Tivoli Netcool Support's Guide to the JDBC2 Gateway

by

Jim Hutchinson

Document release: 2.1



Table of Contents

1Introduction 3

1.1Overview 3

2Database Considerations 4

2.1Example Database drivers and URLs 4
2.1.1DB2 4
2.1.2Oracle 4
2.1.3Sybase 4
2.2Database Schemas 5
2.3Example DB2 REPORTING database 6
2.3.1Reporting database checks 6
2.3.2Property file 7
2.3.3Table replication file 8
2.3.4Map file 8

3JDBC2 Gateway Configuration 9

3.1G_JDBC2.props 10
3.2jdbc.map 11
3.3jdbc2.rdrwtr.tblrep.def 12
3.4Gate.Jdbc.Mode 13
3.4.1AUDIT Mode 14
3.4.2REPORTING Mode 15

4Monitoring Performance 16

4.1Outstanding batches 16
4.2Batch processing 16
4.3Main Gateway 17
4.4SAF Processor 17
4.5Pool Thread 17

5Property Considerations 18

5.1Gate.RdrWtr.Description 18
5.2Java Memory 18
5.3Object Server failover and failback 19
5.4Data Forwarding 20
5.4.1The jdbc.rdrwtr.tblrep.def file 20
5.5Debug logging 21
5.6Dumpprops 21

6Event Data Flow 22

6.1Audit Database Configuration 22
6.2Resident Database Configuration 22
6.2.1Standard Triggers Method 23
6.2.2Temporal Trigger Method 24

7Common Issues 25

7.1Status updates are not happening 25
7.2Event Re-Awaken issues 26
7.3Recommended SQLSTATE settings 27
7.4Oracle RAC URL example 29
7.5Encrypting passwords and other properties 30
7.6Creating an Object Server Gateway User 31
7.7Setting the Gateways Java path 32
7.8DB2 tracing batches 33
7.9Preventing StateChange Feedback 35
7.10Database Down Data loss 36

IBM Copyright 2026

7.11REPORTER_JOURNAL Unique Constraints....37
7.12Fast Forwarding an Object Servers Serial....39
7.12.1Increasing an Object Server Serial using SQL....39
7.12.2Increasing an Object Server Serial using nco_dbinit....40

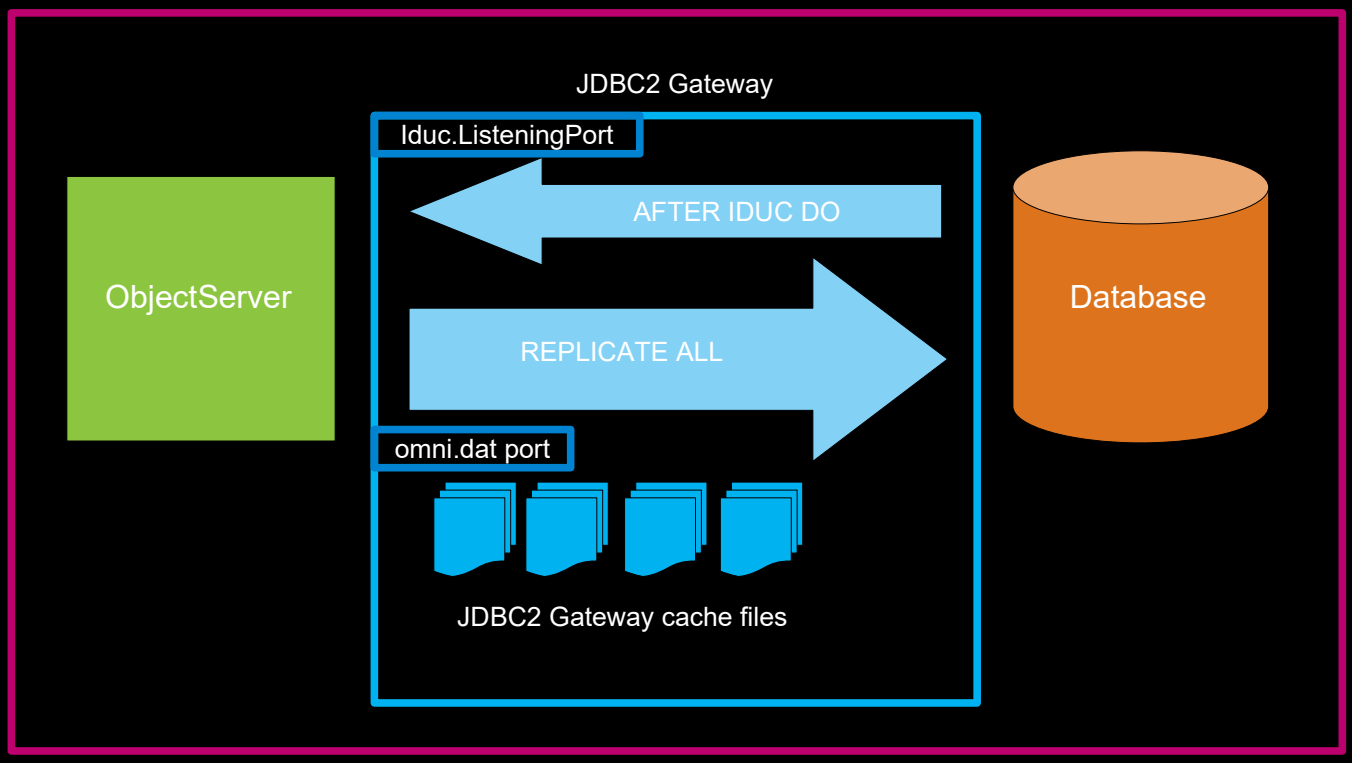
8Full list of property settings....41

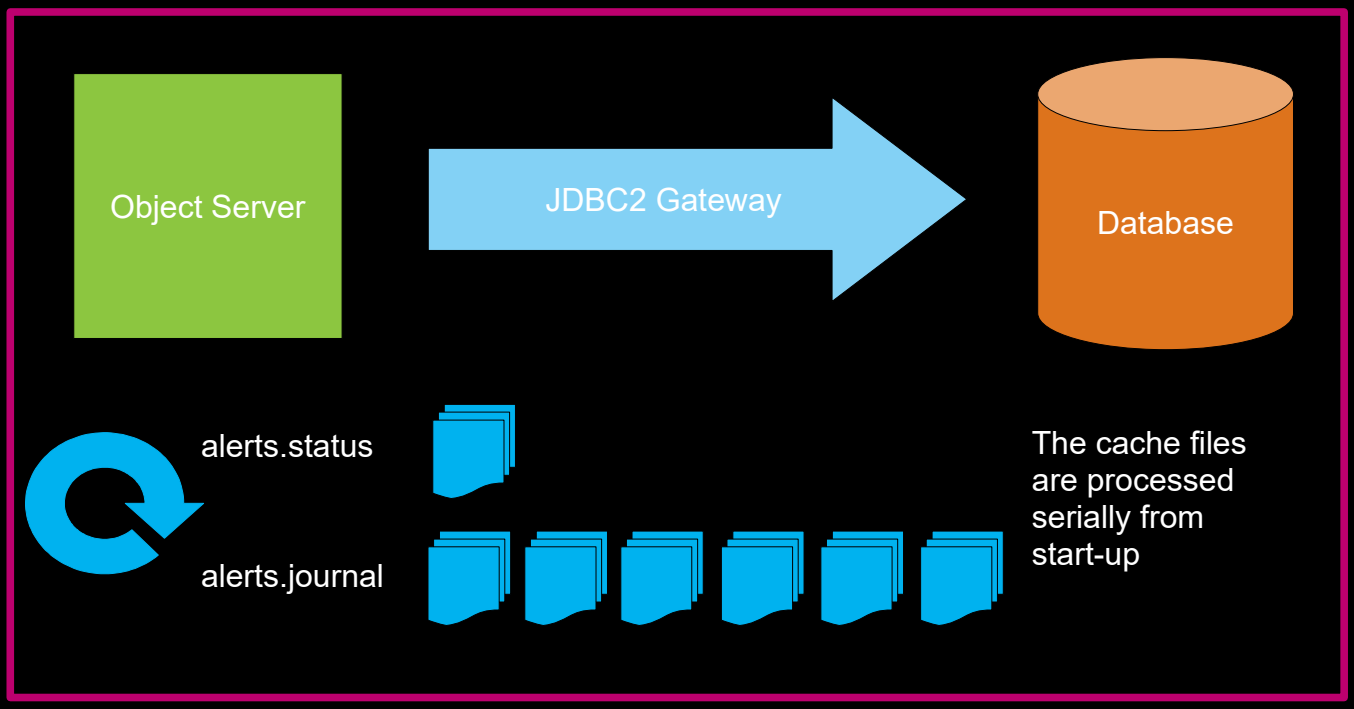
8.1General properties....41
8.2File properties....41
8.3Cache properties....41
8.4Conversion table properties....41
8.5Debugging properties....42
8.6Logging properties....42
8.6.1Trace mode properties....42
8.7JDBC properties....43
8.7.1JDBC Connection properties....43
8.7.2JDBC Connection initialization....43
8.7.3Reconnection properties....43
8.7.4Target database tables....44
8.7.5AUDIT Mode properties....44
8.7.6Field properties....44
8.7.7Database codes....45
8.7.8Important JDBC properties....45
8.7.9Other JDBC properties....45
8.8Object Server properties....46
8.9HTTP Command port properties....46

1 Introduction

1.1 Overview

The JDBC2 Gateway replaces the JDBC gateway. The JDBC2 Gateway is a Java version of the JDBC gateway. The JDBC2 Gateway reads data from the Object Server and inserts this data into the target database using the specified JDBC driver. The JDBC drivers are provided by the Database vendor, and issues with them should be reported to this vendor rather IBM Tivoli Netcool support.





2 Database Considerations

2.1 Example Database drivers and URLs

Note that the chosen Jar files are based on the version of Java being used by the JDBC2 gateway.

2.1.1 DB2

JDBC Driver name : com.ibm.db2.jcc.DB2Driver
JDBC Driver jars : db2jcc.jar
JDBC URL : jdbc:db2//<FQDN-hostname>:50000/<database>

2.1.2 Oracle

JDBC Driver name : oracle.jdbc.driver.OracleDriver
JDBC Driver jars : classes12.jar | ojdbc14.jar
JDBC URL : jdbc:oracle:thin:@ //<FQDN-hostname>:1521: <database>

2.1.3 Sybase

JDBC Driver name : net.sourceforge.jtds.jdbc.Driver
JDBC Driver jars : jtds.jar
JDBC URL : jdbc:jtds:sybase://<FQDN-hostname>:5000; DatabaseName=<database>

IBM Copyright 2026

2.2 Database Schemas

The target JDBC database schemas are provided by the AUDIT and REPORTING packages scripts directory.

Two schema types are provided in the $NCHOME/omnibus/gates directory:

- REPORTING [Reporting Mode : nco-g-jdbc-reporting-scripts]
- AUDIT [Audit Mode : nco-g-audit]

Ensure that the correct mapping file is used for the chosen JDBC2 Gateway mode, and use the example files provided to create a set of JDBC2 Gateway files suitable for the deployment.

- Gate.Jdbc.Mode : 'REPORTING'
  - reporting.G_JDBC2.props
  - reporting.jdbc2.map

- Gate.Jdbc.Mode : 'AUDIT'
  - audit.G_JDBC2.props
  - audit.jdbc2.map

If a custom reporting method is to be used, consider the fields required in the alerts.status that need to be added to allow the various states of the alarms to be captured, and reported on.

For example.

AcknowledgedTime
AcknoweldgedBy
TicketOpenedTime
TicketClosedTime
ClearedTime
ClearedBy
MaximumSeverity
EscalationTime

IBM Copyright 2026

2.3 Example DB2 REPORTING database

The DB2 table spaces need to be created for production usage by a Database Administrator to ensure that issues with performance and loads are not encountered. Onto these table spaces, the REPORTING schema can be applied. The JDBC2 gateway only requires three main tables, and can be configured to send events to only reporter_status. The reporter_status table is the main table in the JDBC2 gateway functionality and must always be replicated.

2.3.1 Reporting database checks

Login to the DB2 server as the DB2 driver user [reporter].
Configure the user environment for DB2 commands.

Schema check.

```
db2 connect to reporter
db2 describe table reporter_status
db2 describe table reporter_journal
db2 describe table reporter_details
db2 disconnect reporter
```

Row count check.

```
db2 connect to reporter
db2 'select count(*) from reporter_status'
db2 'select count(*) from reporter_journal'
db2 'select count(*) from reporter_details'
db2 disconnect reporter
```

Maintenance commands

Number of days in 1-5 years:

```
365*1 =  365
365*2 =  730
365*3 = 1095
365*4 = 1460
365*5 = 1825
```

```
db2 "select count(*) from reporter_status where LastOccurrence < TIMESTAMP(CURRENT DATE - 365 DAYS, '00:00:00')"
```

For the delete commands, working backwards may be required to prevent loading the DB2 server.
For example.

```
db2 "delete from reporter_status where LastOccurrence < TIMESTAMP(CURRENT DATE - 1825 DAYS, '00:00:00')"
db2 "delete from reporter_status where LastOccurrence < TIMESTAMP(CURRENT DATE - 1460 DAYS, '00:00:00')"
db2 "delete from reporter_status where LastOccurrence < TIMESTAMP(CURRENT DATE - 1095 DAYS, '00:00:00')"
db2 "delete from reporter_status where LastOccurrence < TIMESTAMP(CURRENT DATE - 730 DAYS, '00:00:00')"
db2 "delete from reporter_status where LastOccurrence < TIMESTAMP(CURRENT DATE - 365 DAYS, '00:00:00')"
```

After purging reporter_status, purge the other tables, if needed:

```
db2 'delete from reporter_journal where Serial not in (select Serial from reporter_status)'
db2 'delete from reporter_details where Identifier not in "(select Identifier from reporter_status)'
```

2.3.2 Property file

```
File: '$OMNIHOME/gates/G_JDBC2/G_JDBC2.def'

# Unique instance name
Name: 'G_JDBC2'

# Reporting mode and database schema
Gate.Jdbc.Mode: 'REPORTING'

# Standard REPORTER tables
Gate.Jdbc.StatusTableName: 'REPORTER_STATUS'
Gate.Jdbc.JournalTableName: 'REPORTER_JOURNAL'
Gate.Jdbc.DetailsTableName: 'REPORTER_DETAILS'

# Database connection - DB2
Gate.Jdbc.Driver: 'com.ibm.db2.jcc.DB2Driver'
Gate.Jdbc.Url: 'jdbc:db2://db2.ibm.com:50000/REPORTER'
Gate.Jdbc.Username: 'reporter'
Gate.Jdbc.Password: 'reporter'

Gate.Jdbc.ReconnectTimeout: 30
Gate.Jdbc.InitializationString: ''

# Object Server connection
Gate.RdrWtr.Server: 'AGG_V'
Gate.RdrWtr.Username: 'root'
Gate.RdrWtr.Password: ''

# Configuration files
Gate.MapFile: '$OMNIHOME/gates/G_JDBC2/jdbc2.map'
Gate.RdrWtr.TblReplicateDefFile: '$OMNIHOME/gates/G_JDBC2/jdbc2.rdrwtr.tblrep.def'

# Logging - Log size in bytes
MessageLevel: 'info:info'
MaxLogFileSize: 1024000000
MessageLog: '$OMNIHOME/log/G_JDBC2.log'

# Known issue - define Gate.TargetIdField
Gate.TargetIdField: 'Location'

# Recommended settings
Gate.Jdbc.ResyncMode: 'UNI'
Gate.Jdbc.Connections: 12
Gate.RdrWtr.IducFlushRate: 21
```

IBM Copyright 2026

2.3.3 Table replication file

File: '$OMNIHOME/gates/G_JDBC2/jdbc2.rdrwtr.tblrep.def'

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
;
```

2.3.4 Map file

File: '$OMNIHOME/gates/G_JDBC2/jdbc2.map'

```
CREATE MAPPING StatusMap
(
    'IDENTIFIER'        = '@Identifier'         ON INSERT ONLY,
    'SERIAL'            = '@Serial'             ON INSERT ONLY,
    'NODE'              = '@Node'               ON INSERT ONLY,
    'NODEALIAS'         = '@NodeAlias'          ON INSERT ONLY NOTNULL '@Node',
    'MANAGER'           = '@Manager'            ON INSERT ONLY,
    'AGENT'             = '@Agent'              ON INSERT ONLY,
    'ALERTGROUP'        = '@AlertGroup'         ON INSERT ONLY,
    'ALERTKEY'          = '@AlertKey'           ON INSERT ONLY,
    'SEVERITY'          = '@Severity',
    'SUMMARY'           = '@Summary',
    # APAR IV79462: The STATECHANGE field is only defined in the DB2 schema, comment out if not required
    'STATECHANGE'       = '@StateChange'        CONVERT TO DATE,
    'FIRSTOCCURRENCE'   = '@FirstOccurrence'    ON INSERT ONLY CONVERT TO DATE,
    'LASTOCCURRENCE'    = '@LastOccurrence'     CONVERT TO DATE,
    'LASTMODIFIED'      = '@StateChange'        CONVERT TO DATE,
    'POLL'              = '@Poll'               ON INSERT ONLY,
    'TYPE'              = '@Type'               ON INSERT ONLY,
    'TALLY'             = '@Tally',
    'CLASS'             = '@Class'              ON INSERT ONLY,
    'GRADE'             = '@Grade'              ON INSERT ONLY,
    'LOCATION'          = '@Location'           ON INSERT ONLY,
    'OWNERUID'          = '@OwnerUID',
    'OWNERGID'          = '@OwnerGID',
    'ACKNOWLEDGED'      = '@Acknowledged',
    # Oracle Reporter schema does not have this field
    'FLASH'             = '@Flash'              ON INSERT ONLY,
    'EVENTID'           = '@EventId'            ON INSERT ONLY,
    'EXPIRETIME'        = '@ExpireTime'         ON INSERT ONLY,
    'PROCESSREQ'        = '@ProcessReq',
    'SUPPRESSESCL'      = '@SuppressEscl',
    'CUSTOMER'          = '@Customer'           ON INSERT ONLY,
    'SERVICE'           = '@Service'            ON INSERT ONLY,
    'PHYSICALSLOT'      = '@PhysicalSlot'       ON INSERT ONLY,
    'PHYSICALPORT'      = '@PhysicalPort'       ON INSERT ONLY,
    'PHYSICALCARD'      = '@PhysicalCard'       ON INSERT ONLY,
    'TASKLIST'          = '@TaskList',
    'NMOSSERIAL'        = '@NmosSerial'         ON INSERT ONLY,
    'NMOSOBJINST'       = '@NmosObjInst'        ON INSERT ONLY,
    'NMOSCAUSETYPE'     = '@NmosCauseType',
    'LOCALNODEALIAS'    = '@LocalNodeAlias'     ON INSERT ONLY,
    'LOCALPRIOBJ'       = '@LocalPriObj'        ON INSERT ONLY,
    'LOCALSECOBJ'       = '@LocalSecObj'        ON INSERT ONLY,
    'LOCALROOTOBJ'      = '@LocalRootObj'       ON INSERT ONLY,
    'REMOTENODEALIAS'   = '@RemoteNodeAlias'    ON INSERT ONLY,
    'REMOTEPRIOBJ'      = '@RemotePriObj'       ON INSERT ONLY,
    'REMOTESECOBJ'      = '@RemoteSecObj'       ON INSERT ONLY,
    'REMOTEROOTOBJ'     = '@RemoteRootObj'      ON INSERT ONLY,
    'X733EVENTTYPE'     = '@X733EventType'      ON INSERT ONLY,
    'X733PROBABLECAUSE' = '@X733ProbableCause',
    'X733SPECIFICPROB'  = '@X733SpecificProb'   ON INSERT ONLY,
    'X733CORRNOTIF'     = '@X733CorrNotif'      ON INSERT ONLY,
    'ORIGINALSEVERITY'  = '@Severity'           ON INSERT ONLY,

# NB do not concatenate additional values for ServerName and ServerSerial !
    'SERVERNAME'                = '@ServerName'         ON INSERT ONLY,
    'SERVERSERIAL'              = '@ServerSerial'       ON INSERT ONLY
);
```

3 JDBC2 Gateway Configuration

It is best practice to create a gateway specific directory in $NCHOME/omnibus/gates that contains all of the files used by the gateway so that all the files are easy to locate and backup.

The main configuration files are;

- G_JDBC2.props
- jdbc.rdrwtr.tblrep.def
- jdbc.map

JDBC2 files

- jdbc2.generic.js
- jdbc2.notification.js
- nco_g_jdbc2.env

Where:
Gate.Cache.NotificationMap: '${OMNIHOME}/gates/jdbc2/jdbc2.notification.js'
Gate.Cache.GenericScript: '${OMNIHOME}/gates/jdbc2/jdbc2.generic.js'

The JDBC2 gateway environment file.
$NCHOME/omnibus/gates/jdbc2/nco_g_jdbc2.env
$NCHOME/omnibus/platform/linux2x86/bin64/nco_g_jdbc2.env

IBM Copyright 2026

3.1 G_JDBC2.props

Use the '-dumpprops' command to determine the available JDBC2 Gateway properties.

```
nco_g_jdbc2 -propsfile G_JDBC2.props  -dumpprops
```

An example for a REPORTING Oracle 19c historical database is given below;

```
Name: 'G_JDBC2'
Gate.Jdbc.Mode: 'REPORTING'
# Known issue
Gate.TargetIdField: 'Location'
# Tables to replicate to
Gate.Jdbc.StatusTableName: 'REPORTER_STATUS'
Gate.Jdbc.JournalTableName: 'REPORTER_JOURNAL'
Gate.Jdbc.DetailsTableName: 'REPORTER_DETAILS'
# JDBC target database server - Oracle 19c
Gate.Jdbc.Url: 'jdbc:oracle:thin:@server.uk.ibm.com:1521:ORACLE'
Gate.Jdbc.Driver: 'oracle.jdbc.driver.OracleDriver'
Gate.Jdbc.Username: 'reporter'
Gate.Jdbc.Password: 'reporter'
# Object Server Connection settings
Gate.RdrWtr.Server: 'AGG_V'
Gate.RdrWtr.Username: 'root'
Gate.RdrWtr.Password: ''
Gate.Jdbc.ReconnectTimeout: 30
Gate.Jdbc.InitializationString: ''
# Configuration files
Gate.MapFile: '$NCHOME/omnibus/gates/G_JDBC2/jdbc.map'
Gate.RdrWtr.TblReplicateDefFile:
'$NCHOME/omnibus/gates/G_JDBC2/jdbc.rdrwtr.tblrep.def'
# Logging
MessageLevel: 'info:info'
MessageLog: '$NCHOME/omnibus/log/G_JDBC2.log'
MaxLogFileSize: 1024000000
# Synchronisation mode
Gate.Jdbc.ResyncMode: 'UNI'
# Recommended settings
Gate.Jdbc.Connections: 10
Gate.RdrWtr.IducFlushRate: 21
# Debugging - Comment out for Production usage
###
# Debugging
# MessageLevel: 'debug:debug'
# MaxLogFileSize: 1024000000
# Gate.Java.Debug: 'TRUE'
# Gate.Mapper.Debug: 'TRUE'
# Gate.RdrWtr.Debug: 'TRUE'
# Gate.RdrWtr.LogOSSql: 'TRUE'
# EOF
```

3.2 jdbc.map

There are two map file examples provided for the two modes of operation the JDBC2 gateways supports;

- audit.jdbc2.map
- reporting.jdbc2.map

The mapping file conforms to the standard object server mapping file syntax, and needs to be maintained when new fields are added to the object server and are required to be transferred or replicated to the historical database.

Modifications to the mapping file may be required based on the type of historical database.

For example, Sybase is case sensitive, and typically users prefer to use the same column syntax as the object server. The default mapping file uses capital letters, rather than the mixed case column names seen in the object server. Which means the default mapping file for Sybase needs to be edited to meet the requirements of the target database field names.

3.3 jdbc2.rdrwtr.tblrep.def

The table replication definition file is of the same format as the object server gateways' table replication file.
e.g.

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap';
REPLICATE ALL FROM TABLE 'alerts.journal'
USING MAP 'JournalMap';
```

Allowed settings;

```
REPLICATE ALL
FROM TABLE sourcetable
USING MAP mapname
[FILTER WITH filter]
[INTO targettable]
[ORDER BY order, ... ]
[AFTER IDUC DO afteriduc] ;
```

The use of a FILTER and AFTER IDUC DO commands, are used to reduce the volume of events being forwarded. Other options may be used depending upon the requirements of the historical database and the type of events being forwarded.

Note the comment in the table replication file about not forwarding DELETES.

```
# The gateway requires deletes be replicated, so that the cache can be
# purged of deleted alerts. Never suppress deletes here, use the
# Gate.Jdbc.SuppressDeletes property instead
```

Usage.

To prevent deletes from being forwarded, keep the ALL table replication statement and enable the gateway property setting.

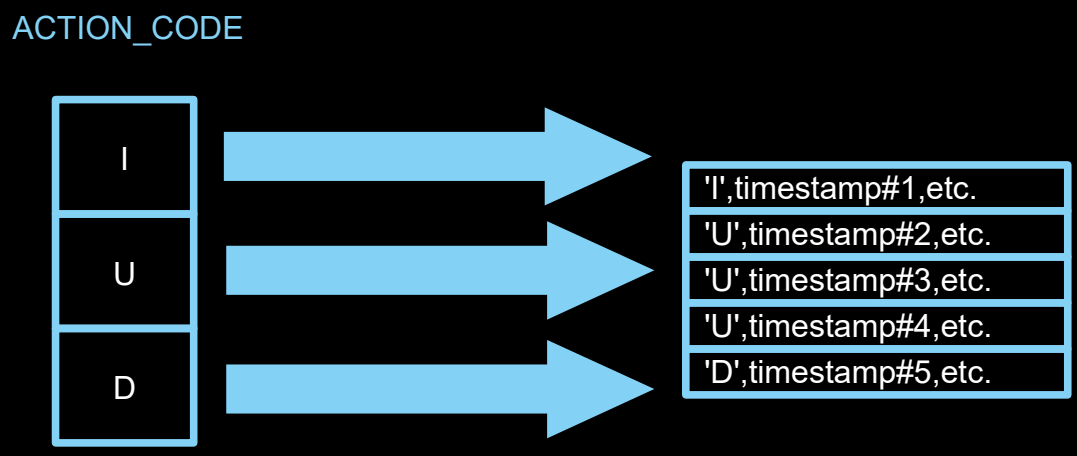
```
Gate.Jdbc.SuppressDeletes : TRUE
```

3.4 Gate.Jdbc.Mode

The JDBC2 gateway property file has a setting named Gate.Jdbc.Mode which controls the way in which the gateway sends events to the target database. This property takes the value of 'REPORTING' or 'AUDIT' and a set of supporting files are provided to assist in the configuration of the JDBC2 gateway. Each mode requires a specific database configuration and a separate downloadable package is provided to assist in the target schema's configuration.

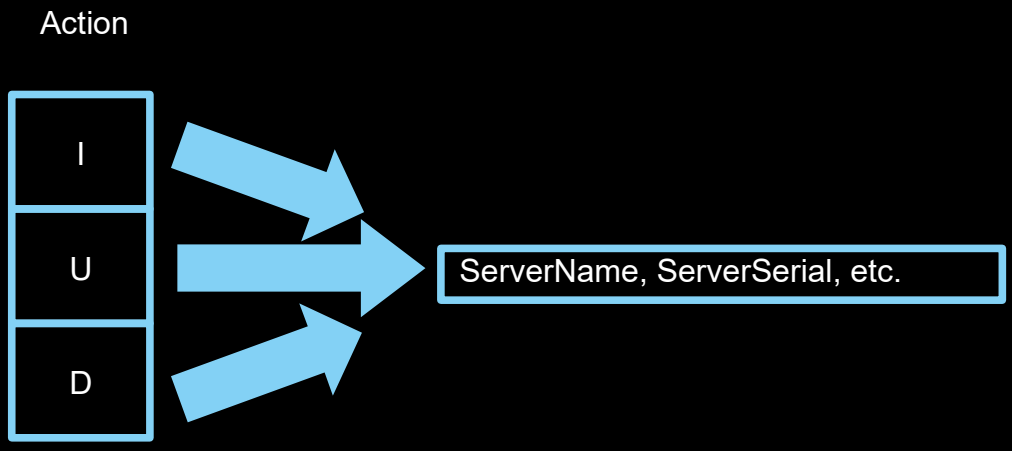
AUDIT Mode Overview
The AUDIT mode uses two functions unique to the AUDIT mode to populate the database. The STATUS rows are unique based on the ActionCode, ActionTime, ServerName and ServerSerial fields.

ACTION_CODE : Auto-generated single character used to represent Insert, Update and Delete
ACTION_TIME : Auto-generated UNIX Timestamp used to hold the object server timestamp when the action occurred



With AUDIT mode the Insert, Update and Delete actions in the object server create a unique row in the database provided that these are sent once within the object servers IDUC period

REPORTING Mode Overview
The REPORTING mode mimics the way rows are updated in the object server, to populate the database. The REPORTER_STATUS rows are unique based on ServerName and ServerSerial fields.



3.4.1 AUDIT Mode

The differences between the AUDIT mode and REPORTING mode are seen in the JDBC2 gateways property and mapping file, as these need to reflect the AUDIT mode database schema. The AUDIT mode attempts to capture every Insert, Update, Delete seen in the object server based on the object servers IDUC period and JDBC2 gateways table replication.

The main table is 'status', with each row there being uniquely defined by the fields ActionCode, ActionTime, ServerName and ServerSerial. This allows the history of a specific event, defined by the ServerName and ServerSerial, to be stored and reported on. Within the object server, the Identifier, ServerName and ServerSerial defines an event, with Identifier being capable of transcending one or more unique ServerName and ServerSerial pair.

Rows a defined uniquely in time using the ActionTime field, and are not updated after creation due to the tables constraints.

File : audit.G_JDBC.props

```
Gate.Jdbc.Mode             : 'AUDIT'

Gate.Jdbc.StatusTableName  : 'status'
Gate.Jdbc.JournalTableName: 'journal'
Gate.Jdbc.DetailsTableName: 'details'
```

File : audit.jdbc.map

```
CREATE MAPPING StatusMap
(
    'ACTIONCODE'         = ACTION_CODE,
    'ACTIONTIME'         = ACTION_TIME CONVERT TO DATE,
…
# NB do not concatenate additional values for ServerName and ServerSerial !
    'SERVERNAME'                = '@ServerName'         ON INSERT ONLY,
    'SERVERSERIAL'              = '@ServerSerial'       ON INSERT ONLY
);
```

AUDIT Database schema highlights

```
DROP TABLE __STATUS__ CASCADE CONSTRAINTS;
CREATE TABLE __STATUS__
(
        ActionTime              DATE NOT NULL,
        ActionCode              CHAR(1) NOT NULL,
…
        ServerName              VARCHAR2(64) NULL,
        ServerSerial            NUMBER(16) NULL
)

CREATE UNIQUE INDEX __STATUS___idx ON __STATUS__
  (ActionTime,ActionCode,ServerSerial,ServerName);
```

3.4.2 REPORTING Mode

The REPORTING mode allows events to stored in a database as single rows per ServerName and ServerSerial. Fields can be updated after the row is initially created, and are used to capture the full life-cycle of the event.

For example the OriginalSeverity is capture using the mapping:

```
'ORIGINALSEVERITY'  = '@Severity'           ON INSERT ONLY,
```

and the events deletion is captured using the DeletedAt database field, which is populated automatically by the JDBC2 gateway when the event is deleted in the object server.

File : reporting.G_JDBC.props

```
Gate.Jdbc.Mode               : 'REPORTING'

Gate.Jdbc.StatusTableName    : 'REPORTER_STATUS'
Gate.Jdbc.JournalTableName   : 'REPORTER_JOURNAL'
Gate.Jdbc.DetailsTableName   : 'REPORTER_DETAILS'
```

File : reporting.jdbc.map

```
CREATE MAPPING StatusMap
(
    'IDENTIFIER'                = '@Identifier'         ON INSERT ONLY,
    'SERIAL'                    = '@Serial'             ON INSERT ONLY,
…
# NB do not concatenate additional values for ServerName and ServerSerial !
    'SERVERNAME'                = '@ServerName'         ON INSERT ONLY,
    'SERVERSERIAL'              = '@ServerSerial'       ON INSERT ONLY
);
```

REPORTING Database schema highlights

```
CREATE TABLE reporter_status
(
       Identifier        VARCHAR2(255)  NULL,
       Serial            NUMBER(16)     NULL,
…
       ServerName        VARCHAR2(64)   NOT NULL,
       ServerSerial      NUMBER(16)     NOT NULL,
       PRIMARY KEY (ServerName, ServerSerial)
)
```

4 Monitoring Performance

The STATS logging is visible at MessageLevel 'info' and provides details on the reader and writer parts of the JDBC Gateway.

Processing sequence.

```
[Main Gateway] → [SAF Processor] → [Thread pool]
```

4.1 Outstanding batches

The STATS log message for Outstanding batches reports on the pending inserts|updates|deletes for the database.

Example heathy logging.

```
Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 1
Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 0
Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 1
```

Example load and recover.

```
26/07/07 07:52:38: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 0
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 19
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 18
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 17
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 16
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 15
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 14
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 13
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 12
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 11
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 10
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 9
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 8
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 7
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 6
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 5
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 4
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 3
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 2
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 1
26/07/07 08:01:41: Information: [pool-3-thread-1] pool-3-thread-1: STATS: Outstanding batches 0
```

4.2 Batch processing

The IUD data is batched, identified in the logging using batch number strings, '`#########`'.

```
07:28:00: Information: [Main Gateway] Main Gateway: STATS: ######### Batch write time 165 ms (3993.939393939394
rows/second) with 659 rows
07:28:01: Information: [Main Gateway] Main Gateway: STATS: Cache write time is 133 ms for #########
07:28:03: Information: [pool-3-thread-1] pool-3-thread-1: STATS: ######### Batch execution time: 1977 ms
(333.3333333333333 rows/second) with 659 rows.
```

The number of rows processed, and time taken to process the rows is recoded.

From the STATS logging the three lines record the processing rate of around 3 seconds [219.667 rows/second].

```
659/0.165 = 3,993.939393939394 rows/second
659/0.133 = 4,954.887 rows/second
659/1.977 =   333.3333333333333 rows/second
```

4.3 Main Gateway

The STATS Main Gateway records the Object Server write performance for the batch and cache tasks [reader].

```
Information: [Main Gateway] Main Gateway: STATS: ######### Batch write time Y ms (X rows/second) with N rows
Information: [Main Gateway] Main Gateway: STATS: Cache write time is YY ms for #########
```

For the same batch number, '##########', the same number of rows are processed, with the log lines recording the processing time in ms [1000th of a second].

The reported rate in rows per second [X] allows representation of the readers performance.

```
Information: [Main Gateway] STATS: [IDUC processing] count=301371 time=180867 rate=1666.2576 rolling=161.72107
Information: [Main Gateway] STATS: [IDUC processing] count=301371 uptime=4115 rate=73.221756 busy=4.394384%
```

The Main Gateway logging records the running count of processed rows, along with time and rate information.

4.4 SAF Processor

The SAF processor receives row notifications from the Object Server; INSERT, UPDATE, DELETE
The logging reports the cumulative count of notifications processed along with time and rate data.
SAF stands for Store And Forward.

```
Information: [SAF Processor] STATS: [SAF processing] count=301353 time=157457 rate=1913.8749 rolling=1861.7821
Information: [SAF Processor] STATS: [SAF processing] count=301353 uptime=4144 rate=72.71984 busy=3.7996132%
```

The SAF Processor logging records the running count of processed rows, along with time and rate information.

4.5 Pool Thread

The pool thread logging reports on the writing of the batch data to the target database [writer].

```
Information: [pool-3-thread-1] pool-3-thread-1: STATS: ######### Batch execution time: YYY ms (XX rows/second)
with N rows.
```

The batch number, '##########', ties back to the Main Gateway logging [reader], as does the number of rows processed in the batch.

The reported rate in rows per second [XX] allows representation of the writers performance.

5 Property Considerations

5.1 Gate.RdrWtr.Description

Using a distinct user to login to the object server allows the JDBC2 gateway to be identified using the ‘users’ or connection details. The JDBC2 gateways shows as a Java Gateway [AppName=GATEWAY_J] in the catalog.connections table. This allows the gateway to be monitored and managed specifically using custom triggers.

e.g.

```
Gate.RdrWtr.Server   : 'AGG_V'
Gate.RdrWtr.Username: 'jdbc2gw'
Gate.RdrWtr.Password: 'netcool'
```

5.2 Java Memory

The Java memory is allocated and/or limited using the JDBC2 gateways environment file as follows.

File : $NCHOME/omnibus/gates/jdbc2/nco_g_jdbc2.env

```
JRE_OPTS='-Xmx4096m -Xms1024m'
```

When the amount of memory allocated to the java process needs to be increased or limited, set the '-Xmx' memory allocation to a suitable value for the system. Java 6 automatically allocates memory based on what memory is available. For Java 6 or above environments, the '-Xmx' option is used to limit the amount of memory the Java process uses.
It is recommended a minimum of 1Gb of RAM is allocated to Java for production JDBC2 gateways, but no more than 1.5Gb, for 32-bit Java.

To see how much memory can be allocated use the –version command with the system Java.
For example to see if 4Gb is available to Java.

```
java -Xmx4096m -Xms4096m -version
```

If the version details are returned, there is enough free memory to allocate 4Gb to Java.

5.3 Object Server failover and failback

The JDBC2 gateway uses the default behaviour of the Aggregation layer object server pair to failover and failback, and should connect to the Aggregation layers virtual object server, with an IPC timeout [Gate.Jdbc.SqlTimeout] of at least 300 seconds.

```
Gate.RdrWtr.Server   : 'AGG_V'
Gate.RdrWtr.Username: 'jdbcgw'
Gate.RdrWtr.Password: 'netcool'

Gate.Jdbc.SqlTimeout: 600
```

5.4 Data Forwarding

The key property settings for forwarding data are;

```
Gate.RdrWtr.IducFlushRate: 21
Gate.Jdbc.Connections    : 10
```

The IducFlushRate must not be lower than 10 seconds unless the system is specifically designed to manage such data forwarding. A value between 21 and 59 should be used.

Note: The MaxBatchSize is sized according to the typical load, and using the default is best.

The default setting of Gate.Jdbc.Connections is well below what the previous historical gateways used. The Gate.Jdbc.Connections setting can be increased to 10. As an example, the Oracle gateway used 7 database connections. The JDBC2 gateway processes the tables defined in the tabled replication file, in sequence.

By default; alerts.status, alerts.journal, alerts.details; Which means that all the connections are used for the current table being replicated, and other tables are not replicated until the current table replication transactions are completed. This can cause problems when large volumes of data are being replicated from one table, giving the impression that the gateway has stalled, when one of the tables is checked in the database. Referring to the STATS messages in the log file will indicate that the JDBC2 gateway is still successfully processing.

5.4.1 The jdbc.rdrwtr.tblrep.def file

The jdbc.rdrwtr.tblrep.def file controls how events are forwarded to the historical database.

Event forwarding can be more precisely controlled using a FILTER and the ‘AFTER IDUC DO’ command statement; e.g.

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'HistoricalReporting=1'
AFTER IDUC DO 'HistoricalReporting=2';

REPLICATE ALL FROM TABLE 'alerts.journal'
USING MAP 'JournalMap';
# EOF
```

5.5 Logging

The JDBC2 gateway uses a set of property settings to manage logging, with the default behaviour of logging to 6 rolling log files with date-timestamp file name extensions to the old log files.

```
# Maximum message log size in bytes
MaxLogFileSize: 1024000000

# Message level to log at error|warn|info|debug
MessageLevel: 'warn'

# Message log to write to
MessageLog: '$OMNIHOME/log/G_JDBC2.log'

# Maximum number of log files to retain plus current log file
MessageLogRollKeep: 5

# Command to process old rolled file - delete file if empty
# MessageLogRollProcess: ''

# Time of day (HHMM) to roll the log
# MessageLogRolltime: ''
```

5.5.1 NDE log settings

The Netcool Deployment Environment settings can be used to control the JDBC2 Gateway log file.

```
# 1024000 bytes (1024 KB) - 100Mb:
NDE_LOGFILE_MAXSIZE=102400000
export NDE_LOGFILE_MAXSIZE

NDE_LOGFILE_ROTATION_TIME=0000
export NDE_LOGFILE_ROTATION_TIME

NDE_LOGFILE_ROTATION_FORMAT=%Y%m%d-%H%M
export NDE_LOGFILE_ROTATION_FORMAT
```

5.6 Debug logging

The main property is ‘MessageLevel’ however, the full set of logging can be enabled using these settings;

```
###
# Debug logging
MessageLevel         : 'debug:debug'
MaxLogFileSize       : 1024000000
Gate.Java.Debug      : 'TRUE'
Gate.Mapper.Debug    : 'TRUE'
Gate.RdrWtr.Debug    : 'TRUE'
Gate.RdrWtr.LogOSSql : 'TRUE'
```

5.7 Dumpprops

Use the -dumpprops option to see the default property settings, and how the custom property file settings are translated.

For example, the default classpath setting:

```
nco_g_jdbc -propsfile G_JDBC2.props -dumpprops
```

6 Event Data Flow

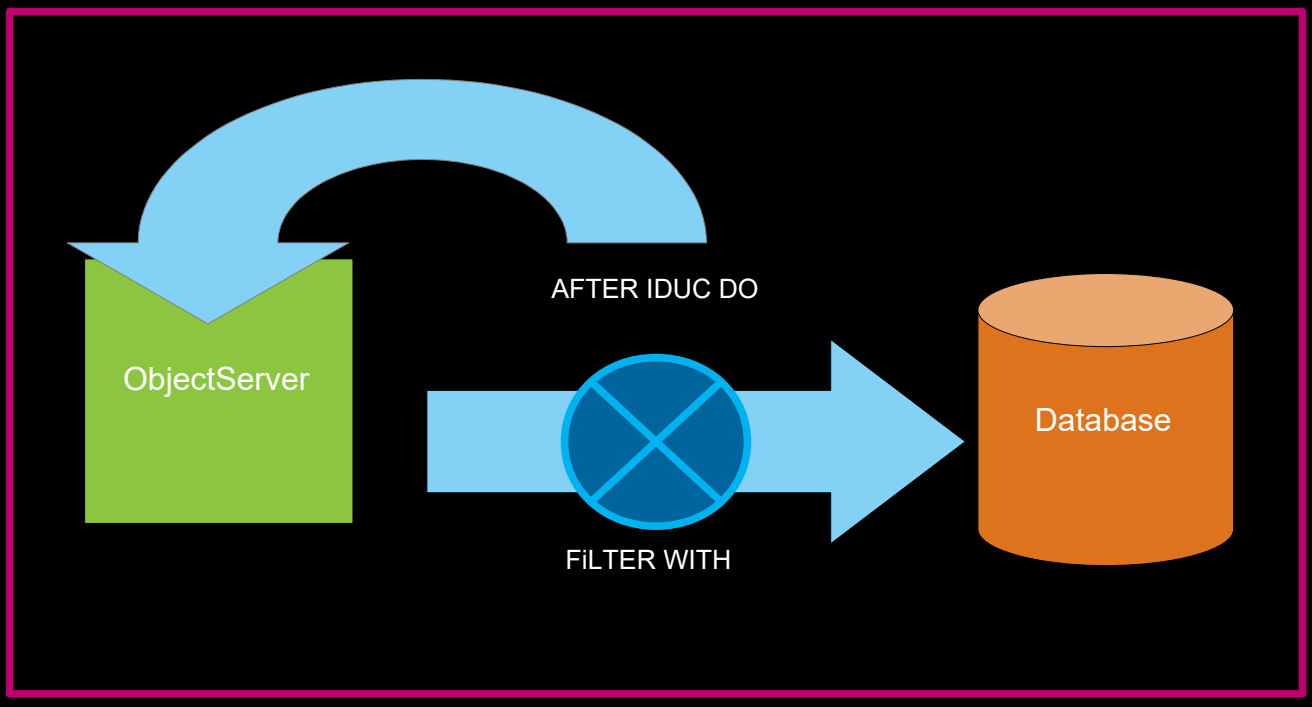
The design of the Target Database is dependent upon how the historical data is to be used.

6.1 Audit Database Configuration

Some customers require all data to be archived for a period of time, before being deleted. In such cases a DBA will control the main Archive Database with the Active Database being controlled locally. Periodically all the data in the Active Database is archived to the Archive Database and purged from the Active Database. This archive process can be performed using a number of techniques, and is outside the scope of this document.

6.2 Resident Database Configuration

More commonly the data is required to be resident, with the data being used to produce weekly, monthly or annual reports. In these cases, there is a need to ensure that the data is pruned before entering the Database. The best method to do this is through the use of a filter, update flag and object server triggers.



6.2.1 Standard Triggers Method

Use a flag such as `ReportGWFlag` to control the flow of data in the following triggers;

- new_row
- deduplication
- delete_clears

In each trigger set the `ReportGWFlag` to '1' as required to allow the required events to be processed by the JDBC2 gateway. The JDBC2 gateway properties file needs to refer to the flag in the FILTER and AFTER-IDUC table replication statement;

e.g.

```
FILTER WITH 'ReportGWFlag>0'
AFTER IDUC DO 'ReportGWFlag=2'
```

Control over which events are forwarded can be applied to the events Class, Manager and LastOccurrence. The setting of the ReportGWFlag can also be controlled within probe rules files.

The ReportGWFlag is added to the Aggregation Object Servers using nco_config or via the nco_sql command line:

```
alter table alerts.status add ReportGWFlag int;
```

and the AGG_GATE.map file updated for StatusMap:

```
    'ReportGWFlag' =          '@ReportGWFlag',
```

6.2.2 Temporal Trigger Method

The temporal trigger method allows the events to be checked periodically, to determine if they are persistent and important enough to be archived to the historical database.

In this case the trigger only actions events with the `ReportGWFlag` flag set to '1', based on whether the event can be archived and the JDBC2 gateway table replication statement defined as;

```
FILTER WITH 'ReportGWFlag>1'
AFTER IDUC DO 'ReportGWFlag=3'
```

This allows for a trigger to set the `ReportGWFlag` flag to '2', after the object server trigger has determined which events are for archiving.

The following trigger then forwards events to the Oracle database based on the age of the event and Severity;

```
create or replace trigger nc_jdbc_gateway_forward
group nc_jdbc_gateway
enabled true
priority 1
comment 'Set ReportGWFlag to forward events that meet specific requirements - OLD  High Severity events'
every 31 seconds
begin
        update alerts.status set ReportGWFlag = 2, where ReportGWFlag = 1 and Severity > 1 and FirstOccurrence
<= (getdate - 600);
end;
```

7 Common Issues

7.1 Status updates are not happening

There are no dedicated threads for alerts.status replication. The JDBC2 gateway uses a pool of threads to manage all the batches ready for forwarding to the database. The gateway works through all the replicated tables in sequence, therefore if there are many rows in alerts.journal or alerts.details, then these will be replicated before alerts.status is.

To resolve, check the gateways table replication file and comment out DETAILS replication, and examine the volume JOURNAL's being forwarded.

e.g.

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap';
REPLICATE ALL FROM TABLE 'alerts.journal'
USING MAP 'JournalMap';
# EOF
```

Increase the connections settings, for example:

```
# Threads available for forwarding data to the database
Gate.Jdbc.Connections: 10
```

Alternatively, try setting the number of connections to 1, to determine the throughput for a single thread, and to observe the databases behaviour:

```
Gate.Jdbc.Connections: 1
```

Increase the number of Connections to accommodate the expected peak load.

Monitor the gateways STATS: log file entries and overall memory and CPU usage for a period to check that the system is coping with the gateways event load.

Notes :

- The earlier historical gateways like the ODBC Gateway used 7 threads [`Gate.Jdbc.Connections:7]`.
- With these increased settings, the JDBC's gateway memory allocation will increase.
- Use the Gate.Jdbc.ResyncFilter to manage the start-up synchronisation behaviour

7.2 Event Re-Awaken issues

In a multitier environment events are cleared In the Aggregation layer, and then deleted.
If the event persists in the collection layer, the next update in the collection layer will cause the event [ServerName/ServerSerial] to be reinserted into the Aggregation layer, causing a reinsert error messages in the JDBC2 gateway.

Note : Reinserts in the aggregation layer can occur too, when the AGG_GATE gateway resynchronises, which causes the deletion of all events in an Object Server and subsequent event re-insertion.

The collection to aggregation layer replication statement is:

```
REPLICATE INSERTS, UPDATES
FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'SentToAgg = 0'
ORDER BY 'Serial ASC'
SET UPDTOINS CHECK TO FORCED
AFTER IDUC DO 'Tally = 0, SentToAgg = 1'
CACHE FILTER 'ServerName = \'COL_P_1\'';
```

which causes events from the collection Object Server to be reinserted into the aggregation layer.

To resolve reinsertion issues from the collection layer the events at the collection layer need to be removed sooner, rather than later, by setting the CollectionExpireTime field to a suitable value, or by using a lower default CollectionExpireTime value in the col_new trigger in the collection layer Object Servers.

You can also keep the cleared events at the aggregation layer for a longer period by modifying the delete_clears trigger, such that it takes longer than the CollectionExpireTime to remove the event.
e.g.

```
delete from alerts.status where Severity = 0 and StateChange < (getdate() - 600);
```

Given the maximum value of CollectionExpireTime is **600**.

The problem occurs since events are identified by the JDBC2 gateway using the events ServerName+ServerSerial, rather than the Identifier.

For alerts.status the important fields for successful replication are:

- Identifier or a unique Keyfield
- Serial
- ServerName
- ServerSerial

When Serial is not replicated, problems will arise for reinserted events, as although the ServerName/ServerSerial is the same, the Serial is different.

7.3 Recommended SQLSTATE settings

Development do not recommend any SQLSTATE settings other than the ones provided with the JDBC2 gateway. This is because the Oracle DBA should recommend SQLSTATE settings based on the available SQLSTATE messages the Oracle database is likely to return, and how the JDBC2 gateway is required to handle them.

The default setting are:

```
Gate.Jdbc.FatalErrors: '0A 42'
Gate.Jdbc.RetryErrors: '08 28 40 HYT'
```

The database vendor will have a list of SQLSTATEs and their meaning with respect to their database and should be able to provide you with a list of SQLSTATE messages for your database:

For example you are connecting to Oracle, you can contact Oracle Support for guidance or else the Oracle DBA.

Default Retry Errors:

08001 : SQL client unable to establish SQL connection
08002 : connection name in use
08003 : connection does not exist - SQL-02121
08004 : SQL server rejected SQL connection
08006 : connection failure
28000 : invalid authorization specification
40000 : transaction rollback - ORA-02091 .. 02092
40001 : serialization failure
40002 : integrity constraint violation
40003 : statement completion unknown

Default Fatal Errors:

0A000 : feature not supported - ORA-03000 .. 03099
0A001 : multiple server transactions
42000 : syntax error or access rule violation
ORA-00022
ORA-00251
ORA-00900 .. 00999
ORA-01031
ORA-01490 .. 01493
ORA-01700 .. 01799
ORA-01900 .. 02099
ORA-02140 .. 02289
ORA-02420 .. 02424
ORA-02450 .. 02499
ORA-03276 .. 03299
ORA-04040 .. 04059
ORA-04070 .. 04099

How to customise the Settings:

If the ORA-00370 message is seen in the logs and the JDBC2 gateway needs to retry to send the batch file when this issue was encountered, you would review the SQLSTATE messages for the database system as follows:

60000 : system errors
ORA-00370 .. 00429
ORA-00600 .. 00899
ORA-06430 .. 06449
ORA-07200 .. 07999
ORA-09700 .. 09999

and decide if all the other errors required in the retry.

If the answer was yes, then you would add 60 to the list of prefixes, for example:

```
Gate.Jdbc.RetryErrors: '06 08 28 40 HYT'
```

7.4 Oracle RAC URL example

The JDBC drivers are documented by the vendor, and these will include examples of how to configure the database URL's for high availability. Here is an example for the Oracle RAC URL:

The Oracle documentation states the JDBC URL syntax as:

```
Gate.Jdbc.Url: 'jdbc:oracle:thin:@(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST =
oraclehost1)(PORT = 1521))(ADDRESS = (PROTOCOL = TCP)(HOST = oraclehost2)(PORT =
1521)) (LOAD_BALANCE = yes)(CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME =
REPORTER)))'
```

With the usual single Oracle database syntax being:

```
Gate.Jdbc.Url: 'jdbc:oracle:thin:@oraclehost:1521:REPORTER'
```

For the Oracle RAC, the DB hosts are required as : oraclehost1, oraclehost2, etc.

```
Gate.Jdbc.Url: 'jdbc:oracle:thin:@oraclehost1,oraclehost2:1521:REPORTER'
```

In general it is best to ask the DBA of the system to provide the correct JDBC URL for the given driver.

7.5 Encrypting passwords and other properties

The JDBC2 gateway supports AES_FIPS encryption for encrypting passwords and other secret properties.

For example:

Generate the AES_FIPS key for the JDBC2 gateway:

```
mkdir $NCHOME/etc/security/keyfiles
$NCHOME/omnibus/bin/nco_keygen -l 256 -o $NCHOME/etc/security/keyfiles/jdbc2_gw.keyfile
```

Obtain the encrypted string using the AES_FIPS key:

```
$OMNIHOME/bin/nco_aes_crypt -c AES_FIPS -k $NCHOME/etc/security/keyfiles/jdbc2_gw.keyfile
netcool
@44:etSQg2r6xBRjOj0g8cYvQM+VM5oaYPEQBJQkAnBeH3k=@
```

Property settings.

```
ConfigKeyFile : '$NCHOME/etc/security/keyfiles/jdbc2_gw.keyfile'
Gate.Jdbc.Password: '@44:etSQg2r6xBRjOj0g8cYvQM+VM5oaYPEQBJQkAnBeH3k=@'
```

It is possible to use the same key for all the products in Netcool/OMNIbus, but it is usually easier for administration purposes, to use a unique key for each product. As this allows the key to be regenerated without affecting other products.

An example property file:

```
# AES_FIPS Encryption properties
ConfigKeyFile        : '$NCHOME/etc/security/keyfiles/jdbc2_gw.keyfile'
# Database connection user/password
Gate.Jdbc.Username   : 'reportdb'
Gate.Jdbc.Password   : '@44:etSQg2r6xBRjOj0g8cYvQM+VM5oaYPEQBJQkAnBeH3k=@'
# ObjectServer Connection properties
Name                 : 'G_JDBC2'
Gate.RdrWtr.Server   : 'AGG_V'
Gate.RdrWtr.Username : 'jdbc2gw'
Gate.RdrWtr.Password : '@44:etSQg2r6xBRjOj0g8cYvQM+VM5oaYPEQBJQkAnBeH3k=@'
```

Note: The nco_aes_crypt string is not the same, even for the same input string.

7.6 Creating an Object Server Gateway User

The JDBC2 gateway user needs to be a member of the Gateway group in the object server. The UserID (e.g. 300) needs to be available, and within the administration or gateway user ranges.

```
create user 'jdbc2gw' id 300 full name 'JDBC2 Gateway' password 'netcool';
go
alter group 'Gateway' assign members  'jdbc2gw';
go
```

If the JDBC2 gateway user needs to be tested from the JDBC2 Gateway server using nco_sql, add the user to the ISQL group to allow table data to be accessed from the command line.

```
alter group 'ISQL' assign members  'jdbc2gw';
go
```

7.7 Setting the Gateways Java path

The JDBC2 Gateways Java is picked up from the environment using the JAVA_HOME environment variable.

The UNIX find command can list all the locations of the nco_g_jdbc2.env files.
e.g.

```
find $NCHOME -name nco_g_jdbc2.env
```

File : $NCHOME/omnibus/platform/linux2x86/bin64/nco_g_jdbc2.env
File : $NCHOME/omnibus/gates/jdbc2/nco_g_jdbc2.env

The nco_g_jdbc2 wrapper script uses the $NCHOME/omnibus/gates/jdbc2/nco_g_jdbc2.env location by default.

Using the users environment.

```
File : BASHRC

JAVA_HOME=/opt/ibm-java-x86_64-80
export JAVA_HOME
echo '*** JAVA_HOME=$JAVA_HOME'
```

7.8 DB2 tracing batches

To trace problems with batches, you need to reduce the flow of data to more easily handled event flow.

- Reduce the replicated tables to 1
- Reduce the connections to 1
- Enabled JDBC Driver tracing

File : jdbc.rdrwtr.tblrep.def

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap' ;

# Commented out for tracing batch issue
#  REPLICATE ALL FROM TABLE 'alerts.journal'
#  USING MAP 'JournalMap';
```

File : G_JDBC.props

```
# Recommended settings
Gate.Jdbc.SqlTimeout  : 600
Gate.Jdbc.ResyncMode  : 'UNI'
# Debugging settings
Gate.Jdbc.MaxBatchSize: 1
Gate.Jdbc.Connections : 1

###
# Debug logging
MessageLevel          : 'debug:debug'
MaxLogFileSize         : 1024000000
Gate.Java.Debug        : 'TRUE'
Gate.Mapper.Debug      : 'TRUE'
Gate.RdrWtr.Debug      : 'TRUE'
Gate.RdrWtr.LogOSSql   : 'TRUE'
```

Directory: /opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java

File : DB2JccConfiguration.properties

```
db2.jcc.override.traceDirectory=/opt/nrv81/IBM/tivoli/netcool/omnibus/tmp
db2.jcc.override.traceFile=jcctrc
db2.jcc.override.traceFileAppend=true
db2.jcc.override.TraceLevel=TRACE_ALL
#EOF
```

Create trace file directory:

```
mkdir /opt/nrv81/IBM/tivoli/netcool/omnibus/tmp
```

To revert after troubleshooting:

```
cd /opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java
mv DB2JccConfiguration.properties DB2JccConfiguration.properties.not-used
```

Revert the gateways property file, for example:

```
# Debugging settings
Gate.Jdbc.Connections    : 10
# Debugging
MessageLevel             : 'info:info'
MaxLogFileSize           : 1024000000
Gate.Java.Debug          : 'FALSE'
Gate.Mapper.Debug        : 'FALSE'
Gate.RdrWtr.Debug        : 'FALSE'
Gate.RdrWtr.LogOSSql     : 'FALSE'
#EOF
```

File : jdbc.rdrwtr.tblrep.def

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap' ;

REPLICATE ALL FROM TABLE 'alerts.journal'
USING MAP 'JournalMap';
#EOF
```

7.9 Preventing StateChange Feedback

In environments where there are multiple integration gateways the StateChange field can trigger feedback through the AFTER-IDUC update. One way around this is to exclude gateways from the state_change triggers update of the StateChange field.

```
create or replace trigger state_change
group default_triggers
priority 1
comment 'State change processing for ALERTS.STATUS - modified for Integrations
gateways'
before update on alerts.status
for each row
begin
  if ( %user.is_gateway = false )
  then
        set new.StateChange = getdate();
  end if;
end;
```

7.10 Database Down Data loss

The JDBC2 gateway can run with the Database down but it cannot start, without the Database running. Therefore this section is for Database outages when the database is shutdown whilst the JDBC2 gateway is still running and is not shutdown before the Database returns fully, and the JDBC2 gateway has completed all its tasks.

To prevent the gateway from losing events ensure that the following properties are set appropriately for the outage.

```
Gate.Jdbc.FatalErrors      : '0A 42'
Gate.Jdbc.RetryErrors      : '08 28 40 HYT'
Gate.Jdbc.UnknownErrors    : 'RECONNECT'
Gate.Jdbc.ReconnectTimeout : 30
```

With the object server flush rate set to a value, event loss can be kept to a minimum during any planned outage. e.g.

```
Gate.RdrWtr.IducFlushRate: 21
```

During testing it was found that all events were preserved:

- G_JDBC started
- 10 events inserted into Object Server
- Database stopped
- 10 events inserted into Object Server
- Pause
- All JDBC2 gateway FILTER matching events deleted [e.g. Poll=1]
- Database started
- 10 events inserted into Object Server

After testing, all 30 inserted events existed in the Database.

Note: The setting of the Gate.RdrWtr.IducFlushRate value depends largely on the event rate and load on the object server. The chosen value should be an odd number that is less than 60 and greater than 10.

7.11 REPORTER_JOURNAL Unique Constraints

```
Oracle Error: [ORA-00001: unique constraint (REPORTER.SYS_SYSC001401) violated] - sqlcode[-1]
```

Confirm REPORTER_JOURNAL is causing the problem;
e.g.

```
select table_name from all_indexes where index_name='SYSC001401';
select * from all_ind_columns where index_name='SYSC001401';
```

View the restrictions on an Oracle table by examining the dba_cons_columns system table.
e.g.

```
select column_name from dba_cons_columns
where constraint_name in
(select constraint_name from dba_constraints where table_name = 'REPORTER_JOURNAL');
```

You can use an key on the REPORTER_JOURNAL table to allow multiple entries in the same second.

```
-- Create the Table REPORTER_JOURNAL

DROP TABLE reporter_journal CASCADE CONSTRAINTS;

CREATE TABLE reporter_journal (
JournalKey NUMBER(24) NOT NULL,
Serial NUMBER(16) NOT NULL,
UserID NUMBER(16) NOT NULL,
Chrono DATE NOT NULL,
Text1 VARCHAR2(255) NULL,
Text2 VARCHAR2(255) NULL,
Text3 VARCHAR2(255) NULL,
Text4 VARCHAR2(255) NULL,
Text5 VARCHAR2(255) NULL,
Text6 VARCHAR2(255) NULL,
Text7 VARCHAR2(255) NULL,
Text8 VARCHAR2(255) NULL,
Text9 VARCHAR2(255) NULL,
Text10 VARCHAR2(255) NULL,
Text11 VARCHAR2(255) NULL,
Text12 VARCHAR2(255) NULL,
Text13 VARCHAR2(255) NULL,
Text14 VARCHAR2(255) NULL,
Text15 VARCHAR2(255) NULL,
Text16 VARCHAR2(255) NULL,
ServerName VARCHAR2(64) NOT NULL,
ServerSerial NUMBER(16) NOT NULL
)
/

-- Create the Indexes for REPORTER_JOURNAL

CREATE INDEX RJ_INDEX
ON reporter_journal (
ServerSerial,
ServerName
)
/

-- Create the Constraints for REPORTER_JOURNAL

ALTER TABLE reporter_journal
ADD PRIMARY KEY (
JournalKey,
ServerSerial,
ServerName,
UserID,
Chrono
)
/


-- The rep_journal_key trigger automatically populates the
-- JournalKey field in journal table with an incremental integer
-- to avoid unique constraint violations from journal inserts


CREATE OR REPLACE TRIGGER rep_journal_key
BEFORE INSERT ON reporter_journal
FOR EACH ROW
BEGIN
SELECT journal_seq.NEXTVAL
INTO :new.JournalKey
FROM dual;
END;
/
```

7.12 Fast Forwarding an Object Servers Serial

If the JDBC2 gateways source object server needs to be rebuilt, the object servers Serial needs to be set to a value greater than the ServerSerial in the historical database. An alternative to this, is to update the ServerName field in the database, to a string other than the object servers ServerName. For example AGG_P can be renamed AGG_P01, and AGG_B can be renamed to AGG_B01. Such fixes are required because the ServerName, ServerSerial are key fields in the historical reporting database, and events from an object server with the same ServerName and ServerSerial will be discarded due to these fields unique constraints. Discuss updating the ServerName with the Database Administrator before making any updates.

7.12.1 Increasing an Object Server Serial using SQL

The Netcool/OMNIbus 8.1 Object Server SQL supports an ALTER TABLE command for increments.

```
ALTER TABLE alerts.status SET INCR <value>
```

Check the maximum ServerSerial in the database.

```
select max( ServerSerial ) from reporter_status where ServerName = 'NCOMS';
315,983
```

Update the Serial in the new Object Server to a value larger than the maximum value returned from the select.

```
1> ALTER TABLE alerts.status SET INCR 320000
2> go
```

Check the Serial is being used in the Object Server.

```
select max(Serial) from alerts.status;
go
320021
```

7.12.2 Increasing an Object Server Serial using nco_dbinit

Dump the current Object Server

The alternative method is to use nco_osreport to capture an object servers SQL files within a test environment. By preventing the dumping of the dynamic tables, the SQL file size and complexity is reduced.

```
mkdir $NCHOME/OSREPORT
cd $NCHOME/OSREPORT
nco_osreport -server AGG_P -directory `pwd` -user root -password '' -norowdata
alerts.status,alerts.journal,alerts.details
```

Set the object Server Serial

The alertsdatafile.sql is edited to set the highest Serial value in the object servers alerts.status table.

The INSERT statement is set to a value that exceeds the databases REPORTER_STATUS table.

```
cd $NCHOME/OSREPORT
vi alertsdatafile.sql
insert into alerts.status ( Identifier, Serial, Severity, Summary ) values
( 'initial-event', 50000, 0, 'initialising event' );
```

In this example the Serial needs to be fast forwarded to 50,000.
This value is determined by checking the target database, and the REPORTER_STATUS table.

```
select max(ServerSerial) from  REPORTER_STATUS where ServerName = 'AGG_P';
```

Building the Object Server

The new object server can be built on the test server using the nco_dbinit command and the object servers SQL files.

```
nco_dbinit \
        -force              \
        -server             AGG_P \
        -alertsdata         \
        -alertsdatafile     ./alertsdata.sql \
        -applicationfile    ./application.sql \
        -automationfile     ./automation.sql \
        -desktopfile        ./desktop.sql \
        -securityfile       ./security.sql \
        -systemfile         ./system.sql
```

Start the object server and check the alerts.status tables Serial's and that the other tables are configured as expected.

8 Full list of property settings

8.1 General properties

```
# Gateway short name
# Name: 'G_JDBC2'

# Encryption key location - AES_FIPS only
# ConfigKeyFile: ''

# Historic resync
# Gate.HistoricResync: 'true'

# Gateway type
Gate.Type: 'jdbc2'

# Gateway base package
# Gate.PackageBase: 'com.ibm.tivoli.netcool.integrations.gateway'

# Allow unknown properties in properties file (TRUE|FALSE)
# Props.AllowUnknown: 'false'
```

8.2 File properties

```
# Map file location
Gate.MapFile: '$OMNIHOME/gates/jdbc2/reporting.jdbc2.map'

# Table replication file location
Gate.RdrWtr.TblReplicateDefFile: '$OMNIHOME/gates/jdbc2/jdbc2.rdrwtr.tblrep.def'
```

8.3 Cache properties

```
# Resynchronization mode (UNI|BI|AUTO)
# Gate.Cache.ResyncMode: 'AUTO'

# Suppress initial resynchronization deletes (TRUE|FALSE)
# Gate.Cache.ResyncSuppressInitialDeletes: 'FALSE'

# Suppress resynchronization deletes (TRUE|FALSE)
# Gate.Cache.ResyncSuppressDeletes: 'FALSE'

# Generic script for additional gateway functionality.
# Gate.Cache.GenericScript: '${OMNIHOME}/gates/jdbc2/jdbc2.generic.js'

# Map to use for alert notifications
# Gate.Cache.NotificationMap: '${OMNIHOME}/gates/jdbc2/jdbc2.notification.js'
```

8.4 Conversion table properties

```
# Conversions direction (BOTH|FORWARD|REVERSE)
# Gate.ConversionsDirection: 'BOTH'

# Table to use for value conversions
# Gate.ConversionsTableName: ''
```

8.5 Debugging properties

```
# Enable/Disable Java Gateway debug messages
# Gate.Java.Debug: 'TRUE'

# Enable/Disable reader/writer debug messages
# Gate.RdrWtr.Debug: 'TRUE'

# Enable/Disable mapper debug messages
# Gate.Mapper.Debug: 'TRUE'

# Log all SQL sent to ObjectServer in debug mode
# Gate.RdrWtr.LogOSSql: 'FALSE'
```

8.6 Logging properties

```
# Maximum message log size
# MaxLogFileSize: 0

# Message level to log at
MessageLevel: 'warn'

# Message log to write to
MessageLog: '$OMNIHOME/log/G_JDBC2.log'

# Maximum number of old log files to retain
# MessageLogRollKeep: 5

# Command to process old rolled file - delete file if empty
# MessageLogRollProcess: ''

# Time of day (HHMM) to roll the log
# MessageLogRolltime: ''
```

8.6.1 Trace mode properties

```
# Trace mode (READ|WRITE|NONE)
# Gate.TraceMode: 'NONE'

# Trace file to read/write
# Gate.TraceFile: ''
```

8.7 JDBC properties

These properties relate to the JDBC driver connectivity [writer].

8.7.1 JDBC Connection properties

```
# JDBC gateway mode (AUDIT|REPORTING)
# Gate.Jdbc.Mode: 'REPORTING'

# JDBC database connection count
# Gate.Jdbc.Connections: 3

# Resync mode (NONE|UNI|BI|AUTO)
# Gate.Jdbc.ResyncMode: 'AUTO'

# JDBC connection URL
# Gate.Jdbc.Url: ''

# JDBC Driver
# Gate.Jdbc.Driver: ''

# JDBC username
# Gate.Jdbc.Username: ''

# JDBC password
# Gate.Jdbc.Password: ''

# JDBC driver-specific user property name
# Gate.Jdbc.DriverUserPropertyName: 'user'

# JDBC driver-specific password property name
# Gate.Jdbc.DriverPasswordPropertyName: 'password'

# Java system properties file
# Gate.Jdbc.JavaSystemPropsFile: ''

# JDBC connection properties file
# Gate.Jdbc.JdbcPropsFile: ''
```

8.7.2 JDBC Connection initialization

```
# JDBC connection initialization string
# Gate.Jdbc.InitializationString: ''

# Initialization timeout (sec)
# Gate.Jdbc.InitializationTimeout: 10

# Initialize the configured initialization string in all JDBC sessions
# Gate.Jdbc.InitializeAllSessions: 'TRUE'
```

8.7.3 Reconnection properties

```
# JDBC preconnection wait (millisec)
# Gate.Jdbc.PreconnectionWait: 1000

# JDBC database reconnection timeout
# Gate.Jdbc.ReconnectTimeout: 30
```

8.7.4 Target database tables

```
# Target table for alerts.status
# Gate.Jdbc.StatusTableName: 'REPORTER_STATUS'

# Target table for alerts.journal
# Gate.Jdbc.JournalTableName: 'REPORTER_JOURNAL'

# Target table for alerts.details
# Gate.Jdbc.DetailsTableName: 'REPORTER_DETAILS'
```

8.7.5 AUDIT Mode properties

```
# ActionCode field in audit mode
# Gate.Jdbc.ActionCodeField: 'ACTIONCODE'

# ActionTime field in audit mode
# Gate.Jdbc.ActionTimeField: 'ACTIONTIME'
```

8.7.6 Field properties

```
# Status table column name to use for target identifier tracking
# Gate.TargetIdField: ''

# DeletedAt field for alerts.details
# Gate.Jdbc.DeletedAtField: 'DELETEDAT'

# Fields to ignore when de-duplicating updates
# Gate.Jdbc.DupIgnore: 'LastModified StateChange ActionCode ActionTime'

# Partitioning field
# Gate.Jdbc.PartitioningField: ''

# Target Serial field for alerts.status
# Gate.Jdbc.SerialField: ''

# Target ServerName field for alerts.status
# Gate.Jdbc.ServerNameField: 'SERVERNAME'

# ServerSerial field for alerts.details
# Gate.Jdbc.ServerSerialField: 'SERVERSERIAL'
```

8.7.7 Database codes

```
# SQLSTATE prefixes that denote that a batch should be retried
# Gate.Jdbc.RetryErrors: '08 28 40 HYT'

# SQLSTATE prefixes considered fatal when processing a batch
# Gate.Jdbc.FatalErrors: '0A 42'

# How to handle unknown SQL errors (RECONNECT|RETRY|ABORT|IGNORE)
# Gate.Jdbc.UnknownErrors: 'RECONNECT'
```

8.7.8 Important JDBC properties

```
# SQL operation timeout (sec)
# Gate.Jdbc.SqlTimeout: 600

# Suppress writing of alert deletes
# Gate.Jdbc.SuppressDeletes: 'FALSE'

# Resync filter to find open events in target
# Gate.Jdbc.ResyncFilter: ''
```

8.7.9 Other JDBC properties

```
# JDBC statement maximum batch row count
# Gate.Jdbc.MaxBatchSize: 250

# Insert status table rows before journal and details rows
# Gate.Jdbc.OrderedWrites: 1

# Test the gateway against the connected database.
# Gate.Jdbc.TestCount: 0
```

8.8 Object Server properties

These properties relate to the object server connectivity [reader].

```
# ObjectServer location
# Gate.RdrWtr.Server: 'localhost:4100'

# Log in Id for ObjectServer
# Gate.RdrWtr.Username: 'root'

# Log in password for ObjectServer
# Gate.RdrWtr.Password: ''

# Status table name
# Gate.RdrWtr.StatusTableName: 'alerts.status'

# Journal table name
# Gate.RdrWtr.JournalTableName: 'alerts.journal'

# Details table name
# Gate.RdrWtr.DetailsTableName: 'alerts.details'

# IDUC Flush Rate
# Gate.RdrWtr.IducFlushRate: 0

# Size of the SQL command buffer
# Gate.RdrWtr.BufferSize: 25

# Reconnection retry timeout in seconds
# Gate.RdrWtr.ReconnectTimeout: 30
```

8.9 HTTP Command port properties

```
# Port to use for the HTTP command port.
# HttpServer.Port: 0

# Authentication realm for the HTTP command port.
# HttpServer.Realm: 'netcool'

# Maximum request content size we will handle
# HttpServer.RequestMax: 2097152

# Document root for static files
# HttpServer.Root: '${OMNIHOME}/www'

# Username for the HTTP command port.
# HttpServer.Username: 'netcool'

# Password for the HTTP command port.
# HttpServer.Password: ''
```