update`	The primary key update is treated like any other update operation. Use this configuration alternative only if you can guarantee that the primary key is not used as selection criteria to select row data from a Big Data system.
`delete-insert`	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.

17.2.8.2 Data Consolidation

Big Data applications append data to the underlying storage. Analytic tools generally spawn MapReduce programs that traverse the data files and consolidate all the operations for a given row into a single output. Therefore, it is important to specify the order of operations. The Delimited Text formatter provides a number of metadata fields to do this. The operation timestamp may be sufficient to fulfill this requirement. Alternatively, the current timestamp may be the best indicator of the order of operations. In this situation, 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.

17.3 Using the JSON Formatter

The JavaScripts Object Notation (JSON) formatter can output operations from the source trail file in either row-based format or operation-based format. It formats operation data from the source trail file into a JSON objects. Each insert, update, delete, and truncate operation is formatted into an individual JSON message.

Topics:

- Operation Metadata Formatting Details (page 17-34)
- Operation Data Formatting Details (page 17-35)
- Row Data Formatting Details (page 17-36)
- Sample JSON Messages (page 17-36)
- JSON Schemas (page 17-40)
- JSON Formatter Configuration Properties (page 17-47)
- Review a Sample Configuration (page 17-49)
- Metadata Change Events (page 17-50)
- Setting Metacolumn Output (page 17-50)
- JSON Primary Key Updates (page 17-52)
- Integrating Oracle Stream Analytics (page 17-52)

17.3.1 Operation Metadata Formatting Details

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

Table 17-10 JSON Metadata

Value	Description
`table`	Contains the fully qualified table name. The format of the fully qualified table name is: *`CATALOG NAME.SCHEMA NAME.TABLE NAME`*
`op_type`	Indicates the type of database operation from the source trail file. Default values are `I` for insert, `U` for update, `D` for delete, and `T` for truncate.

Table 17-10 (Cont.) JSON Metadata

Value	Description
`op_ts`	The timestamp of the operation from the source trail file. Because this timestamp is from the source trail, it is fixed. Replaying the trail file results in the same timestamp for the same operation.
`current_ts`	The time when the 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`	The trail file position, which includes the concatenated sequence number and the 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 that holds the column names of the primary keys of the source table. The `primary_keys` field is only included in the JSON output if the `includePrimaryKeys` configuration property is set to `true`.
`tokens`	The tokens field is only included in the output if the `includeTokens` handler configuration property is set to `true`.

17.3.2 Operation Data Formatting Details

JSON messages begin with the operation metadata fields, which are followed by the operation data fields. This data is represented by `before` and `after` members that are objects. These objects contain members whose keys are the column names and whose values are the column values.

Operation data is modeled as follows:

- Inserts: Includes the after-image data.
- Updates: Includes both the before-image and the after-image data.
- Deletes: Includes the before-image data.

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

- The 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"    }
```

- The 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    }
```

- The column value is missing: The JSON contains 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 usually results in the mapping of numeric fields from the source trail file to members typed as numbers. This data type mapping can be configured treat all data as strings.

17.3.3 Row Data Formatting Details

JSON messages begin with the operation metadata fields, which are followed by the operation data fields. For row data formatting, this are the source column names and source column values as JSON key value pairs. This data is represented by `before` and `after` members that are objects. These objects contain members whose keys are the column names and whose values are the column values.

Row data is modeled as follows:

- Inserts: Includes the after-image data.
- Updates: Includes the after-image data.
- Deletes: Includes the before-image data.

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

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

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

- The column value is null :The default output value is a JSON NULL. In the following example, the member `STATE` is null.

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

- The column value is missing: The JSON contains no element for a missing column value. In the following example, the member `STATE` is missing.

```
        "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 usually results in the mapping of numeric fields from the source trail file to members typed as numbers. This data type mapping can be configured to treat all data as strings.

17.3.4 Sample JSON Messages

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

- Sample Operation Modeled JSON Messages (page 17-37)

- Sample Flattened Operation Modeled JSON Messages (page 17-38)
- Sample Row Modeled JSON Messages (page 17-39)
- Sample Primary Key Output JSON Message (page 17-40)

17.3.4.1 Sample Operation Modeled JSON Messages

Insert

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"I",
    "op_ts":"2015-11-05 18:45:36.000000",
    "current_ts":"2016-10-05T10:15:51.267000",
    "pos":"00000000000000002928",
    "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
    }
}
```

Update

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"U",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T10:15:51.310002",
    "pos":"00000000000000004300",
    "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
    }
}
```

Delete

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"D",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T10:15:51.312000",
    "pos":"00000000000000005272",
```

```
    "before":{
        "CUST_CODE":"DAVE",
        "ORDER_DATE":"1993-11-03:07:51:35",
        "PRODUCT_CODE":"PLANE",
        "ORDER_ID":600,
        "PRODUCT_PRICE":135000.00,
        "PRODUCT_AMOUNT":2,
        "TRANSACTION_ID":200
    }
}
```

Truncate

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"T",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T10:15:51.312001",
    "pos":"00000000000000005480",
}
```

17.3.4.2 Sample Flattened Operation Modeled JSON Messages

Insert

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"I",
    "op_ts":"2015-11-05 18:45:36.000000",
    "current_ts":"2016-10-05T10:34:47.956000",
    "pos":"00000000000000002928",
    "after.CUST_CODE":"WILL",
    "after.ORDER_DATE":"1994-09-30:15:33:00",
    "after.PRODUCT_CODE":"CAR",
    "after.ORDER_ID":144,
    "after.PRODUCT_PRICE":17520.00,
    "after.PRODUCT_AMOUNT":3,
    "after.TRANSACTION_ID":100
}
```

Update

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"U",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T10:34:48.192000",
    "pos":"00000000000000004300",
    "before.CUST_CODE":"BILL",
    "before.ORDER_DATE":"1995-12-31:15:00:00",
    "before.PRODUCT_CODE":"CAR",
    "before.ORDER_ID":765,
    "before.PRODUCT_PRICE":15000.00,
    "before.PRODUCT_AMOUNT":3,
    "before.TRANSACTION_ID":100,
    "after.CUST_CODE":"BILL",
    "after.ORDER_DATE":"1995-12-31:15:00:00",
    "after.PRODUCT_CODE":"CAR",
    "after.ORDER_ID":765,
    "after.PRODUCT_PRICE":14000.00
}
```

Delete

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"D",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T10:34:48.193000",
    "pos":"00000000000000005272",
    "before.CUST_CODE":"DAVE",
    "before.ORDER_DATE":"1993-11-03:07:51:35",
    "before.PRODUCT_CODE":"PLANE",
    "before.ORDER_ID":600,
    "before.PRODUCT_PRICE":135000.00,
    "before.PRODUCT_AMOUNT":2,
    "before.TRANSACTION_ID":200
}
```

Truncate

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"D",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T10:34:48.193001",
    "pos":"00000000000000005480",
    "before.CUST_CODE":"JANE",
    "before.ORDER_DATE":"1995-11-11:13:52:00",
    "before.PRODUCT_CODE":"PLANE",
    "before.ORDER_ID":256,
    "before.PRODUCT_PRICE":133300.00,
    "before.PRODUCT_AMOUNT":1,
    "before.TRANSACTION_ID":100
}
```

17.3.4.3 Sample Row Modeled JSON Messages

Insert

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"I",
    "op_ts":"2015-11-05 18:45:36.000000",
    "current_ts":"2016-10-05T11:10:42.294000",
    "pos":"00000000000000002928",
    "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
}
```

Update

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"U",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T11:10:42.350005",
```

```
    "pos":"00000000000000004300",
    "CUST_CODE":"BILL",
    "ORDER_DATE":"1995-12-31:15:00:00",
    "PRODUCT_CODE":"CAR",
    "ORDER_ID":765,
    "PRODUCT_PRICE":14000.00
}
```

Delete

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"D",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T11:10:42.351002",
    "pos":"00000000000000005272",
    "CUST_CODE":"DAVE",
    "ORDER_DATE":"1993-11-03:07:51:35",
    "PRODUCT_CODE":"PLANE",
    "ORDER_ID":600,
    "PRODUCT_PRICE":135000.00,
    "PRODUCT_AMOUNT":2,
    "TRANSACTION_ID":200
}
```

Truncate

```
{
    "table":"QASOURCE.TCUSTORD",
    "op_type":"T",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-10-05T11:10:42.351003",
    "pos":"00000000000000005480",
}
```

17.3.4.4 Sample Primary Key Output JSON Message

```
{
    "table":"DDL_OGGSRC.TCUSTMER",
    "op_type":"I",
    "op_ts":"2015-10-26 03:00:06.000000",
    "current_ts":"2016-04-05T08:59:23.001000",
    "pos":"00000000000000006605",
    "primary_keys":[
        "CUST_CODE"
    ],
    "after":{
        "CUST_CODE":"WILL",
        "NAME":"BG SOFTWARE CO.",
        "CITY":"SEATTLE",
        "STATE":"WA"
    }
}
```

17.3.5 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. By default, the JSON schemas are created in the *GoldenGate_Home*/dirdef directory and are named by the following convention:

FULLY_QUALIFIED_TABLE_NAME.schema.json

The generation of the JSON schemas is suppressible.

The following JSON schema example is for the JSON object listed in Sample Operation Modeled JSON Messages (page 17-37).

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"QASOURCE.TCUSTORD",
    "description":"JSON schema for table QASOURCE.TCUSTORD",
    "definitions":{
        "row":{
            "type":"object",
            "properties":{
                "CUST_CODE":{
                    "type":[
                        "string",
                        "null"
                    ]
                },
                "ORDER_DATE":{
                    "type":[
                        "string",
                        "null"
                    ]
                },
                "PRODUCT_CODE":{
                    "type":[
                        "string",
                        "null"
                    ]
                },
                "ORDER_ID":{
                    "type":[
                        "number",
                        "null"
                    ]
                },
                "PRODUCT_PRICE":{
                    "type":[
                        "number",
                        "null"
                    ]
                },
                "PRODUCT_AMOUNT":{
                    "type":[
                        "integer",
                        "null"
                    ]
                },
                "TRANSACTION_ID":{
                    "type":[
                        "number",
```

```
                    "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"
    },
    "primary_keys":{
        "description":"Array of the primary key column names.",
        "type":"array",
        "items":{
            "type":"string"
        },
        "minItems":0,
        "uniqueItems":true
    },
    "tokens":{
        "$ref":"#/definitions/tokens"
    },
    "before":{
        "$ref":"#/definitions/row"
    },
    "after":{
        "$ref":"#/definitions/row"
    }
},
"required":[
    "table",
    "op_type",
    "op_ts",
```

```
        "current_ts",
        "pos"
    ],
    "additionalProperties":false
}
```

The following JSON schema example is for the JSON object listed in Sample Flattened Operation Modeled JSON Messages (page 17-38).

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"QASOURCE.TCUSTORD",
    "description":"JSON schema for table QASOURCE.TCUSTORD",
    "definitions":{
        "tokens":{
            "type":"object",
            "description":"Token keys and values are free form key value pairs.",
            "properties":{
            },
            "additionalProperties":true
        }
    },
    "type":"object",
    "properties":{
        "table":{
            "description":"The fully qualified table name",
            "type":"string"
        },
        "op_type":{
            "description":"The operation type",
            "type":"string"
        },
        "op_ts":{
            "description":"The operation timestamp",
            "type":"string"
        },
        "current_ts":{
            "description":"The current processing timestamp",
            "type":"string"
        },
        "pos":{
            "description":"The position of the operation in the data source",
            "type":"string"
        },
        "primary_keys":{
            "description":"Array of the primary key column names.",
            "type":"array",
            "items":{
                "type":"string"
            },
            "minItems":0,
            "uniqueItems":true
        },
        "tokens":{
            "$ref":"#/definitions/tokens"
        },
        "before.CUST_CODE":{
            "type":[
                "string",
                "null"
```

```
        ]
    },
    "before.ORDER_DATE":{
        "type":[
            "string",
            "null"
        ]
    },
    "before.PRODUCT_CODE":{
        "type":[
            "string",
            "null"
        ]
    },
    "before.ORDER_ID":{
        "type":[
            "number",
            "null"
        ]
    },
    "before.PRODUCT_PRICE":{
        "type":[
            "number",
            "null"
        ]
    },
    "before.PRODUCT_AMOUNT":{
        "type":[
            "integer",
            "null"
        ]
    },
    "before.TRANSACTION_ID":{
        "type":[
            "number",
            "null"
        ]
    },
    "after.CUST_CODE":{
        "type":[
            "string",
            "null"
        ]
    },
    "after.ORDER_DATE":{
        "type":[
            "string",
            "null"
        ]
    },
    "after.PRODUCT_CODE":{
        "type":[
            "string",
            "null"
        ]
    },
    "after.ORDER_ID":{
        "type":[
            "number",
```

```
                "null"
            ]
        },
        "after.PRODUCT_PRICE":{
            "type":[
                "number",
                "null"
            ]
        },
        "after.PRODUCT_AMOUNT":{
            "type":[
                "integer",
                "null"
            ]
        },
        "after.TRANSACTION_ID":{
            "type":[
                "number",
                "null"
            ]
        }
    },
    "required":[
        "table",
        "op_type",
        "op_ts",
        "current_ts",
        "pos"
    ],
    "additionalProperties":false
}
```

The following JSON schema example is for the JSON object listed in Sample Row Modeled JSON Messages (page 17-39).

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"QASOURCE.TCUSTORD",
    "description":"JSON schema for table QASOURCE.TCUSTORD",
    "definitions":{
        "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"
    },
    "primary_keys":{
        "description":"Array of the primary key column names.",
        "type":"array",
        "items":{
            "type":"string"
        },
        "minItems":0,
        "uniqueItems":true
    },
    "tokens":{
        "$ref":"#/definitions/tokens"
    },
    "CUST_CODE":{
        "type":[
            "string",
            "null"
        ]
    },
    "ORDER_DATE":{
        "type":[
            "string",
            "null"
        ]
    },
    "PRODUCT_CODE":{
        "type":[
            "string",
            "null"
        ]
    },
    "ORDER_ID":{
        "type":[
            "number",
            "null"
        ]
    },
    "PRODUCT_PRICE":{
        "type":[
            "number",
            "null"
        ]
    },
    "PRODUCT_AMOUNT":{
        "type":[
            "integer",
            "null"
        ]
    },
    "TRANSACTION_ID":{
```

```
                "type":[
                    "number",
                    "null"
                ]
            }
        },
        "required":[
            "table",
            "op_type",
            "op_ts",
            "current_ts",
            "pos"
        ],
        "additionalProperties":false
    }
```

17.3.6 JSON Formatter Configuration Properties

Table 17-11 JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format`	Optional	`json` | `json_row`	None	Controls whether the generated JSON output messages are operation modeled or row modeled. Set to `json` for operation modeled or`json_row` for row modeled.
`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.prettyPrint`	Optional	`true` | `false`	`false`	Controls the output format of the JSON data. True formats the data with white space for easy reading. False generates more compact output that is difficult to read..
`gg.handler.name.format.jsonDelimiter`	Optional	Any string	`""` (no value)	Inserts a delimiter between generated JSONs so that they can be more easily parsed in 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 schemahelp indicate what the JSON documents look like and can be used for a validating JSON parse.

Table 17-11 (Cont.) JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`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. When false, the formatter attempts to map Oracle GoldenGate types to the corresponding JSON type. When true, all data is 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 creates a schema with a timestamp in the schema directory on the local file system every time a new schema is created. True enables schema versioning. False disables schema versioning.
`gg.handler.name.format.iso8601Format`	Optional	`true` | `false`	`true`	Controls the format of the current timestamp. The default is the ISO 8601 format. A setting of `false` removes the "`T`" between the date and time in the current timestamp, which outputs a single space(" ") instead.
`gg.handler.name.format.includePrimaryKeys`	Optional	`true` | `false`	`false`	When set to `true`, to include an array of the primary key column names from the source table in the JSON output.
`gg.handler.name.format.flatten`	Optional	`true` | `false`	`false`	Controls sending flattened JSON formatted data to the target entity. Must be set to `true` for the flatten Delimiter property to work. This property is applicable only to Operation Formatted JSON (`gg.handler.name.format=json`).
`gg.handler.name.format.flattenDelimiter`	Optional	Any legal character or character string for a JSON field name.	`.`	Controls the delimiter for concatenated JSON element names. This property supports `CDATA[]` wrapping to preserve whitespace. It is only relevant when `gg.handler.name.format.flatten` is set to `true`.

Table 17-11 (Cont.) JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.beforeObjectName`	Optional	Any legal character or character string for a JSON field name.	Any legal JSON attribute name.	Allows you to set whether the JSON element-before, that contains the change column values, can be renamed. This property is only applicable to Operation Formatted JSON (`gg.handler.name.format=json`).
`gg.handler.name.format.afterObjectName`	Optional	Any legal character or character string for a JSON field name.	Any legal JSON attribute name.	Allows you to set whether the JSON element, that contains the after-change column values, can be renamed. This property is only applicable to Operation Formatted JSON (`gg.handler.name.format=json`).
`gg.handler.name.format.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Specifies how the formatter handles update operations that change a primary key. Primary key operations can be problematic for the JSON formatter and require special consideration by you. You can only use this property in conjunction with the row modeled JSON output messages. This property is only applicable to Row Formatted JSON (`gg.handler.name.format=json_row`). • `abend` : indicates that the process terminates. • `update`: the process handles the operation as a normal update. • `delete` or `insert`: the process handles the operation as a delete and an insert. Full supplemental logging must be enabled. Without full before and after row images, the insert data will be incomplete.
`gg.handler.name.format.omitNullValues`	Optional	`true` | `false`	`true`	Set to `false` to omit fields that have null values from being included in the generated JSON output.

17.3.7 Review a Sample Configuration

The following is a sample configuration for the JSON Formatter in 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.treatAllColumnsAsStrings=false
```

17.3.8 Metadata Change Events

Metadata change events are handled at runtime. When metadata is changed in a table, the JSON schema is regenerated the next time an operation for the table is encountered. The content of created JSON messages changes to reflect the metadata change. For example, if an additional column is added, the new column is included in created JSON messages after the metadata change event.

17.3.9 Setting Metacolumn Output

The following are the configurable values for the Delimited Text Format metacolumns template property that controls metacolumn output.

Table 17-12 Metacolumns Template Property

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.metaColumnsTemplate`	Optional	`${alltokens} | ${token} | ${env} | ${sys} | ${javaprop} | ${optype} | ${position} | ${timestamp} | ${catalog} | ${schema} | ${table} | ${objectname} | ${csn} | ${xid} | ${currenttimestamp} | ${opseqno} | ${timestampmicro} | ${currenttimestampmicro}`	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: `insertOpKey | updateOpKey | deleteOpKey | truncateOpKey | includeTableName | includeOpTimestamp | includeOpType | includePosition | includeCurrentTimestamp, useIso8601Format` It is a comma-delimited string consisting of one or more templated values that represent the template.

Explanation of the Metacolumn Keywords

${alltokens}
All of the Oracle GoldenGate tokens.

${token}
The value of a specific Oracle GoldenGate token. The token key should follow token key should follow the token using the period (.) operator. For example:

```
${token.MYTOKEN}
```

${sys}
A system environmental variable. The variable name should follow `sys` using the period (.) operator. For example:

```
${sys.MYVAR}
```

${env}
An Oracle GoldenGate environment variable. The variable name should follow `env` using the period (.) operator. For example:

```
${env.someVariable}
```

${javaprop}
A Java JVM variable. The variable name should follow `javaprop` using the period (.) operator. For example:

```
${javaprop.MYVAR}
```

${optype}
Operation Type

${position}
Record Position

${timestamp}
Record Timestamp

${catalog}
Catalog Name

${schema}
Schema Name

${table}
Table Name

${objectname}
The fully qualified table name.

${csn}
Source Commit Sequence Number

${xid}
Source Transaction ID

${currenttimestamp}
Current Timestamp

${opseqno}
Record sequence number within the transaction.

`${timestampmicro}`
Record timestamp (in microseconds after epoch).

`${currenttimestampmicro}`
Current timestamp (in microseconds after epoch).

17.3.10 JSON Primary Key Updates

When the JSON formatter is configured to model operation data, primary key updates require no special treatment and are treated like any other update. The before and after values reflect the change in the primary key.

When the JSON formatter is configured to model row data, primary key updates must be specially handled. The default behavior is to abend. However, by using the`gg.handler.name.format.pkUpdateHandling` configuration property, you can configure the JSON formatter to model row data to treat primary key updates as either a regular update or as delete and then insert operations. When you configure the formatter to handle primary key updates as delete and insert operations, Oracle recommends that you configure your replication stream to contain the complete before-image and after-image data for updates. Otherwise, the generated insert operation for a primary key update will be missing data for fields that did not change.

17.3.11 Integrating Oracle Stream Analytics

You can integrate Oracle GoldenGate for Big Data with Oracle Stream Analytics (OSA) by sending operation-modeled JSON messages to the Kafka Handler. This works only when the JSON formatter is configured to output operation-modeled JSON messages.

Because OSA requires flattened JSON objects, a new feature in the JSON formatter generates flattened JSONs. To use this feature, set the `gg.handler.name.format.flatten=false` to `true`. (The default setting is false). The following is an example of a flattened JSON file:

```
{
    "table":"QASOURCE.TCUSTMER",
    "op_type":"U",
    "op_ts":"2015-11-05 18:45:39.000000",
    "current_ts":"2016-06-22T13:38:45.335001",
    "pos":"00000000000000005100",
    "before.CUST_CODE":"ANN",
    "before.NAME":"ANN'S BOATS",
    "before.CITY":"SEATTLE",
    "before.STATE":"WA",
    "after.CUST_CODE":"ANN",
    "after.CITY":"NEW YORK",
    "after.STATE":"NY"
}
```

17.4 Using the Length Delimited Value Formatter

The Length Delimited Value (LDV) Formatter is a row-based formatter. It formats database operations from the source trail file into a length delimited value output. Each insert, update, delete, or truncate operation from the source trail is formatted into an individual length delimited message. With length delimited, there are no field delimiters. The fields are variable in size based on the data.

By default, the length delimited maps these column value states into the length delimited value output. Column values for an operation from the source trail file can have one of three states:

- Column has a value —The column value is output with the prefix indicator `P`.
- Column value is NULL —The default output value is `N`. The output for the case of a `NULL` column value is configurable.
- Column value is missing - The default output value is `M`. The output for the case of a missing column value is configurable.

Topics:

- Formatting Message Details (page 17-53)
- Sample Formatted Messages (page 17-53)
- LDV Formatter Configuration Properties (page 17-54)
- Additional Considerations (page 17-57)

17.4.1 Formatting Message Details

The default format for output of data is the following:

First is the row Length followed by metadata:

```
<ROW LENGTH><PRESENT INDICATOR><FIELD LENGTH><OPERATION TYPE><PRESENT
INDICATOR><FIELD LENGTH><FULLY QUALIFIED TABLE NAME><PRESENT INDICATOR><FIELD
LENGTH><OPERATION TIMESTAMP><PRESENT INDICATOR><FIELD LENGTH><CURRENT
TIMESTAMP><PRESENT INDICATOR><FIELD LENGTH><TRAIL POSITION><PRESENT
INDICATOR><FIELD LENGTH><TOKENS>
```

Or

```
<ROW LENGTH><FIELD LENGTH><FULLY QUALIFIED TABLE NAME><FIELD LENGTH><OPERATION
TIMESTAMP><FIELD LENGTH><CURRENT TIMESTAMP><FIELD LENGTH><TRAIL POSITION><FIELD
LENGTH><TOKENS>
```

Next is the row data:

```
<PRESENT INDICATOR><FIELD LENGTH><COLUMN 1 VALUE><PRESENT INDICATOR><FIELD
LENGTH><COLUMN N VALUE>
```

17.4.2 Sample Formatted Messages

Insert Message:

```
0133P01IP161446749136000000P161529311765024000P262015-11-05
18:45:36.000000P04WILLP191994-09-30 15:33:00P03CARP03144P0817520.00P013P03100
```

Update Message

```
0133P01UP161446749139000000P161529311765035000P262015-11-05
18:45:39.000000P04BILLP191995-12-31 15:00:00P03CARP03765P0814000.00P013P03100
```

Delete Message

```
0136P01DP161446749139000000P161529311765038000P262015-11-05
18:45:39.000000P04DAVEP191993-11-03
07:51:35P05PLANEP03600P09135000.00P012P03200
```

17.4.3 LDV Formatter Configuration Properties

Table 17-13 LDV Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.`*`name`*`.format.`*`binaryLengthMode`	Optional	`true` | `false`	`false`	The output can be controlled to display the field or record length in either binary or ASCII format. If set to `true`, the record or field length is represented in binary format else in ASCII.
`gg.handler.`*`name`*`.format.recordLength`	Optional	`4` | `8`	`true`	Set to `true`, the record length is represented using either a 4 or 8–byte big Endian integer. Set to `false`, the string representation of the record length with padded value with configured length of 4 or 8 is used.
`gg.handler.`*`name`*`.format.fieldLength`	Optional	`2` | `4`	`true`	Set to `true`, the record length is represented using either a 2 or 4-byte big Endian integer. Set to `false`, the string representation of the record length with padded value with configured length of 2 or 4 is used.
`gg.handler.`*`name`*`.format.format`	Optional	`true` | `false`	`true`	Use to configure the `P`indicator with MetaColumn. Set to `false`, enables the indicator `P` before the MetaColumns. If set to `true`, disables the indicator.
`gg.handler.`*`name`*`.format.presentValue`	Optional	Any string	`P`	Use to configure what is included in the output when a column value is present. This value supports CDATA[] wrapping.
`gg.handler.`*`name`*`.format.missingValue`	Optional	Any string	`M`	Use to configure what is included in the output when a missing value is present. This value supports CDATA[] wrapping.
`gg.handler.`*`name`*`.format.nullValue`	Optional	Any string	`N`	Use to configure what is included in the output when a NULL value is present. This value supports CDATA[] wrapping.

Table 17-13 (Cont.) LDV Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.metaColumnsTemplate`	Optional	`${alltokens}`, `${token}`, `${env}`, `${sys}`, `${javaprop}`, `${optype}`, `${position}`, `${timestamp}`, `${catalog}`, `${schema}`, `${table}`, `${objectname}`, `${csn}`, `${xid}`, `${currenttimestamp}`, `${opseqno}`, `${timestampmicro}`, `${currenttimestampmicro}`	None	Use to configure the current meta column information in a simple manner and removes the explicit need of `insertOpKey`, `updateOpKey`, `deleteOpKey`, `truncateOpKey`, `includeTableName`, `includeOpTimestamp`, `includeOpType`, `includePosition`, `includeCurrentTimestamp` and `useIso8601Format`. A comma-delimited string consisting of one or more templated values represents the template. This example produces a list of meta columns: `${optype}, ${token.ROWID},${sys.username},${currenttimestamp}` The valid properties are as follows: AllTokens – to display All Tokens Token – to display Specific Token SysEnv – to display System Environment Variable GGEnv – to display GG Environment Property JavaProp – to display Java Property OpType – to display Operation Type Position – to display Record Position TimeStamp – to display Record Timestamp Catalog – to display Catalog Name Schema – to display Schema Name Table – to display Table Name ObjectName – to display Fully Qualified Object Name CSN – to display Source Commit Sequence Number XID – to display Source Transaction ID CurrentTimeStamp – to display The current timestamp OpSeqno – to display Record sequence number within transaction TimeStampMicro – to display Record timestamp (in microseconds after epoch) CurrentTimeStampMicro – to display Current timestamp (in microseconds after epoch)

Table 17-13 (Cont.) LDV Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.pkUpdateHandling`	Optional	`abend` | `update` | `delete-insert`	`abend`	Specifies how the formatter handles 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 handles this as a delete and an insert. Full supplemental logging must be enabled for this to work. Without full before and after row images, the insert data will be incomplete.
`gg.handler.name.format.encoding`	Optional	Any encoding name or alias supported by Java.	The native system encoding of the machine hosting the Oracle GoldenGate process.	Use to set the output encoding for character data and columns.

Review a Sample Configuration

```
#The LDV Handler
gg.handler.filewriter.format=binary
gg.handler.filewriter.format.binaryLengthMode=false
gg.handler.filewriter.format.recordLength=4
gg.handler.filewriter.format.fieldLength=2
gg.handler.filewriter.format.legacyFormat=false
gg.handler.filewriter.format.presentValue=CDATA[P]
gg.handler.filewriter.format.missingValue=CDATA[M]
gg.handler.filewriter.format.nullValue=CDATA[N]
gg.handler.filewriter.format.metaColumnsTemplate=${optype},${timestampmicro},$
{currenttimestampmicro},${timestamp}
gg.handler.filewriter.format.pkUpdateHandling=abend
```

17.4.4 Additional 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. Data is not changed in Big Data applications, it is simply appended to existing data. The current state of a given row becomes a consolidation of all of the existing operations for that row in the HDFS system.

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 Length Delimited Value Formatter provides specialized handling for primary keys that is configurable to you. These are the configurable behaviors:

Table 17-14 Primary Key Update Behaviors

Value	Description
Abend	The default behavior is that the length delimited value formatter will abend in the case of a primary key update.
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 do not contain all of the data for all of the columns for a full representation of the row data in the Big Data application.

Consolidating Data

Big Data applications simply append data to the underlying storage. Typically, analytic tools spawn map reduce programs that traverse the data files and consolidate all the operations for a given row into a single output. It is important to have an indicator of the order of operations. The Length Delimited Value 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.

17.5 Using Operation-Based versus Row-Based Formatting

The Oracle GoldenGate for Big Data formatters include operation-based and row-based formatters.

The operation-based formatters 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), because a new row is being added to the source database. Update operations provide both before-change and after-change data that shows how existing row data is modified. Delete operations only provide before-change data to identify 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-change and after-change images.

The row-based formatters model the row data as it exists after the operation data is applied. Row-based formatters contain only a single image of the data. The following sections describe what data is displayed for both the operation-based and the row-based formatters.

Topics:

- Operation Formatters (page 17-58)
- Row Formatters (page 17-58)
- Table Row or Column Value States (page 17-58)

17.5.1 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-image and after-image data is output.
- Delete operation: Before-image data is output. After-image data is null.
- Truncate operation: Both before-image and after-image data is null.

17.5.2 Row Formatters

The formatters that support row-based formatting are 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: The table name is provided, but both before-image 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.

17.5.3 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 data has a value, or it is null. However; when data is transferred to the Oracle GoldenGate trail file by the Oracle GoldenGate capture process, the data can have three possible states: it can have a value, it can be null, or it can be missing.

For an insert operation, the after-image contains data for all column values regardless of whether the data 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 only the primary key values and the values of the columns that changed are required to modify the data in the target database. In addition, only the primary key values are required to delete the row from the target database. Therefore, even though values are present in the source database, the values may be missing in the source trail file. Because data in the source trail file may have three states, the Plugable Formatters must also be able to represent data in all three states.

Because the row and column data in the Oracle GoldenGate trail file has an important effect on a Big Data integration, it is important to understand the data that is required. Typically, you can control the data that is included for operations in the Oracle GoldenGate trail file. In an Oracle database, this data is controlled by the supplemental logging level. To understand how to control the row and column values that are included in the Oracle GoldenGate trail file, see the *Oracle GoldenGate* documentation for your source database implementation..

17.6 Using the XML Formatter

The XML Formatter formats before-image and after-image data from the source trail file 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.

Topics:

- Message Formatting Details (page 17-59)
- Sample XML Messages (page 17-60)
- XML Schema (page 17-63)
- XML Formatter Configuration Properties (page 17-64)
- Review a Sample Configuration (page 17-65)
- Metadata Change Events (page 17-65)
- Setting Metacolumn Output (page 17-65)
- Primary Key Updates (page 17-67)

17.6.1 Message Formatting Details

The XML formatted messages contain the following information:

Table 17-15 XML formatting details

Value	Description
`table`	The fully qualified table name.
`type`	The operation type.

Table 17-15 (Cont.) XML formatting details

Value	Description
`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.

17.6.2 Sample XML Messages

The following sections provide sample XML messages.

- Sample Insert Message (page 17-60)
- Sample Update Message (page 17-61)
- Sample Delete Message (page 17-61)
- Sample Truncate Message (page 17-62)

17.6.2.1 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>
```

17.6.2.2 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>
```

17.6.2.3 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>
```

17.6.2.4 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>
```

17.6.3 XML Schema

The XML Formatter does not generate an XML schema (XSD). The XSD applies to all messages generated by the XML Formatter. The following XSD defines the structure of the XML documents that are generated by the XML Formatter.

```
<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>
```

17.6.4 XML Formatter Configuration Properties

Table 17-16 XML Formatter Configuration Properties

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.
`gg.handler.`*`name`*`.format.encoding`	Optional	Any legal encoding name or alias supported by Java.	UTF-8 (the XML default)	The output encoding of generated XML documents.
`gg.handler.`*`name`*`.format.includeProlog`	Optional	`true` | `false`	`false`	Determines whether an XML prolog is included in generated XML documents. An XML prolog is optional for well-formed XML. An XML prolog resembles the following:`<?xml version='1.0' encoding='UTF-8'?>`

Table 17-16 (Cont.) XML Formatter Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
`gg.handler.name.format.iso8601Format`	Optional	`true` | `false`	`true`	Controls the format of the current timestamp in the XML message. The default adds a `T` between the date and time. Set to `false` to suppress the `T` between the date and time and instead include blank space.

17.6.5 Review a Sample Configuration

The following is a sample configuration for the XML Formatter in 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
```

17.6.6 Metadata Change Events

The XML Formatter seamlessly handles metadata change events. A metadata change event does not 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.

If the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events, the XML Formatter can take action when metadata changes. Changes in the metadata are reflected in messages after the change. For example, when a column is added, the new column data appears in XML messages for the table.

17.6.7 Setting Metacolumn Output

The following are the configurable values for the XML metacolumns template property that controls metacolumn output.

Table 17-17 Metacolumns Template Property

Properties	Required/ Optional	Legal Values	Default	Explanation
`gg.handler.name.format.metaColumnsTemplate`	Optional	`${alltokens} | ${token} | ${env} | ${sys} | ${javaprop} | ${optype} | ${position} | ${timestamp} | ${catalog} | ${schema} | ${table} | ${objectname} | ${csn} | ${xid} | ${currenttimestamp} | ${opseqno} | ${timestampmicro} | ${currenttimestampmicro}`	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: `insertOpKey | updateOpKey | deleteOpKey | truncateOpKey | includeTableName | includeOpTimestamp | includeOpType | includePosition | includeCurrentTimestamp, useIso8601Format` It is a comma-delimited string consisting of one or more templated values that represent the template.

Explanation of the Metacolumn Keywords

`${alltokens}`
All of the Oracle GoldenGate tokens.

`${token}`
The value of a specific Oracle GoldenGate token. The token key should follow token key should follow the token using the period (`.`) operator. For example:

```
${token.MYTOKEN}
```

`${sys}`
A system environmental variable. The variable name should follow `sys` using the period (`.`) operator. For example:

```
${sys.MYVAR}
```

${env}
An Oracle GoldenGate environment variable. The variable name should follow `env` using the period (`.`) operator. For example:

```
${env.someVariable}
```

${javaprop}
A Java JVM variable. The variable name should follow `javaprop` using the period (`.`) operator. For example:

```
${javaprop.MYVAR}
```

${optype}
Operation Type

${position}
Record Position

${timestamp}
Record Timestamp

${catalog}
Catalog Name

${schema}
Schema Name

${table}
Table Name

${objectname}
The fully qualified table name.

${csn}
Source Commit Sequence Number

${xid}
Source Transaction ID

${currenttimestamp}
Current Timestamp

${opseqno}
Record sequence number within the transaction.

${timestampmicro}
Record timestamp (in microseconds after epoch).

${currenttimestampmicro}
Current timestamp (in microseconds after epoch).

17.6.8 Primary Key Updates

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

18

Using Oracle GoldenGate Capture for Cassandra

Learn how to use Oracle GoldenGate capture (Extract) to get changes from Apache Cassandra databases.

Topics:

- Overview (page 18-1)
- Setting Up Cassandra Change Data Capture (page 18-2)
- Deduplication (page 18-3)
- Topology Changes (page 18-4)
- Data Availability in the CDC Logs (page 18-4)
- Using Extract Initial Load (page 18-4)
- Using Change Data Capture Extract (page 18-5)
- Replicating to RDMBS Targets (page 18-7)
- Partition Update or Insert of Static Columns (page 18-7)
- Partition Delete (page 18-8)
- Security and Authentication (page 18-8)
- Multiple Extract Support (page 18-9)
- CDC Configuration Reference (page 18-9)
- Troubleshooting (page 18-15)

18.1 Overview

Apache Cassandra is a NoSQL Database Management System designed to store large amounts of data. A Cassandra cluster configuration provides horizontal scaling and replication of data across multiple machines. It can provide high availability and eliminate a single point of failure by replicating data to multiple nodes within a Cassandra cluster. Apache Cassandra is open source and designed to run on low-cost commodity hardware.

Cassandra relaxes the axioms of a traditional relational database management systems (RDBMS) regarding atomicity, consistency, isolation, and durability. When considering implementing Cassandra, it is important to understand its differences from a traditional RDBMS and how those differences affect your specific use case.

Cassandra provides eventual consistency. Under the eventual consistency model, accessing the state of data for a specific row eventually returns the latest state of the data for that row as defined by the most recent change. However, there may be a latency period between the creation and modification of the state of a row and what is returned when the state of that row is queried. The benefit of eventual consistency is that the latency period is predicted based on your Cassandra configuration and the

level of work load that your Cassandra cluster is currently under, see http://cassandra.apache.org/.

Review the data type support, see About the Cassandra Data Types (page 3-2).

18.2 Setting Up Cassandra Change Data Capture

Prerequisites

- Apache Cassandra cluster must have at least one node up and running.
- Read and write access to CDC commit log files on every live node in the cluster is done through SFTP or NFS.
- Every node in the Cassandra cluster must have the `cdc_enabled` parameter set to `true` in the `cassandra.yaml` configuration file.
- Virtual nodes must be enabled on every Cassandra node by setting the `num_tokens` parameter in `cassandra.yaml` .
- You must download and provide the third party libraries listed in Cassandra Capture Client Dependencies (page B-1).
- New tables can be created with Change Data Capture (CDC) enabled using the `WITH CDC=true` clause in the `CREATE TABLE` command. For example:

  ```
  CREATE TABLE ks_demo_rep1.mytable (col1 int, col2 text, col3 text, col4 text,
  PRIMARY KEY (col1)) WITH cdc=true;
  ```

  You can enable CDC on existing tables as follows:

  ```
  ALTER TABLE ks_demo_rep1.mytable WITH cdc=true;
  ```

- Data Types (page 18-2)
- Cassandra Database Operations (page 18-3)

18.2.1 Data Types

Supported Cassandra Data Types

The following are the supported data types:

- `ASCII`
- `BIGINT`
- `BLOB`
- `BOOLEAN`
- `DATE`
- `DECIMAL`
- `DOUBLE`
- `DURATION`
- `FLOAT`
- `INET`
- `INT`

- SMALLINT
- TEXT
- TIME
- TIMESTAMP
- TIMEUUID
- TINYINT
- UUID
- VARCHAR
- VARINT

Unsupported Data Types

The following are the unsupported data types:

- COUNTER
- MAP
- SET
- LIST
- UDT (user defined type)
- TUPLE
- CUSTOM_TYPE

18.2.2 Cassandra Database Operations

Supported Operations

The following are the supported operations:

- INSERT
- UPDATE (Captured as INSERT)
- DELETE

Unsupported Operations

The TRUNCATE DDL (CREATE, ALTER, and DROP) operation is not supported. Because the Cassandra commit log files do not record any before images for the UPDATE or DELETE operations. The result is that the captured operations can never have a before image. Oracle GoldenGate features that rely on before image records, such as Conflict Detection and Resolution, are not available.

18.3 Deduplication

One of the features of a Cassandra cluster is its high availability. To support high availability, multiple redundant copies of table data are stored on different nodes in the cluster. Oracle GoldenGate for Big Data Cassandra Capture automatically filters out duplicate rows (**deduplicate**). Deduplication is active by default. Oracle recommends

using it if your data is captured and applied to targets where duplicate records are discouraged (for example RDBMS targets).

18.4 Topology Changes

Cassandra nodes can change their status (**topology change**) and the cluster can still be alive. Oracle GoldenGate for Big Data Cassandra Capture can detect the node status changes and react to these changes when applicable. The Cassandra capture process can detect the following events happening in the cluster:

- Node shutdown and boot.
- Node decommission and commission.
- New keyspace and table created.

Due to topology changes, if the capture process detects that an active producer node goes down, it tries to recover any missing rows from an available replica node. During this process, there is a possibility of data duplication for some rows. This is a transient data duplication due to the topology change. For more details about reacting to changes in topology, see Troubleshooting (page 18-15).

18.5 Data Availability in the CDC Logs

The Cassandra CDC API can only read data from commit log files in the CDC directory. There is a latency for the data in the active commit log directory to be archived (moved) to the CDC commit log directory.

The input data source for the Cassandra capture process is the CDC commit log directory. There could be delays for the data to be captured mainly due to the commit log files not yet visible to the capture process.

On a production cluster with a lot of activity, this latency is very minimal as the data is archived from the active commit log directory to the CDC commit log directory in the order of microseconds.

18.6 Using Extract Initial Load

Cassandra Extract supports the standard initial load capability to extract source table data to Oracle GoldenGate trail files.

Initial load for Cassandra can be performed to synchronize tables, either as a prerequisite step to replicating changes or as a standalone function.

Direct loading from a source Cassandra table to any target table is *not* supported.

Configuring the Initial Load

You need to add these parameters to your `GLOBALS` parameter file:

```
OGGSOURCE CASSANDRA
CLUSTERCONTACTPOINTS nodeadresses
```

For example, to write to a single trail file:

```
SOURCEISTABLE
SOURCEDB keyspace1, USERID user1, PASSWORD pass1
```

```
EXTFILE ./dirdat/load_data.dat, PURGE
TABLE keyspace1.table1;
```

Then you would run this command in GGSCI:

```
EXTRACT PARAMFILE ./dirprm/load.prm REPORTFILE ./dirrpt/load.rpt
```

If you want to write to multiple files, you could use:

```
EXTRACT load
SOURCEISTABLE
SOURCEDB keyspace1, USERID user1, PASSWORD pass1
EXTFILE ./dirdat/la, megabytes 2048, MAXFILES 999
TABLE keyspace1.table1;
```

> **Note:**
>
> Save the file with the name specified in the example (`load.prm`) into the `dirprm` directory.

Then you would run these commands in GGSCI:

```
ADD EXTRACT load, SOURCEISTABLE
START EXTRACT load
```

18.7 Using Change Data Capture Extract

Review the example `.prm` files from Oracle GoldenGate for Big Datainstallation directory under `$HOME/AdapterExamples/big-data/cassandracapture`.

1. When adding the Cassandra Extract trail, you need to use `EXTTRAIL` to create a local trail file.

   The Cassandra Extract trail file should not be configured with the `RMTTRAIL` option.

   ```
   ggsci> ADD EXTRACT groupname, TRANLOG
   ggsci> ADD EXTTRAIL trailprefix, EXTRACT groupname
   Example:
   ggsci> ADD EXTRACT cass, TRANLOG
   ggsci> ADD EXTTRAIL ./dirdat/z1, EXTRACT cass
   ```

2. To configure the Extract, see the example `.prm` files in the Oracle GoldenGate for Big Data installation directory in `$HOME/AdapterExamples/big-data/cassandracapture`.

3. Position the Extract.

   ```
   ggsci> ADD EXTRACT groupname, TRANLOG, BEGIN NOW
   ggsci> ADD EXTRACT groupname, TRANLOG, BEGIN 'yyyy-mm-dd hh:mm:ss'
   ggsci> ALTER EXTRACT groupname, BEGIN 'yyyy-mm-dd hh:mm:ss'
   ```

4. Manage the transaction data logging for the tables.

   ```
   ggsci> DBLOGIN SOURCEDB nodeaddress USERID userid PASSWORD password
   ggsci> ADD TRANDATA keyspace.tablename
   ggsci> INFO TRANDATA keyspace.tablename
   ggsci> DELETE TRANDATA keyspace.tablename
   ```

   Examples:

```
ggsci> DBLOGIN SOURCEDB 127.0.0.1
ggsci> INFO TRANDATA ks_demo_rep1.mytable
ggsci> INFO TRANDATA ks_demo_rep1.*
ggsci> INFO TRANDATA *.*
ggsci> INFO TRANDATA ks_demo_rep1."CamelCaseTab"
ggsci> ADD TRANDATA ks_demo_rep1.mytable
ggsci> DELETE TRANDATA ks_demo_rep1.mytable
```

5. Append the following line in the GLOBALS parameter file:

```
JVMBOOTOPTIONS -Dlogback.configurationFile=AdapterExamples/big-data/
cassandracapture/logback.xml
```

6. Configure the Extract and GLOBALS parameter files:

Apache Cassandra 3.11 SDK, compatible with Apache Cassandra 3.9, 3.10, 3.11

Extract parameter file:

```
EXTRACT groupname
TRANLOGOPTIONS CDCREADERSDKVERSION 3.11
TRANLOGOPTIONS CDCLOGDIRTEMPLATE /path/to/data/cdc_raw
SOURCEDB nodeaddress
VAM libggbigdata_vam.so
EXTTRAIL trailprefix
TABLE *.*;
```

GLOBALS parameter file:

```
OGGSOURCE CASSANDRA
CLUSTERCONTACTPOINTS nodeadresses
JVMCLASSPATH ggjava/ggjava.jar:/path/to/cassandra-driver-core/3.3.1/cassandra-
driver-core-3.3.1.jar:dirprm:/path/to/apache-cassandra-3.11.0/lib/*:/path/to/
gson/2.3/gson-2.3.jar:/path/to/jsch/0.1.54/jsch-0.1.54.jar:/path/to/commons-
lang3/3.5/commons-lang3-3.5.jar
```

Oracle recommends that you use the latest Cassandra 3.11 JAR files (`TRANLOGOPTIONS CDCREADERSDKVERSION 3.11` and `JVMCLASSPATH` configuration) for all supported Cassandra database versions.

Apache Cassandra 3.9 SDK

Extract parameter file:

```
EXTRACT groupname
TRANLOGOPTIONS CDCREADERSDKVERSION 3.9
TRANLOGOPTIONS CDCLOGDIRTEMPLATE /path/to/data/cdc_raw
SOURCEDB nodeaddress
VAM libggbigdata_vam.so
EXTTRAIL trailprefix
TABLE *.*;
```

GLOBALS parameter file:

```
OGGSOURCE CASSANDRA
CLUSTERCONTACTPOINTS nodeadresses
JVMCLASSPATH ggjava/ggjava.jar:/path/to/cassandra-driver-core/3.3.1/cassandra-
driver-core-3.3.1.jar:dirprm:/path/to/apache-cassandra-3.9/lib/*:/path/to/
gson/2.3/gson-2.3.jar:/path/to/jsch/0.1.54/jsch-0.1.54.jar:/path/to/commons-
lang3/3.5/commons-lang3-3.5.jar
```

18.8 Replicating to RDMBS Targets

You must take additional care when replicating source `UPDATE` operations from Cassandra trail files to RDMBS targets. Any source `UPDATE` operation appears as an `INSERT` record in the Oracle GoldenGate trail file. Replicat may abend when a source `UPDATE` operation is applied as an `INSERT` operation on the target database.

You have these options:

- `OVERRIDEDUPS`: If you expect that the source database is to contain mostly `INSERT` operations and very few `UPDATE` operations, then `OVERRIDEDUPS` is the recommended option. Replicat can recover from duplicate key errors while replicating the small number of the source `UPDATE` operations. See `OVERRIDEDUPS` \ `NOOVERRIDEDUPS`
- `UPDATEINSERTS` and `INSERTMISSINGUPDATES`: Use this configuration if the source database is expected to contain mostly `UPDATE` operations and very few `INSERT` operations. With this configuration, Replicat has fewer missing row errors to recover, which leads to better throughput. See `UPDATEINSERTS` | `NOUPDATEINSERTS` and `INSERTMISSINGUPDATES` | `NOINSERTMISSINGUPDATES`.
- No additional configuration is required if the target table can accept duplicate rows or you want to abend Replicat on duplicate rows.

If you configure Replicat to use `BATCHSQL`, there may be duplicate row or missing row errors in batch mode. Although there is a reduction in the Replicat throughput due to these errors, Replicat automatically recovers from these errors. If the source operations are mostly `INSERTS`, then `BATCHSQL` is a good option.

18.9 Partition Update or Insert of Static Columns

When the source Cassandra table has static columns, the static column values can be modified by skipping any clustering key columns that are in the table.

For example:

```
create table ks_demo_rep1.nls_staticcol
(
    teamname text,
    manager text static,
    location text static,
    membername text,
    nationality text,
    position text,
    PRIMARY KEY ((teamname), membername)
)
WITH cdc=true;
insert into ks_demo_rep1.nls_staticcol (teamname, manager, location) VALUES ('Red
Bull', 'Christian Horner', '<unknown>
```

The insert `CQL` is missing the clustering key `membername`. Such an operation is a partition insert.

Similarly, you could also update a static column with just the partition keys in the `WHERE` clause of the `CQL` that is a partition update operation. Cassandra Extract cannot write a `INSERT` or `UPDATE` operation into the trail with missing key columns. It abends on detecting a partition `INSERT` or `UPDATE` operation.

18.10 Partition Delete

A Cassandra table may have a primary key composed on one or more partition key columns and clustering key columns. When a `DELETE` operation is performed on a Cassandra table by skipping the clustering key columns from the `WHERE` clause, it results in a partition delete operation.

For example:

```
create table ks_demo_rep1.table1
(
 col1 ascii, col2 bigint, col3 boolean, col4 int,
 PRIMARY KEY((col1, col2), col4)
) with cdc=true;

delete from ks_demo_rep1.table1 where col1 = 'asciival' and col2 = 9876543210; /**
skipped clustering key column col4 **/
```

Cassandra Extract cannot write a `DELETE` operation into the trail with missing key columns and abends on detecting a partition `DELETE` operation.

18.11 Security and Authentication

- Cassandra Extract can connect to a Cassandra cluster using username and password based authentication and SSL authentication.
- Connection to Kerberos enabled Cassandra clusters is *not* supported in this release.
- Configuring SSL (page 18-8)

18.11.1 Configuring SSL

To enable SSL, add the SSL parameter to your `GLOBALS` file or Extract parameter file. Additionally, a separate configuration is required for the Java and CPP drivers, see CDC Configuration Reference (page 18-9).

SSL configuration for Java driver

```
JVMBOOTOPTIONS -
Djavax.net.ssl.trustStore=/path/to/SSL/truststore.file -
Djavax.net.ssl.trustStorePassword=password -
Djavax.net.ssl.keyStore=/path/to/SSL/keystore.file -
Djavax.net.ssl.keyStorePassword=password
```

The keystore and truststore certificates can be generated using these instructions:

https://docs.datastax.com/en/cassandra/2.1/cassandra/security/secureSSLCertificates_t.html

SSL configuration for Cassandra CPP driver

To operate with an SSL configuration, you have to add the following parameter in the Oracle GoldenGate `GLOBALS` file or Extract parameter file:

```
CPPDRIVEROPTIONS SSL PEMPUBLICKEYFILE /path/to/PEM/formatted/public/key/file/
cassandra.pem CPPDRIVEROPTIONS SSL PEERCERTVERIFICATIONFLAG 0
```

This configuration is required to connect to a Cassandra cluster with SSL enabled. Additionally, you need to add these settings to your `cassandra.yaml` file:

```
client_encryption_options:
    enabled: true
    # If enabled and optional is set to true encrypted and unencrypted connections
are handled.
    optional: false
    keystore: /path/to/keystore
    keystore_password: password
    require_client_auth: false
```

The PEM formatted certificates can be generated using these instructions:

https://docs.datastax.com/en/developer/cpp-driver/2.8/topics/security/ssl/

18.12 Multiple Extract Support

Multiple Extract groups in a single Oracle GoldenGate for Big Data installation can be configured to connect to the same Cassandra cluster.

To run multiple Extract groups:

1. One (and only one) Extract group can be configured to move the commit log files in the `cdc_raw` directory on the Cassandra nodes to a staging directory. The `movecommitlogstostagingdir` parameter is enabled by default and no additional configuration is required for this Extract group.
2. All the other Extract groups should be configured with the `nomovecommitlogstostagingdir` parameter in the Extract parameter (`.prm`) file.

18.13 CDC Configuration Reference

The following properties are used with Cassandra change data capture.

Properties	Required/ Optional	Location	Default	Explanation
`DBOPTIONS ENABLECASSANDRACPPDRIVERTRACE true`	Optional	Extract parameter (`.prm`) file.	`false`	Use only during initial load process. When set to `true`, the Cassandra driver logs all the API calls to a `driver.log` file. This file is created in the Oracle GoldenGate for Big Data installation directory. This is useful for debugging.

Properties	Required/ Optional	Location	Default	Explanation
`DBOPTIONS FETCHBATCHSIZE` *`number`*	Optional	Extract parameter (`.prm`) file.	`1000` Minimum is `1` Maximum is `100000`	Use only during initial load process. Specifies the number of rows of data the driver attempts to fetch on each request submitted to the database server. The parameter value should be lower than the database configuration parameter, `tombstone_warn_threshold`, in the database configuration file, `cassandra.yaml`. Otherwise the initial load process might fail. Oracle recommends that you set this parameter value to 5000 for initial load Extract optimum performance.
`TRANLOGOPTIONS CDCLOGDIRTEMPLATE` *`path`*	Required	Extract parameter (`.prm`) file.	None	The CDC commit log directory path template. The template can optionally have the `$nodeAddress` meta field that is resolved to the respective node address.
`TRANLOGOPTIONS SFTP` *`options`*	Optional	Extract parameter (`.prm`) file.	None	The secure file transfer protocol (SFTP) connection details to pull and transfer the commit log files. You can use one or more of these options: **`USER`** ***`user`*** The SFTP user name. **`PASSWORD`** ***`password`*** The SFTP password. **`KNOWNHOSTSFILE`** ***`file`*** The location of the Secure Shell (SSH)known hosts file. **`LANDINGDIR`** ***`dir`*** The SFTP landing directory for the commit log files on the local machine. **`PRIVATEKEY`** ***`file`*** The SSH private key file. **`PASSPHRASE`** ***`password`*** The SSH private key pass phrase. **`PORTNUMBER`** ***`portnumber`*** The SSH port number.

Properties	Required/ Optional	Location	Default	Explanation
`CLUSTERCONTACTPOINTS` *`nodes`* `USER` *`dbuser`* `PASSWORD` *`dbpassword`*	Optional	`GLOBALS` parameter file	`127.0.0.1`	A comma separated list of nodes to be used for a connection to the Cassandra cluster. You should provide at least one node address. The parameter options are: **`USER`** ***`dbuser`*** No default Optional The user name to use when connecting to the database. **`PASSWORD`** ***`dbpassword`*** No default Required when `USER` is used. The user password to use when connecting to the database. **`PORT`** ***`<port number`*** No default Optional The port to use when connecting to the database.
`TRANLOGOPTIONS CDCREADERSDKVERSION` *`version`*	Optional	Extract parameter (`.prm`) file.	`3.11`	The SDK Version for the CDC reader capture API.
`ABENDONMISSEDRECORD | NOABENDONMISSEDRECORD`	Optional	Extract parameter (`.prm`) file.	`true`	When set to `true` and the possibility of a missing record is found, the process stops with the diagnostic information. This is generally detected when a node goes down and the CDC reader doesn't find a replica node with a matching last record from the dead node. You can set this parameter to `false` to continue processing. A warning message is logged about the scenario.
`TRANLOGOPTIONS CLEANUPCDCCOMMITLOGS`	Optional	Extract parameter (`.prm`) file.	`false`	Purge CDC commit log files post extract processing. When the value is set to `false`, the CDC commit log files are moved to the `cdc_raw_processed` directory.
`JVMBOOTOPTIONS` *`jvm_options`*	Optional	`GLOBALS` parameter file	None	The boot options for the Java Virtual Machine. Multiple options are delimited by a space character.
`JVMCLASSPATH` *`classpath`*	Required	`GLOBALS` parameter file	None	The classpath for the Java Virtual Machine. You can include an asterisk (`*`) wildcard to match all JAR files in any directory. Multiple paths should be delimited with a colon (`:`) character.

`OGGSOURCE` *`source`*	Required		None	The source database for CDC capture or database queries. The valid value is `CASSANDRA`.
`SOURCEDB` *`nodeaddress`* `USERID` *`dbuser`* `PASSWORD` *`dbpassword`*	Required	Extract parameter (`.prm`) file.	None	A single Cassandra node address that is used for a connection to the Cassandra cluster and to query the metadata for the captured tables. **`USER` *`dbuser`*** No default Optional The user name to use when connecting to the database. **`PASSWORD` *`dbpassword`*** No default Required when `USER` is used. The user password to use when connecting to the database.
`ABENDONUPDATERECORDWITHMISSINGKEYS` | `NOABENDONUPDATERECORDWITHMISSINGKEYS`	Optional	Extract parameter (`.prm`) file.	`true`	If this value is `true`, anytime an `UPDATE` operation record with missing key columns is found, the process stops with the diagnostic information. You can set this property to `false` to continue processing and write this record to the trail file. A warning message is logged about the scenario. This operation is a partition update, see Partition Update or Insert of Static Columns (page 18-7).
`ABENDONDELETERECORDWITHMISSINGKEYS` | `NOABENDONDELETERECORDWITHMISSINGKEYS`	Optional	Extract parameter (`.prm`) file.	`true`	If this value is `true`, anytime an `DELETE` operation record with missing key columns is found, the process stops with the diagnostic information. You can set this property to `false` to continue processing and write this record to the trail file. A warning message is logged about the scenario. This operation is a partition update, see Partition Delete (page 18-8).
`MOVECOMMITLOGSTOSTAGINGDIR` | `NOMOVECOMMITLOGSTOSTAGINGDIR`	Optional	Extract parameter (`.prm`) file.	`true`	Enabled by default and this instructs the Extract group to move the commit log files in the `cdc_raw` directory on the Cassandra nodes to a staging directory for the commit log files. Only one Extract group can have `movecommitlogstostagingdir` enabled, and all the other Extract groups disable this by specifying `nomovecommitlogstostagingdir`.

`SSL`	Optional	GLOBALS or Extract parameter (`.prm`) file.	`false`	Use for basic SSL support during connection. Additional JSSE configuration through Java System properties is expected when enabling this. **Note:** The following SSL properties are in `CPPDRIVEROPTIONS SSL` so this keyword must be added to any other SSL property to work.
`CPPDRIVEROPTIONS SSL PEMPUBLICKEYFILE` *`cassadra.pem`*	Optional	GLOBALS or Extract parameter (`.prm`) file. String that indicates the absolute path with fully qualified name. This file is must for the SSL connection.	None, unless the `PEMPUBLICKEYFILE` property is specified, then you must specify a value.	Indicates that it is PEM formatted public key file used to verify the peer's certificate. This property is needed for one-way handshake or basic SSL connection.
`CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH | DISABLECLIENTAUTH`	Optional	GLOBALS or Extract parameter (`.prm`) file.	`false`	Enabled indicates a two-way SSL encryption between client and server. It is required to authenticate both the client and the server through PEM formatted certificates. This property also needs the `pemclientpublickeyfile` and `pemclientprivatekeyfile` properties to be set. The `pemclientprivatekeypasswd` property must be configured if the client private key is password protected. Setting this property to `false` disables client authentication for two-way handshake.

`CPPDRIVEROPTIONS SSL PEMCLIENTPUBLICKEYFILE` *`public.pem`*	Optional	GLOBALS or Extract parameter (`.prm`) file. String that indicates the absolute path with fully qualified name. This file is must for the SSL connection.	None, unless the `PEMCLIENTPUBLICKEYFILE` property is specified, then you must specify a value.	Use for a PEM formatted public key file name used to verify the client's certificate. This is must if you are using `CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH` or for two-way handshake.
`CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATEKEYFILE` *`public.pem`*	Optional	GLOBALS or Extract parameter (`.prm`) file. String that indicates the absolute path with fully qualified name. This file is must for the SSL connection.	None, unless the `PEMCLIENTPRIVATEKEYFILE` property is specified, then you must specify a value.	Use for a PEM formatted private key file name used to verify the client's certificate. This is must if you are using `CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH` or for two-way handshake.

`CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATEKEYPASSWD` *`privateKeyPasswd`*	Optional	GLOBALS or Extract parameter (`.prm`) file. A string	None, unless the `PEMCLIENTPRIVATEKEYPASSWD` property is specified, then you must specify a value.	Sets the password for the PEM formatted private key file used to verify the client's certificate. This is must if the private key file is protected with the password.
`CPPDRIVEROPTIONS SSL PEERCERTVERIFICATIONFLAG` *`value`*	Optional	GLOBALS or Extract parameter (`.prm`) file. An integer	`0`	Sets the verification required on the peer's certificate. The range is 0–4: 0–Disable certificate identity verification. 1–Verify the peer certificate 2–Verify the peer identity 3– Not used so it is similar to disable certificate identity verification. 4 –Verify the peer identity by its domain name
`CPPDRIVEROPTIONS SSL ENABLEREVERSEDNS`	Optional	GLOBALS or Extract parameter (`.prm`) file.	`false`	Enables retrieving host name for IP addresses using reverse IP lookup.

18.14 Troubleshooting

No data captured by the Cassandra Extract process.

- The Cassandra database has not flushed the data from the active commit log files to the CDC commit log files. The flush is dependent on the load of the Cassandra cluster.
- The Cassandra Extract captures data from the CDC commit log files only.
- Check the CDC property of the source table. The CDC property of the source table should be set to `true`.
- Data is not captured if the `TRANLOGOPTIONS CDCREADERSDKVERSION` 3.9 parameter is in use and the `JVMCLASSPATH` is configured to point to Cassandra 3.10 or 3.11 JAR files.

Error: OGG-01115 Function getInstance not implemented.

- The following line is missing from the `GLOBALS` file.

  ```
  OGGSOURCE CASSANDRA
  ```

- The `GLOBALS` file is missing from the Oracle GoldenGate directory.

Error: Unable to connect to Cassandra cluster, Exception: com.datastax.driver.core.exceptions.NoHostAvailableException

This indicates that the connection to the Cassandra cluster was unsuccessful.

Check the following parameters:

`CLUSTERCONTACTPOINTS`

Error: Exception in thread "main" java.lang.NoClassDefFoundError: oracle/goldengate/capture/cassandra/CassandraCDCProcessManager

Check the `JVMCLASSPATH` parameter in the `GLOBALS` file.

Error: oracle.goldengate.util.Util - Unable to invoke method while constructing object. Unable to create object of class "oracle.goldengate.capture.cassandracapture311.SchemaLoader3DOT11" Caused by: java.lang.NoSuchMethodError: org.apache.cassandra.config.DatabaseDescriptor.clientInitialization()V

There is a mismatch in the Cassandra SDK version configuration. The `TRANLOGOPTIONS CDCREADERSDKVERSION 3.11` parameter is in use and the `JVMCLASSPATH` may have the Cassandra 3.9 JAR file path.

Error: OGG-25171 Trail file '/path/to/trail/gg' is remote. Only local trail allowed for this extract.

A Cassandra Extract should only be configured to write to local trail files. When adding trail files for Cassandra Extract, use the `EXTTRAIL` option. For example:

```
ADD EXTTRAIL ./dirdat/z1, EXTRACT cass
```

Errors: OGG-868 *error message* or OGG-4510 *error message*

The cause could be any of the following:

- Unknown user or invalid password
- Unknown node address
- Insufficient memory

Another cause could be that the connection to the Cassandra database is broken. The *error message* indicates the database error that has occurred.

Error: OGG-251712 Keyspace keyspacename does not exist in the database.

The issue could be due to these conditions:

- During the Extract initial load process, you may have deleted the `KEYSPACE` *`keyspacename`* from the Cassandra database.
- The `KEYSPACE` *`keyspacename`* does not exist in the Cassandra database.

Error: OGG-25175 Unexpected error while fetching row.

This can occur if the connection to the Cassandra database is broken during initial load process.

Error: "Server-side warning: Read 915936 live rows and 12823104 tombstone cells for query SELECT * FROM *keyspace.table*(see tombstone_warn_threshold)".

When the value of the initial load `DBOPTIONS FETCHBATCHSIZE` parameter is greater than the Cassandra database configuration parameter,`tombstone_warn_threshold`, this is likely to occur.

Increase the value of `tombstone_warn_threshold` or reduce the `DBOPTIONS FETCHBATCHSIZE` value to get around this issue.

Duplicate records in the Cassandra Extract trail.

Internal tests on a multi-node Cassandra cluster have revealed that there is a possibility of duplicate records in the Cassandra CDC commit log files. The duplication in the Cassandra commit log files is more common when there is heavy write parallelism, write errors on nodes, and multiple retry attempts on the Cassandra nodes. In these cases, it is expected that Cassandra trail file will have duplicate records.

JSchException or SftpException in the Extract Report File

Verify that the SFTP credentials (`user`, `password`, and `privatekey`) are correct. Check that the SFTP user has read and write permissions for the `cdc_raw` directory on each of the nodes in the Cassandra cluster.

ERROR o.g.c.c.CassandraCDCProcessManager - Exception during creation of CDC staging directory [{}]java.nio.file.AccessDeniedException

The Extract process does not have permission to create CDC commit log staging directory. For example, if the `cdc_raw` commit log directory is `/path/to/cassandra/home/data/cdc_raw`, then the staging directory would be `/path/to/cassandra/home/data/cdc_raw/../cdc_raw_staged`.

Extract report file shows a lot of DEBUG log statements

On production system, you do not need to enable debug logging. To use `INFO` level logging, make sure that the `GLOBALS` file includes this parameter:

```
JVMBOOTOPTIONS -Dlogback.configurationFile=AdapterExamples/big-data/cassandracapture/logback.xml
```

To enable SSL in Oracle Golden Gate Cassandra Extract you have to enable SSL in the GLOBALS file or in the Extract Parameter file.

If SSL Keyword is missing, then Extract assumes that you wanted to connect without SSL. So if the `Cassandra.yaml` file has an SSL configuration entry, then the connection fails.

SSL is enabled and it is one-way handshake

You must specify the `CPPDRIVEROPTIONS SSL PEMPUBLICKEYFILE` *`/scratch/testcassandra/testssl/ssl/cassandra.pem`* property.

If this property is missing, then Extract generates this error:.

2018-06-09 01:55:37 ERROR OGG-25180 The PEM formatted public key file used to verify the peer's certificate is missing. If SSL is enabled, then it is must to set PEMPUBLICKEYFILE in your Oracle GoldenGate GLOBALS file or in Extract parameter file

SSL is enabled and it is two-way handshake

You must specify these properties for SSL two-way handshake:

```
CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH
CPPDRIVEROPTIONS SSL PEMCLIENTPUBLICKEYFILE /scratch/testcassandra/testssl/ssl/
datastax-cppdriver.pem
CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATEKEYFILE /scratch/testcassandra/testssl/ssl/
datastax-cppdriver-private.pem
CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATEKEYPASSWD cassandra
```

Additionally, consider the following:

- If `ENABLECLIENTAUTH` is missing then Extract assumes that it is one-way handshake so it ignores `PEMCLIENTPRIVATEKEYFILE` and `PEMCLIENTPRIVATEKEYFILE`. The following error occurs because the `cassandra.yaml` file should have `require_client_auth` set to `true`.

  ```
  2018-06-09 02:00:35  ERROR   OGG-00868  No hosts available for the control
  connection.
  ```

- If `ENABLECLIENTAUTH` is used and `PEMCLIENTPRIVATEKEYFILE` is missing, then this error occurs:

  ```
  2018-06-09 02:04:46  ERROR   OGG-25178  The PEM formatted private key file used
  to verify the client's certificate is missing. For two way handshake or if
  ENABLECLIENTAUTH is set, then it is mandatory to set PEMCLIENTPRIVATEKEYFILE in
  your Oracle GoldenGate GLOBALS file or in Extract parameter file.
  ```

- If `ENABLECLIENTAUTH` is use and `PEMCLIENTPUBLICKEYFILE` is missing, then this error occurs:

  ```
  2018-06-09 02:06:20  ERROR   OGG-25179  The PEM formatted public key file used
  to verify the client's certificate is missing. For two way handshake or if
  ENABLECLIENTAUTH is set, then it is mandatory to set PEMCLIENTPUBLICKEYFILE in
  your Oracle GoldenGate GLOBALS file or in Extract parameter file.
  ```

- If the password is set while generating the client private key file then you must add `PEMCLIENTPRIVATEKEYPASSWD` to avoid this error:

  ```
  2018-06-09 02:09:48  ERROR   OGG-25177  The SSL certificate: /scratch/jitiwari/
  testcassandra/testssl/ssl/datastax-cppdriver-private.pem can not be loaded.
  Unable to load private key.
  ```

- If any of the PEM file is missing from the specified absolute path, then this error occurs:

  ```
  2018-06-09 02:12:39  ERROR   OGG-25176  Can not open the SSL certificate: /
  scratch/jitiwari/testcassandra/testssl/ssl/cassandra.pem.
  ```

General SSL Errors

Consider these general errors:

- The SSL connection may fail if you have enabled all SSL required parameters in Extract or GLOBALS file and the SSL is not configured in the `cassandra.yaml` file.

- The absolute path or the qualified name of the PEM file may not correct. There could be access issue on the PEM file stored location.
- The password added during generating the client private key file may not be correct or you may not have enabled it in the Extract parameter or GLOBALS file.

19

Connecting to Microsoft Azure Data Lake

Learn how to connect to Microsoft Azure Data Lake to process big data jobs with Oracle GoldenGate for Big Data.

Use these steps to connect to Microsoft Azure Data Lake from Oracle GoldenGate for Big Data.

1. Download Hadoop 2.9.1 from http://hadoop.apache.org/releases.html.
2. Unzip the file in a temporary directory. For example, `/ggwork/hadoop/hadoop-2.9`.
3. Edit the `/ggwork/hadoop/hadoop-2.9/hadoop-env.sh` file in the directory.
4. Add entries for the `JAVA_HOME` and `HADOOP_CLASSPATH` environment variables:

   ```
   export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
   export HADOOP_CLASSPATH=/ggwork/hadoop/hadoop-2.9.1/share/hadoop/tools/lib/
   *:$HADOOP_CLASSPATH
   ```

   This points to Java 8 and adds the `share/hadoop/tools/lib` to the Hadoop classpath. The library path is not in the variable by default and the required Azure libraries are in this directory.
5. Edit the `/ggwork/hadoop/hadoop-2.9.1/etc/hadoop/core-site.xml` file and add:

   ```
   <configuration>
   <property>
   <name>fs.adl.oauth2.access.token.provider.type</name>
   <value>ClientCredential</value>
   </property>
   <property>
   <name>fs.adl.oauth2.refresh.url</name>
   <value>Insert the Azure https URL here to obtain the access token</value>
   </property>
   <property>
   <name>fs.adl.oauth2.client.id</name>
   <value>Insert the client id here</value>
   </property>
   <property>
   <name>fs.adl.oauth2.credential</name>
   <value>Insert the password here</value>
   </property>
   <property>
   <name>fs.defaultFS</name>
   <value>adl://Account Name.azuredatalakestore.net</value>
   </property>
   </configuration>
   ```
6. Open your firewall to connect to both the Azure URL to get the token and the Azure Data Lake URL. Or disconnect from your network or VPN. Access to Azure Data Lake does not currently support using a proxy server per the Apache Hadoop documentation.

7. Use the Hadoop shell commands to prove connectivity to Azure Data Lake. For example, in the 2.9.1 Hadoop installation directory, execute this command to get a listing of the root HDFS directory.

   ```
   ./bin/hadoop fs -ls /
   ```

8. Verify connectivity to Azure Data Lake.

9. Configure either the HDFS Handler or the File Writer Handler using the HDFS Event Handler to push data to Azure Data Lake, see Using the File Writer Handler (page 5-1). Oracle recommends that you use the File Writer Handler with the HDFS Event Handler.

   Setting the `gg.classpath` example:

   ```
   gg.classpath=/ggwork/hadoop/hadoop-2.9.1/share/hadoop/common/:/ggwork/hadoop/
   hadoop-
   2.9.1/share/hadoop/common/lib/:/ggwork/hadoop/hadoop-
   2.9.1/share/hadoop/hdfs/:/ggwork/hadoop/hadoop-2.9.1/share/hadoop/hdfs/lib/:/
   ggwork/hadoop/hadoop-
   2.9.1/etc/hadoop:/ggwork/hadoop/hadoop-2.9.1/share/hadoop/tools/lib/*
   ```

See https://hadoop.apache.org/docs/current/hadoop-azure-datalake/index.html.

A

Cassandra Handler Client Dependencies

What are the dependencies for the Cassandra Handler to connect to Apache Cassandra databases?

The maven central repository artifacts for Cassandra databases are:

Maven groupId: org.apache.cassandra

Maven atifactId: cassandra-clients

Maven version: the Cassandra version numbers listed for each section

Topics:

- Cassandra Datastax Java Driver 3.1.0 (page A-1)

A.1 Cassandra Datastax Java Driver 3.1.0

```
cassandra-driver-core-3.1.0.jar
cassandra-driver-extras-3.1.0.jar
cassandra-driver-mapping-3.1.0.jar
asm-5.0.3.jar
asm-analysis-5.0.3.jar
asm-commons-5.0.3.jar
asm-tree-5.0.3.jar
asm-util-5.0.3.jar
guava-16.0.1.jar
HdrHistogram-2.1.9.jar
jackson-annotations-2.6.0.jar
jackson-core-2.6.3.jar
jackson-databind-2.6.3.jar
javax.json-api-1.0.jar
jffi-1.2.10.jar
jffi-1.2.10-native.jar
jnr-constants-0.9.0.jar
jnr-ffi-2.0.7.jar
jnr-posix-3.0.27.jar
jnr-x86asm-1.0.2.jar
joda-time-2.9.1.jar
lz4-1.3.0.jar
metrics-core-3.1.2.jar
netty-buffer-4.0.37.Final.jar
netty-codec-4.0.37.Final.jar
netty-common-4.0.37.Final.jar
netty-handler-4.0.37.Final.jar
netty-transport-4.0.37.Final.jar
slf4j-api-1.7.7.jar
snappy-java-1.1.2.6.jar
```

B

Cassandra Capture Client Dependencies

What are the dependencies for the Cassandra Capture (Extract) to connect to Apache Cassandra databases?

The following third party libraries are needed to run Cassandra Change Data Capture.

`cassandra-driver-core` (com.datastax.cassandra)

Download version 3.3.1 from Maven central at: http://central.maven.org/maven2/com/datastax/cassandra/cassandra-driver-core/3.3.1/cassandra-driver-core-3.3.1.jar

Maven coordinates:

```
<dependency>
    <groupId>com.datastax.cassandra</groupId>
    <artifactId>cassandra-driver-core</artifactId>
    <version>3.3.1</version>
</dependency>
```

`cassandra-all` (org.apache.cassandra)

When using 3.9 SDK (see the `TRANLOGOPTIONS CDCREADERSDKVERSION` parameter), download version 3.9 from Maven central: http://central.maven.org/maven2/org/apache/cassandra/cassandra-all/3.9/cassandra-all-3.9.jar

Maven coordinates:

```
<dependency>
    <groupId>org.apache.cassandra</groupId>
    <artifactId>cassandra-all</artifactId>
    <version>3.9</version>
</dependency>
```

When using 3.10 or 3.11 SDK, download version 3.11.0 from Maven central at: http://central.maven.org/maven2/org/apache/cassandra/cassandra-all/3.11.0/cassandra-all-3.11.0.jar

Maven coordinates:

```
<dependency>
    <groupId>org.apache.cassandra</groupId>
    <artifactId>cassandra-all</artifactId>
    <version>3.11.0</version>
</dependency>
```

`commons-lang3` (org.apache.commons)

Download version 3.5 from Maven central at: http://central.maven.org/maven2/org/apache/commons/commons-lang3/3.5/commons-lang3-3.5.jar

Maven coordinates

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
```

```
    <version>3.5</version>
</dependency>
```

`gson` (com.google.code.gson)

Download version 2.8.0 from Maven central at: http://central.maven.org/maven2/com/google/code/gson/gson/2.8.0/gson-2.8.0.jar

Maven coordinates:

```
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.0</version>
</dependency>
```

`jsch` (com.jcraft)

Download version 0.1.54 from maven central:

http://central.maven.org/maven2/com/jcraft/jsch/0.1.54/jsch-0.1.54.jar

Maven coordinates:

```
<dependency>
    <groupId>com.jcraft</groupId>
    <artifactId>jsch</artifactId>
    <version>0.1.54</</version>
</dependency>
```

C
Elasticsearch Handler Client Dependencies

What are the dependencies for the Elasticsearch Handler to connect to Elasticsearch databases?

The maven central repository artifacts for Elasticsearch databases are:

Maven groupId: `org.elasticsearch`

Maven atifactId: `elasticsearch`

Maven version: 2.2.0

Topics:

- Elasticsearch 2.4.4 and Shield Plugin 2.2.2 (page C-1)
- Elasticsearch 5.1.2 with X-Pack 5.1.2 (page C-2)

C.1 Elasticsearch 2.4.4 and Shield Plugin 2.2.2

```
automaton-1.11-8.jar
commons-cli-1.3.1.jar
compress-lzf-1.0.2.jar
elasticsearch-2.4.4.jar
guava-18.0.jar
HdrHistogram-2.1.6.jar
hppc-0.7.1.jar
jackson-core-2.8.1.jar
jackson-dataformat-cbor-2.8.1.jar
jackson-dataformat-smile-2.8.1.jar
jackson-dataformat-yaml-2.8.1.jar
joda-time-2.9.5.jar
jsr166e-1.1.0.jar
lucene-analyzers-common-5.5.2.jar
lucene-backward-codecs-5.5.2.jar
lucene-core-5.5.2.jar
lucene-grouping-5.5.2.jar
lucene-highlighter-5.5.2.jar
lucene-join-5.5.2.jar
lucene-memory-5.5.2.jar
lucene-misc-5.5.2.jar
lucene-queries-5.5.2.jar
lucene-queryparser-5.5.2.jar
lucene-sandbox-5.5.2.jar
lucene-spatial3d-5.5.2.jar
lucene-spatial-5.5.2.jar
lucene-suggest-5.5.2.jar
netty-3.10.6.Final.jar
securesm-1.0.jar
shield-2.2.2.jar
snakeyaml-1.15.jar
spatial4j-0.5.jar
t-digest-3.0.jar
unboundid-ldapsdk-2.3.8.jar
```

C.2 Elasticsearch 5.1.2 with X-Pack 5.1.2

```
commons-codec-1.10.jar
commons-logging-1.1.3.jar
compiler-0.9.3.jar
elasticsearch-5.1.2.jar
HdrHistogram-2.1.6.jar
hppc-0.7.1.jar
httpasyncclient-4.1.2.jar
httpclient-4.5.2.jar
httpcore-4.4.5.jar
httpcore-nio-4.4.5.jar
jackson-core-2.8.1.jar
jackson-dataformat-cbor-2.8.1.jar
jackson-dataformat-smile-2.8.1.jar
jackson-dataformat-yaml-2.8.1.jar
jna-4.2.2.jar
joda-time-2.9.5.jar
jopt-simple-5.0.2.jar
lang-mustache-client-5.1.2.jar
lucene-analyzers-common-6.3.0.jar
lucene-backward-codecs-6.3.0.jar
lucene-core-6.3.0.jar
lucene-grouping-6.3.0.jar
lucene-highlighter-6.3.0.jar
lucene-join-6.3.0.jar
lucene-memory-6.3.0.jar
lucene-misc-6.3.0.jar
lucene-queries-6.3.0.jar
lucene-queryparser-6.3.0.jar
lucene-sandbox-6.3.0.jar
lucene-spatial3d-6.3.0.jar
lucene-spatial-6.3.0.jar
lucene-spatial-extras-6.3.0.jar
lucene-suggest-6.3.0.jar
netty-3.10.6.Final.jar
netty-buffer-4.1.6.Final.jar
netty-codec-4.1.6.Final.jar
netty-codec-http-4.1.6.Final.jar
netty-common-4.1.6.Final.jar
netty-handler-4.1.6.Final.jar
netty-resolver-4.1.6.Final.jar
netty-transport-4.1.6.Final.jar
percolator-client-5.1.2.jar
reindex-client-5.1.2.jar
rest-5.1.2.jar
securesm-1.1.jar
snakeyaml-1.15.jar
t-digest-3.0.jar
transport-5.1.2.jar
transport-netty3-client-5.1.2.jar
transport-netty4-client-5.1.2.jar
x-pack-transport-5.1.2.jar
```

D

Flume Handler Client Dependencies

What are the dependencies for the Flume Handler to connect to Flume databases?

The maven central repository artifacts for Flume databases are:

Maven groupId: `org.apache.flume`

Maven atifactId: `hadoop-ng-skd`

Maven version: the Flume version numbers listed for each section

Topics:

- Flume 1.7.0 (page D-1)
- Flume 1.6.0 (page D-1)
- Flume 1.5.2 (page D-2)
- Flume 1.4.0 (page D-2)

D.1 Flume 1.7.0

```
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-1.7.0.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-3.9.4.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
```

D.2 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
```

D.3 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
```

D.4 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
```

E

HBase Handler Client Dependencies

What are the dependencies for the HBase Handler to connect to Apache HBase databases?

The maven central repository artifacts for HBase databases are:

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

The `hbase-client-x.x.x.jar` file is not distributed with Apache HBase, nor is it mandatory to be in the classpath. The `hbase-client-x.x.x.jar` file is an empty Maven project whose purpose of aggregating all of the HBase client dependencies.

Topics:

- HBase 1.2.5 (page E-1)
- HBase 1.1.1 (page E-2)
- HBase 1.0.1.1 (page E-3)

E.1 HBase 1.2.5

```
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-3.2.2.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-1.2.5.jar
hbase-client-1.2.5.jar
hbase-common-1.2.5.jar
hbase-protocol-1.2.5.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-1.6.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.42.jar
jsr305-1.3.9.jar
junit-4.12.jar
log4j-1.2.17.jar
metrics-core-2.2.0.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
```

E.2 HBase 1.1.1

HBase 1.1.1 is effectively the same as HBase 1.1.0.1. You can substitute 1.1.0.1 in the libraries that are 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
```

E.3 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
```

F

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

Topics:

- Hadoop Client Dependencies (page F-1)

F.1 Hadoop Client Dependencies

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

- HDFS 2.8.0 (page F-1)
- HDFS 2.7.1 (page F-2)
- HDFS 2.6.0 (page F-4)
- HDFS 2.5.2 (page F-5)
- HDFS 2.4.1 (page F-6)
- HDFS 2.3.0 (page F-7)
- HDFS 2.2.0 (page F-8)

F.1.1 HDFS 2.8.0

```
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-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.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-2.8.0.jar
hadoop-auth-2.8.0.jar
hadoop-client-2.8.0.jar
hadoop-common-2.8.0.jar
hadoop-hdfs-2.8.0.jar
hadoop-hdfs-client-2.8.0.jar
hadoop-mapreduce-client-app-2.8.0.jar
hadoop-mapreduce-client-common-2.8.0.jar
hadoop-mapreduce-client-core-2.8.0.jar
hadoop-mapreduce-client-jobclient-2.8.0.jar
hadoop-mapreduce-client-shuffle-2.8.0.jar
hadoop-yarn-api-2.8.0.jar
hadoop-yarn-client-2.8.0.jar
hadoop-yarn-common-2.8.0.jar
hadoop-yarn-server-common-2.8.0.jar
htrace-core4-4.0.1-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.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
jcip-annotations-1.0.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-sslengine-6.1.26.jar
jetty-util-6.1.26.jar
json-smart-1.1.1.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
nimbus-jose-jwt-3.9.jar
okhttp-2.4.0.jar
okio-1.4.0.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
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

F.1.2 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
```

F.1.3 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
```

F.1.4 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
```

F.1.5 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
```

F.1.6 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
```

F.1.7 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
```

G

Kafka Handler Client Dependencies

What are the dependencies for the Kafka Handler to connect to Apache Kafka databases?

The maven central repository artifacts for Kafka databases are:

Maven groupId: `org.apache.kafka`

Maven atifactId: `kafka-clients`

Maven version: the Kafka version numbers listed for each section

Topics:

- Kafka 1.1.0 (page G-1)
- Kafka 1.0.0 (page G-1)
- Kafka 0.11.0.0 (page G-1)
- Kafka 0.10.2.0 (page G-2)
- Kafka 0.10.1.1 (page G-2)
- Kafka 0.10.0.1 (page G-2)
- Kafka 0.9.0.1 (page G-2)

G.1 Kafka 1.1.0

```
kafka-clients-1.1.0.jar
lz4-java-1.4.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.7.1.jar
```

G.2 Kafka 1.0.0

```
kafka-clients-1.0.0.jar
lz4-java-1.4.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.4.jar
```

G.3 Kafka 0.11.0.0

```
kafka-clients-0.11.0.0.jar
lz4-1.3.0.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.2.6.jar
```

G.4 Kafka 0.10.2.0

```
kafka-clients-0.10.2.0.jar
lz4-1.3.0.jar
slf4j-api-1.7.21.jar
snappy-java-1.1.2.6.jar
```

G.5 Kafka 0.10.1.1

```
kafka-clients-0.10.1.1.jar
lz4-1.3.0.jar
slf4j-api-1.7.21.jar
snappy-java-1.1.2.6.jar
```

G.6 Kafka 0.10.0.1

```
kafka-clients-0.10.0.1.jar
lz4-1.3.0.jar
slf4j-api-1.7.21.jar
snappy-java-1.1.2.6.jar
```

G.7 Kafka 0.9.0.1

```
kafka-clients-0.9.0.1.jar
lz4-1.2.0.jar
slf4j-api-1.7.6.jar
snappy-java-1.1.1.7.jar
```

H

Kafka Connect Handler Client Dependencies

What are the dependencies for the Kafka Connect Handler to connect to Apache Kafka Connect databases?

The maven central repository artifacts for Kafka Connect databases are:

Maven groupId: `org.apache.kafka`

Maven artifactId: `kafka_2.11 & connect-json`

Maven version: the Kafka Connect version numbers listed for each section

Topics:

- Kafka 0.11.0.0 (page H-1)
- Kafka 0.10.2.0 (page H-2)
- Kafka 0.10.2.0 (page H-2)
- Kafka 0.10.0.0 (page H-2)
- Kafka 0.9.0.1 (page H-3)
- Confluent 4.1.2 (page H-3)
- Confluent 4.0.0 (page H-4)
- Confluent 3.2.1 (page H-4)
- Confluent 3.2.0 (page H-5)
- Confluent 3.2.1 (page H-5)
- Confluent 3.1.1 (page H-5)
- Confluent 3.0.1 (page H-6)
- Confluent 2.0.1 (page H-6)
- Confluent 2.0.1 (page H-7)

H.1 Kafka 0.11.0.0

```
connect-api-0.11.0.0.jar
connect-json-0.11.0.0.jar
jackson-annotations-2.8.0.jar
jackson-core-2.8.5.jar
jackson-databind-2.8.5.jar
jopt-simple-5.0.3.jar
kafka_2.11-0.11.0.0.jar
kafka-clients-0.11.0.0.jar
log4j-1.2.17.jar
lz4-1.3.0.jar
metrics-core-2.2.0.jar
```

```
scala-library-2.11.11.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.7.25.jar
snappy-java-1.1.2.6.jar
zkclient-0.10.jar
zookeeper-3.4.10.jar
```

H.2 Kafka 0.10.2.0

```
connect-api-0.10.2.0.jar
connect-json-0.10.2.0.jar
jackson-annotations-2.8.0.jar
jackson-core-2.8.5.jar
jackson-databind-2.8.5.jar
jopt-simple-5.0.3.jar
kafka_2.11-0.10.2.0.jar
kafka-clients-0.10.2.0.jar
log4j-1.2.17.jar
lz4-1.3.0.jar
metrics-core-2.2.0.jar
scala-library-2.11.8.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.21.jar
slf4j-log4j12-1.7.21.jar
snappy-java-1.1.2.6.jar
zkclient-0.10.jar
zookeeper-3.4.9.jar
```

H.3 Kafka 0.10.2.0

```
connect-api-0.10.1.1.jar
connect-json-0.10.1.1.jar
jackson-annotations-2.6.0.jar
jackson-core-2.6.3.jar
jackson-databind-2.6.3.jar
jline-0.9.94.jar
jopt-simple-4.9.jar
kafka_2.11-0.10.1.1.jar
kafka-clients-0.10.1.1.jar
log4j-1.2.17.jar
lz4-1.3.0.jar
metrics-core-2.2.0.jar
netty-3.7.0.Final.jar
scala-library-2.11.8.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.21.jar
slf4j-log4j12-1.7.21.jar
snappy-java-1.1.2.6.jar
zkclient-0.9.jar
zookeeper-3.4.8.jar
```

H.4 Kafka 0.10.0.0

```
activation-1.1.jar
connect-api-0.10.0.0.jar
connect-json-0.10.0.0.jar
jackson-annotations-2.6.0.jar
```

```
jackson-core-2.6.3.jar
jackson-databind-2.6.3.jar
jline-0.9.94.jar
jopt-simple-4.9.jar
junit-3.8.1.jar
kafka_2.11-0.10.0.0.jar
kafka-clients-0.10.0.0.jar
log4j-1.2.15.jar
lz4-1.3.0.jar
mail-1.4.jar
metrics-core-2.2.0.jar
netty-3.7.0.Final.jar
scala-library-2.11.8.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.21.jar
slf4j-log4j12-1.7.21.jar
snappy-java-1.1.2.4.jar
zkclient-0.8.jar
zookeeper-3.4.6.jar
```

H.5 Kafka 0.9.0.1

```
activation-1.1.jar
connect-api-0.9.0.1.jar
connect-json-0.9.0.1.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-databind-2.5.4.jar
jline-0.9.94.jar
jopt-simple-3.2.jar
junit-3.8.1.jar
kafka_2.11-0.9.0.1.jar
kafka-clients-0.9.0.1.jar
log4j-1.2.15.jar
lz4-1.2.0.jar
mail-1.4.jar
metrics-core-2.2.0.jar
netty-3.7.0.Final.jar
scala-library-2.11.7.jar
scala-parser-combinators_2.11-1.0.4.jar
scala-xml_2.11-1.0.4.jar
slf4j-api-1.7.6.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.1.1.7.jar
zkclient-0.7.jar
zookeeper-3.4.6.jar
```

H.6 Confluent 4.1.2

```
avro-1.8.1.jar
common-config-4.1.2.jar
commons-compress-1.8.1.jar
common-utils-4.1.2.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.6.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.6.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
```

```
kafka-avro-serializer-4.1.2.jar
kafka-clients-1.1.1-cp1.jar
kafka-schema-registry-client-4.1.2.jar
log4j-1.2.16.jar
lz4-java-1.4.1.jar
netty-3.10.5.Final.jar
paranamer-2.7.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.1.7.1.jar
xz-1.5.jar
zkclient-0.10.jar
zookeeper-3.4.10.jar
```

H.7 Confluent 4.0.0

```
avro-1.8.2.jar
common-config-4.0.0.jar
commons-compress-1.8.1.jar
common-utils-4.0.0.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.1.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.1.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
kafka-avro-serializer-4.0.0.jar
kafka-schema-registry-client-4.0.0.jar
log4j-1.2.16.jar
netty-3.10.5.Final.jar
paranamer-2.7.jar
slf4j-api-1.7.7.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.1.1.3.jar
xz-1.5.jar
zkclient-0.10.jar
zookeeper-3.4.10.jar
```

H.8 Confluent 3.2.1

```
avro-1.7.7.jar
common-config-3.2.1.jar
commons-compress-1.4.1.jar
common-utils-3.2.1.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.5.4.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
kafka-avro-serializer-3.2.1.jar
kafka-schema-registry-client-3.2.1.jar
log4j-1.2.17.jar
netty-3.7.0.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.0.5.jar
xz-1.0.jar
```

```
zkclient-0.10.jar
zookeeper-3.4.8.jar
```

H.9 Confluent 3.2.0

```
avro-1.7.7.jar
common-config-3.2.0.jar
commons-compress-1.4.1.jar
common-utils-3.2.0.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.5.4.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
kafka-avro-serializer-3.2.0.jar
kafka-schema-registry-client-3.2.0.jar
log4j-1.2.17.jar
netty-3.7.0.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.0.5.jar
xz-1.0.jar
zkclient-0.10.jar
zookeeper-3.4.8.jar
```

H.10 Confluent 3.2.1

```
avro-1.7.7.jar
common-config-3.1.2.jar
commons-compress-1.4.1.jar
common-utils-3.1.2.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.5.4.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
kafka-avro-serializer-3.1.2.jar
kafka-schema-registry-client-3.1.2.jar
log4j-1.2.17.jar
netty-3.7.0.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.0.5.jar
xz-1.0.jar
zkclient-0.9.jar
zookeeper-3.4.8.jar
```

H.11 Confluent 3.1.1

```
avro-1.7.7.jar
common-config-3.1.1.jar
commons-compress-1.4.1.jar
common-utils-3.1.1.jar
jackson-annotations-2.5.0.jar
```

```
jackson-core-2.5.4.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.5.4.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
kafka-avro-serializer-3.1.1.jar
kafka-schema-registry-client-3.1.1.jar
log4j-1.2.17.jar
netty-3.7.0.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.0.5.jar
xz-1.0.jar
zkclient-0.9.jar
zookeeper-3.4.8.jar
```

H.12 Confluent 3.0.1

```
avro-1.7.7.jar
common-config-3.0.1.jar
commons-compress-1.4.1.jar
common-utils-3.0.1.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.5.4.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
junit-3.8.1.jar
kafka-avro-serializer-3.0.1.jar
kafka-schema-registry-client-3.0.1.jar
log4j-1.2.17.jar
netty-3.2.2.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.0.5.jar
xz-1.0.jar
zkclient-0.5.jar
zookeeper-3.4.3.jar
```

H.13 Confluent 2.0.1

```
avro-1.7.7.jar
common-config-2.0.1.jar
commons-compress-1.4.1.jar
common-utils-2.0.1.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.5.4.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
junit-3.8.1.jar
kafka-avro-serializer-2.0.1.jar
kafka-schema-registry-client-2.0.1.jar
log4j-1.2.17.jar
netty-3.2.2.Final.jar
```

```
paranamer-2.3.jar
slf4j-api-1.6.4.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.0.5.jar
xz-1.0.jar
zkclient-0.5.jar
zookeeper-3.4.3.jar
```

H.14 Confluent 2.0.1

```
avro-1.7.7.jar
common-config-2.0.0.jar
commons-compress-1.4.1.jar
common-utils-2.0.0.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.5.4.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
junit-3.8.1.jar
kafka-avro-serializer-2.0.0.jar
kafka-schema-registry-client-2.0.0.jar
log4j-1.2.17.jar
netty-3.2.2.Final.jar
paranamer-2.3.jar
slf4j-api-1.6.4.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.0.5.jar
xz-1.0.jar
zkclient-0.5.jar
zookeeper-3.4.3.jar
```

MongoDB Handler Client Dependencies

What are the dependencies for the MongoDB Handler to connect to MongoDB databases?

Oracle GoldenGate requires that you use the 3.4.3 MongoDB Java Driver or higher integration with MongoDB. You can download this driver from:

http://mongodb.github.io/mongo-java-driver/

Topics:

- MongoDB Java Driver 3.4.3 (page I-1)

I.1 MongoDB Java Driver 3.4.3

You must include the path to the MongoDB Java driver in the `gg.classpath` property. To automatically download the Java driver from the Maven central repository, add the following lines in the `pom.xml` file, substituting your correct information:

```
<!-- https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver -->
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>3.4.3</version>
</dependency>
```

J

Optimized Row Columnar Event Handler Client Dependencies

What are the dependencies for the Optimized Row Columnar (OCR) Handler?

The maven central repository artifacts for ORC are:

Maven groupId: `org.apache.orc`

Maven atifactId: `orc-core`

Maven version: 1.4.0

The Hadoop client dependencies are also required for the ORC Event Handler, see Hadoop Client Dependencies (page F-1).

Topics:

- ORC Client Dependencies (page J-1)

J.1 ORC Client Dependencies

```
aircompressor-0.3.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
asm-3.1.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.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
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.6.4.jar
hadoop-auth-2.6.4.jar
hadoop-common-2.6.4.jar
hive-storage-api-2.2.1.jar
htrace-core-3.0.4.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jdk.tools-1.6.jar
```

```
jersey-core-1.9.jar
jersey-server-1.9.jar
jsch-0.1.42.jar
log4j-1.2.17.jar
netty-3.7.0.Final.jar
orc-core-1.4.0.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

K

Parquet Event Handler Client Dependencies

What are the dependencies for the Parquet Event Handler?

The maven central repository artifacts for Parquet are:

Maven groupId: `org.apache.parquet`

Maven atifactId: `parquet-avro`

Maven version: 1.9.0

Maven groupId: `org.apache.parquet`

Maven atifactId: `parquet-hadoop`

Maven version: 1.9.0

The Hadoop client dependencies are also required for the Parquet Event Handler, see Hadoop Client Dependencies (page F-1).

Topics:

- Parquet Client Dependencies (page K-1)

K.1 Parquet Client Dependencies

```
avro-1.8.0.jar
commons-codec-1.5.jar
commons-compress-1.8.1.jar
commons-pool-1.5.4.jar
fastutil-6.5.7.jar
jackson-core-asl-1.9.11.jar
jackson-mapper-asl-1.9.11.jar
paranamer-2.7.jar
parquet-avro-1.9.0.jar
parquet-column-1.9.0.jar
parquet-common-1.9.0.jar
parquet-encoding-1.9.0.jar
parquet-format-2.3.1.jar
parquet-hadoop-1.9.0.jar
parquet-jackson-1.9.0.jar
slf4j-api-1.7.7.jar
snappy-java-1.1.1.6.jar
xz-1.5.jar
```