Tivoli Netcool Support's Guide to the JDBC Gateway

by

Jim Hutchinson

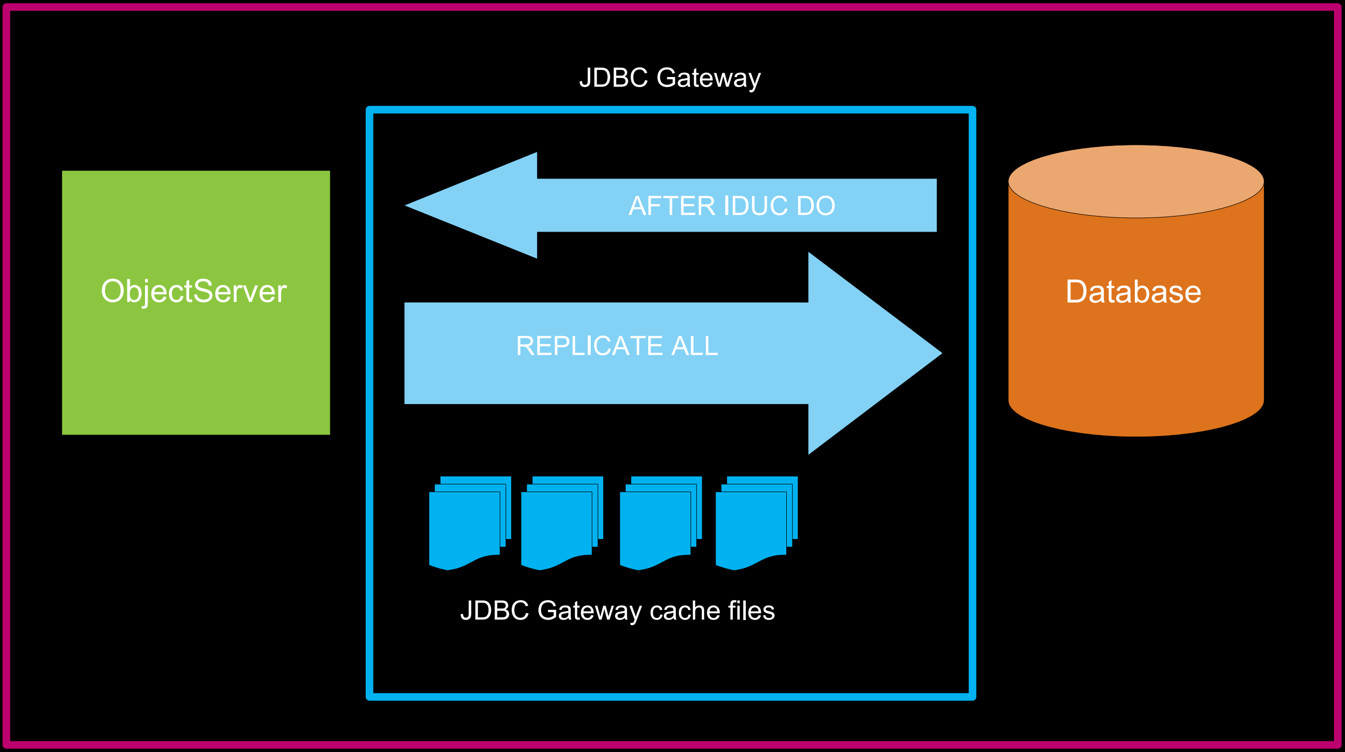
Document release: 3.3

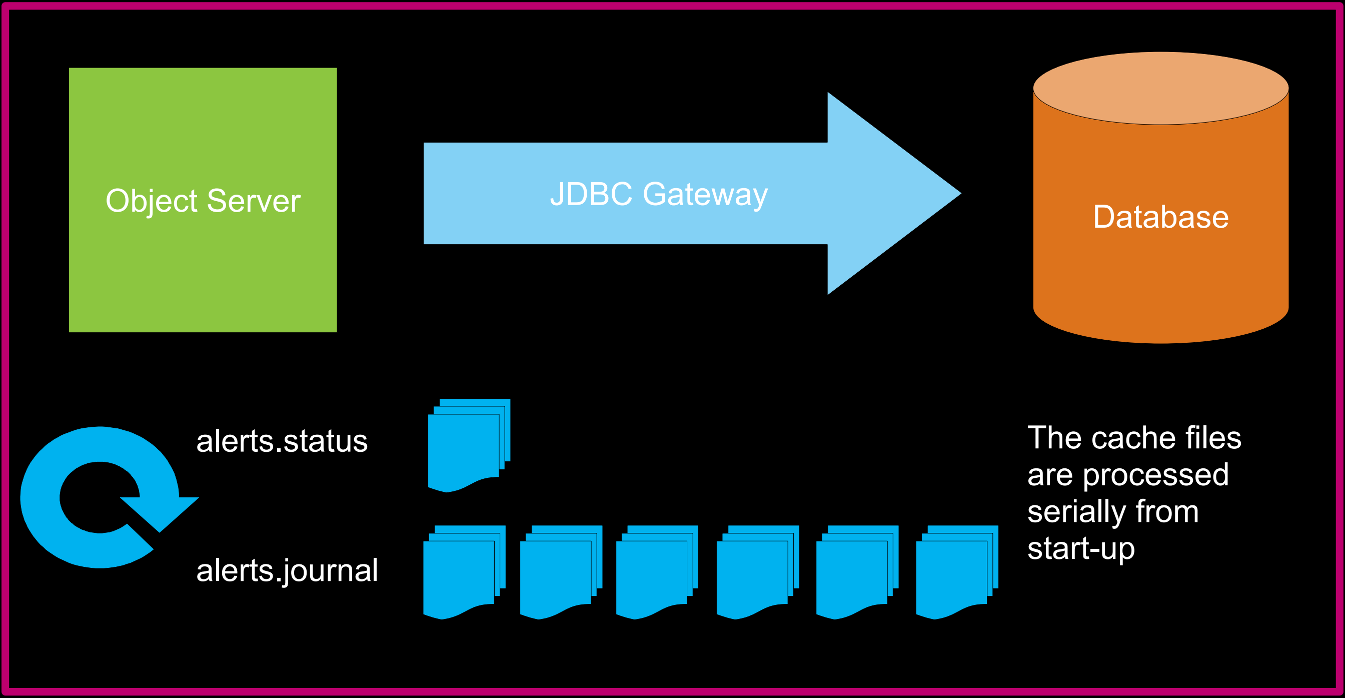
Table of Contents

1 Introduction

1.1 Overview

The JDBC 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 provider, 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 JDBC 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>

2.2 Database Schemas

The JDBC database schemas are provided as separate packages and not included with the JDBC gateway configuration files. Always download the latest schema when applying the JDBC Gateway, and check the schema for your database against the current SQL files provided in the package.

Two schema types are provided:

- REPORTING
- AUDIT

Ensure that the correct mapping used for your chosen JDBC Gateway mode, and use the example files provided to create a set of JDBC Gateway files suitable for your installation.

Whilst the REPORTING schema includes a full set of tables suitable for the Netcool/Reporter reporting tool, and the Netcool/OMNIbus TCR reports, the reporter_status and reporter_journal tables are the most useful, and can be applied alone, without the other tables, with custom reports

If a custom reporting tool is to be used, consider the fields required in the alerts.status that will need to be added to allow the various states of the alarms to be captured, and reported on.
e.g.
AcknowledgedTime
AcknoweldgedBy
TicketOpenedTime
TicketClosedTime
ClearedTime
ClearedBy
MaximumSeverity
EscalationTime

3 JDBC 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 key files and directories are;

- G_JDBC.props
- jdbc.map
- jdbc.rdrwtr.tblrep.def
- jdbc.startup.cmd

3.1 G_JDBC.props

You can use the '-dumpprops' command to determine all of the available properties, that will include the JDBC gateway specific property settings as well as the library properties.

An example for a REPORTING Oracle 11g historical database is given below;

```
Name: 'G_JDBC'
Gate.RdrWtr.Description: 'Oracle JDBC gateway'
Gate.Jdbc.Mode: 'REPORTING'
# Tables to replicate to
Gate.Jdbc.StatusTableName: 'REPORTER_STATUS'
Gate.Jdbc.JournalTableName: 'REPORTER_JOURNAL'
Gate.Jdbc.DetailsTableName: 'REPORTER_DETAILS'
# JDBC target server
Gate.Jdbc.Url: 'jdbc:oracle:thin:@server.uk.ibm.com:1521:ORACLE11G'
Gate.Jdbc.Driver: 'oracle.jdbc.driver.OracleDriver'
Gate.Jdbc.Username: 'reporter'
Gate.Jdbc.Password: 'reporter'
# Connection settings
Gate.RdrWtr.Server: 'AGG_V'
Gate.RdrWtr.Username: 'root'
Gate.RdrWtr.Password: ''
Gate.Jdbc.ReconnectTimeout: 30
Gate.Jdbc.InitializationString: ''
# Configuration files
Gate.StartupCmdFile: '$NCHOME/omnibus/gates/G_JDBC/jdbc.startup.cmd'
Gate.MapFile: '$NCHOME/omnibus/gates/G_JDBC/jdbc.map'
Gate.RdrWtr.TblReplicateDefFile: '$NCHOME/omnibus/gates/G_JDBC/jdbc.rdrwtr.tblrep.def'
MessageLog: '$NCHOME/omnibus/log/G_JDBC.log'
MaxLogFileSize: 102400
# Synchronisation
Gate.Jdbc.ResyncMode: 'UNI'
Gate.Jdbc.ResyncFilter: 'LastOccurrence > (getdate - 3600)'
# Other settings - default
Gate.RdrWtr.FailbackEnabled: FALSE
Gate.RdrWtr.FailbackTimeout: 30
Gate.Transfer.FailoverSyncRate: 60
Gate.G_JDBC.FailbackEnabled: FALSE
Gate.G_JDBC.FailbackTimeout: 30
# Debugging - Comment out for Production usage
MessageLevel: 'info'
Gate.G_JDBC.Debug: TRUE
Gate.Java.Debug: TRUE
Gate.Mapper.Debug: TRUE
Gate.NGtkDebug: 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 JDBC gateways supports;

- audit.jdbc.map
- reporting.jdbc.map

The mapping file is referred to by the table replication file, start-up TRANSFER commands, and ad hoc command line [nco_sql] TRANSFER commands.

The mapping file conforms to the 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 for your historical database. For instance 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. Therefore the mapping file would need to be edited to meet the requirements of the target database fields.

3.3 *jdbc.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 | INSERTS, UPDATES, DELETES}
      FROM TABLE sourcetable
      USING MAP mapname
      [FILTER WITH filter_clause]
      [INTO destinationtable ]
      [ ORDER BY column_name ]
      [WITH NORESYNC][RESYNC DELETES FILTER condition]
      [SET UPDTOINS CHECK TO {ENABLED|DISABLED|FORCED}]
      [AFTER IDUC DO command]
      [ CACHE FILTER condition]
```

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 your historical database and the type of events being forwarded.

3.4 *jdbc.startup.cmd*

The start-up commands main usage is to allow static object server tables to be transferred to the historical database when the JDBC gateway is started.

The default is to transfer no tables, however, a few transfer examples are given which are used in the Netcool/Reporter and TDW integration;

```
#TRANSFER FROM 'alerts.conversions' TO 'REPORTER_CONVERSIONS' DELETE USING TRANSFER_MAP ConversionsMap;
#TRANSFER FROM 'alerts.objclass' TO 'REPORTER_CLASSES' DELETE USING TRANSFER_MAP ObjectClassesMap;
#TRANSFER FROM 'master.groups' TO 'REPORTER_GROUPS' DELETE USING TRANSFER_MAP GroupsMap;
#TRANSFER FROM 'master.members' TO 'REPORTER_MEMBERS' DELETE USING TRANSFER_MAP MembersMap;
#TRANSFER FROM 'master.names' TO 'REPORTER_NAMES' DELETE USING TRANSFER_MAP NamesMap;
```

By default the mappings are provided for the default static tables. There may be some updates required depending upon the target database, due to database restrictions. Custom static tables can also be transferred upon start-up or via the nco_sql command line.

The alerts.objclass table is a legacy Object Server table, and is no longer populated by default; Class definitions can be obtained from the alerts.conversions table.

To create a custom classes table create a new table as a copy of REPORTER_COVERSIONS and use the 'VIA FILTER' clause to TRANSFER only Classes.

For example, in Oracle:

```
-- Create table to store Groups for Netcool Conversions
DROP TABLE custom_classes CASCADE CONSTRAINTS;

CREATE TABLE custom_classes (
        Conversion_Key  VARCHAR2(255)   NOT NULL,
        Colname         VARCHAR2(255)   NOT NULL,
        Value           NUMBER(16)      NOT NULL,
        Conversion      VARCHAR2(255)   NOT NULL,
        PRIMARY KEY(Conversion_Key)
)
/
```

Add a custom mapping to jdbc.map:

```
CREATE MAPPING CustomConversionsMap
(
    'CONVERSION_KEY'    = '@KeyField',
    'COLNAME'           = '@Colname',
    'VALUE'             = '@Value',
    'CONVERSION'        = '@Conversion'
);
```

Then use the TRANSFER statement with VIA FILTER:
TRANSFER FROM 'alerts.conversions' TO '**CUSTOM_CLASSES**' VIA FILTER '**Colname** = \'Class\'' DELETE USING TRANSFER_MAP **CustomConversionsMap**;

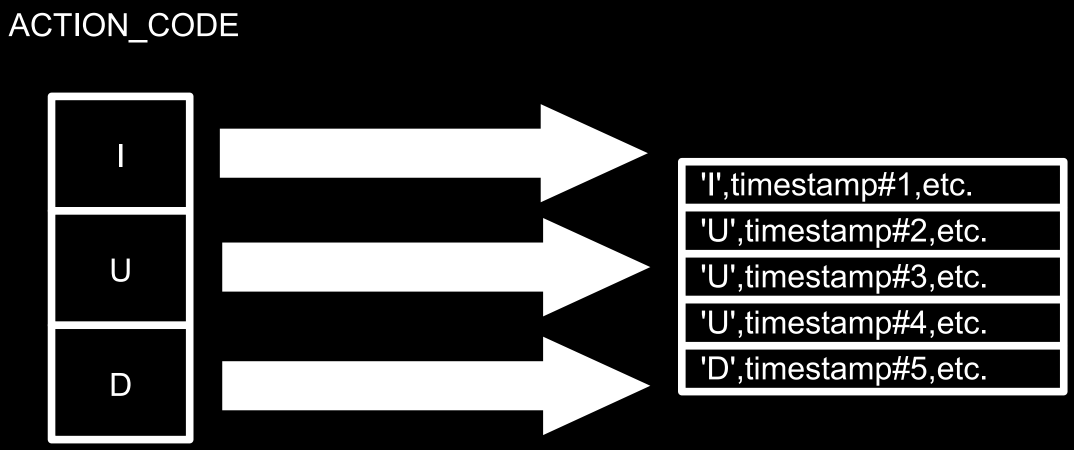
Notice that the new database table has the same field name as the Object Server when it is used in the filter.

3.5 Gate.Jdbc.Mode

The JDBC 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 JDBC Gateway. Each mode requires a specific database configuration and a separate downloadable package is provided to assist in the target schema's configuration.

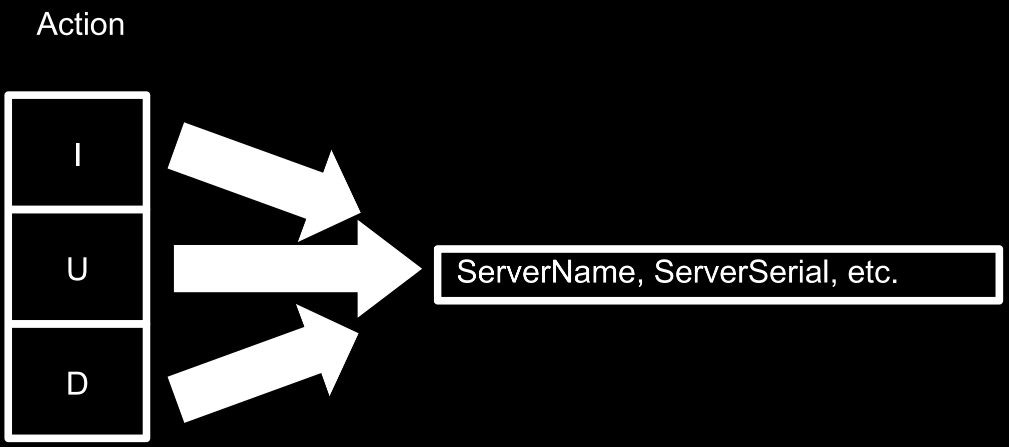
AUDIT Mode Overview
The AUDIT mode uses two functions unique to the AUDIT mode to populate the database. The STATUS rows are unique based on the ActionCode, ActionTime, ServerName and ServerSerial fields.

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.



3.5.1 AUDIT Mode

The differences between the AUDIT mode and REPORTING mode are seen in the JDBC 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 JDBC 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);
```

3.5.2 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 JDBC 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 JDBC gateway logs performance information which is enabled when the messagelevel is set to informational. The main documentation describes how these statistics are calculated and how they are used.

However, this data is not intuitive. It is recommended that if performance issues are seen that a set number of events are inserted into a test system, and the statistics are reviewed there, with the actual loading being taken as a measure of the systems performance.

Here are two statistics entries for 50,000 event being updated:

```
Information: I-GJA-000-000: [ngjava]: G_JDBC: Thread-2: STATS: f60c894f-eb73-4855-a72c-85eb2d82f014
Batch write time 36147 ms (1383.627963593106 rows/second)
Information: I-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-1: STATS: f60c894f-eb73-4855-a72c-85eb2d82f014
Batch execution time: 37651 ms (1328.3578125414995 rows/second)
```

The Batch write time refers to the time to create the cache file.
The Batch execution time refers to the time taken to send the data to the database.
The approximate number of events can be determined and checked by these values:
e.g.

```
36147 ms *  1383 ~ 50k
37651 ms *  1328 ~ 50k
```

In this example the batch size was the default 250 [Gate.Jdbc.MaxBatchSize] and there were no errors seen in the log file so there were no delays in the events being written to the database. Therefore for this system it takes around 60 seconds [36147 + 37651 ms] to perform the task of writing the cache file, and sending the updates to the database, if the writes are done in sequence. In the gateways log file on one pool thread [pool-1-thread-1] was visible for writing to the database. Performance for writing could be improved for writing using the Gate.Jdbc.Connections property, which is set to 3 by default. The Oracle gateway, for example, used 7 writer Connections by default, so increasing Gate.Jdbc.Connections to 10 is reasonable.

Note that although the Gate.Jdbc.MaxBatchSize is 250, only one cache file is written. The batching is performed within the cache files and will affect how the gateway sends data to the database. If there is a build of cache files over time, it means that there are problems writing to the database, or that the gateway is unable to find enough time to process the batch files, due to loading at the object server.

Example 10,000 INSERTED events

```
Information: I-GJA-000-000: [ngjava]: G_JDBC: Thread-2: STATS: 1bd5a159-1b69-4f3a-896e-a175b970be30
Batch write time 6465 ms (1546.6357308584686 rows/second)
Information: I-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-1: STATS: 1bd5a159-1b69-4f3a-896e-a175b970be30
Batch execution time: 5096 ms (1962.12715855573 rows/second)
```

Example 10,000 UPDATED events

```
Information: I-GJA-000-000: [ngjava]: G_JDBC: Thread-2: STATS: 7f4df28a-cf07-4f2f-a720-15e834fb32f3
Batch write time 4633 ms (2158.428663932657 rows/second)
Information: I-GJA-000-000: [ngjava]: G_JDBC: pool-1-thread-1: STATS: 7f4df28a-cf07-4f2f-a720-15e834fb32f3
Batch execution time: 7064 ms (1415.6285390713476 rows/second)
```

In the two examples it is apparent that the time to perform the two tasks was comparable.

It was noted that the logging happened within the IDUC cycle [flush], that the events were processed in a single cache file, and that all the events were processed. Events were only pushed to the database where the mapping files requirements were met; For the updates both the gateway flag and LastOccurrence were updated.

5 Property Considerations

5.1 Gate.RdrWtr.Description

It is important to set the gateway description and if necessary use a distinct user to login to the object server so that the JDBC gateway can be identified using the ‘users’ or connection details. This allows the gateway to be monitored and managed accurately using custom triggers.

e.g.

```
Gate.RdrWtr.Server: 'AGG_V'
Gate.RdrWtr.Description: 'jdbc_db2_gateway'
Gate.RdrWtr.Username: 'jdbcgw'
Gate.RdrWtr.Password: 'netcool'
```

5.2 Java Memory

The Java memory is allocated and/or limited using the JDBC gateways property file as follows;

```
Gate.Java.Arguments: '-Xmx2048m'
```

When the log file reports:

```
Error: [ngjava]: Failed to create a row entry for mapped table row data object instance
within embedded JVM. (12:Not enough space)
```

The amount of memory available to the java process needs to be increased or the volume of alarms the JDBC gateway uses reduced. Java 6 automatically allocates memory based on what memory is available. Therefore with Java 6 or above environments, the '-Xmx' option is used to limit the amount of memory the non-native java process uses.
It is recommended that a minimum of 1Gb of RAM is allocated to Java for production JDBC gateways, but no more than 1.5Gb, if the Java is a 32-bit Java.
To see how much memory can be allocated use the –version command;
e.g.

```
java -Xmx1500m -version
```

32-Bit Java General guidance:
For a 32-bit java process no more than twice the maximum allowed memory should be allocated to the non-native process, as the native process will require what is left of the memory to operate.

For example:
AIX: **1500 mb**
Solaris: **1500 mb**
Linux x86: **1500 mb**
HPUX: **700 mb**
zLinux: **700 mb**
Windows: **500 mb**

5.3 Object Server Ipc.StackSize

For large volumes of inserts|updates the object server needs to have the Ipc.StackSize increased.

e.g.

```
vi $NCHOME/etc/NCOMS.props
#Ipc.StackSize: 67584
# Increased for large data inserts - 4*standard
Ipc.StackSize: 524288
:wq
```

The default stack size is between 65k to 128k depending upon the version of Netcool/OMNIbus. Increasing the stack size limits the number of connections that can be established to the Object Server, since each connection requires its own stack.

i.e. stacksize * number of clients connected = memory used

Maximum memory sizes for Object Server range from 2Gb to 3.5 Gb for 32-bit processes.
Otherwise, for 64-bit processes, the maximum memory allocated will be related to usage and what is available on the platform.

The current memory size of the Object Server needs to be measured and taken into account, before increasing the stack size property. The Ipc.StackSize in the gateway and object server should be set to the same value.

5.4 Object Server failover and failback

The JDBC 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 of at least 300 seconds.

```
Gate.RdrWtr.Server: 'AGG_V'
Gate.RdrWtr.Username: 'jdbcgw'
Gate.RdrWtr.Password: 'netcool'
Ipc.Timeout: 600
```

5.5 Java CLASSPATH : Gate.Java.ClassPath

The JDBC Gateway uses the $NCHOME/omnibus/gates/java directory to create its CLASSPATH by default. Therefore, usually all that is required is for the required Database jar files be added to the gates java directory.

However, when multiple versions of the Database jar files are required; for example when there are multiple instances of the JDBC Gateway connecting to multiple Database versions, the Gate.Java.ClassPath needs to be set specifically. In such cases a gateway specific jars directory can be used to hold the Database specific jar files and the Gate.Java.ClassPath set accordingly.

e.g.

```
Gate.Java.ClassPath:
'/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/ngtktk.jar:/opt/nrv81/IBM/tivoli/ne
tcool/omnibus/gates/java/ngjava.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/
nco_g_jdbc.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/db2jcc_license_cu.jar
:/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/db2jcc4.jar'
```

5.6 Data Forwarding

The key property settings for forwarding data are;

```
Gate.RdrWtr.IducFlushRate: 11
Gate.Jdbc.MaxBatchSize: 250
Gate.Jdbc.Connections: 12
Gate.RdrWtr.UseBulkInsCmd: FALSE
```

The IducFlushRate should not be lower than 10 seconds unless the system is specifically designed to manage aggressive data forwarding.

The MaxBatchSize should be sized according to the typical load, or to manage event floods, otherwise the default setting should suffice.

The default setting of Gate.Jdbc.Connections is below what previous historical gateways used. The setting can be increased to 10, for example, to be comparable with the Oracle gateway it would be set to 7. The JDBC gateway processes the replicated tables in sequence, after first performing the start-up command files TRANSFER commands. This means that all the connections are used for the current table being replicated, and other tables are not replicated until the current 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 another table is checked in the database. Referring back to the STATS messages in the log file will indicate that the JDBC gateway is still successfully processing.

Using Gate.RdrWtr.UseBulkInsCmd set to TRUE can improve performance, but may cause problems if there are differences in languages and character sets between the Object Server and database.

5.6.1 The jdbc.rdrwtr.tblrep.def file

The jdbc.rdrwtr.tblrep.def file controls how events are forwarded to the historical database. Typically the file will use the default settings.

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
```

5.6.2 Journal considerations

The forwarding of journals can be controlled further using the following properties;

```
Gate.Mapper.ForwardHistoricJournals: TRUE
Gate.RdrWtr.IgnoreStatusFilter: TRUE
```

Set IgnoreStatusFilter to TRUE if all journals need to be forwarded to the historical database.

5.7 Debug logging

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

```
MessageLevel: 'debug'
Gate.G_JDBC.Debug: TRUE
Gate.Java.Debug: TRUE
Gate.Mapper.Debug: TRUE
Gate.NGtkDebug: TRUE
Gate.RdrWtr.Debug: TRUE
Gate.RdrWtr.LogOSSql: TRUE
```

As well as logging SQL stubs for further analysis of exactly what data is being forwarded:

```
Gate.RdrWtr.LogOSSql: TRUE
```

5.8 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 -dumpprops

…

Gate.Java.ClassPath:
'/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/ngjava.jar:/opt/nrv81/IBM/tivoli/ne
tcool/omnibus/gates/java/ngtktk.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/
nco_g_jdbc.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/db2jcc_license_cu.jar
:/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/db2jcc.jar:/opt/nrv81/IBM/tivoli/ne
tcool/omnibus/gates/java/db2jcc4.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/
hsqldb.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/icu4j-
51_2.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/icu4j-charset-
51_2.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/jms.jar:/opt/nrv81/IBM/tivol
i/netcool/omnibus/java/jars/log4j-
1.2.8.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/ControlTower.jar:/opt/nrv81
/IBM/tivoli/netcool/omnibus/java/jars/baroctool.jar:/opt/nrv81/IBM/tivoli/netcool/omn
ibus/java/jars/confpack.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/icw.jar:/
opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/VersionFinder.jar:/opt/nrv81/IBM/tivol
i/netcool/omnibus/java/jars/jconn3.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jar
s/niduc.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/utility.jar:/opt/nrv81/IB
M/tivoli/netcool/omnibus/java/jars/OSReport.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus
/java/jars/repository.jar:/opt/nrv81/IBM/tivoli/netcool/omnibus/java/jars/org.eclipse
.swt.gtk.solaris.sparc_3.7.1.v3738a.jar'
```

Which uses $NCHOME/omnibus/gates/java and $NCHOME/omnibus/java/jars.

6 Example Installations

6.1 *Linux Red Hat : REPORTING*

Platform	: Linux Red Hat 7
Database	: Oracle 11g
Environment	: Netcool/OMNIbus v8.1

6.1.1 Configuration

The REPORTING mode of the JDBC gateway allows key data to be stored in a historical database. New columns can be added to the object server and historical tables to allow event states to be stored for reporting.

For example;

- FirstAcknowledged
- MaximumSeverity
- EscalationTime

Because each event is held within a single row, it is important that events are captured before they are deleted from the object server. It is recommended that a filter is used, alongside an ‘AFTER IDUC DO’ statement, so as to allow the delete triggers the capacity to delete only events that have been processed.

The other important benefit of using the REPORTING mode is that only the last state of an event needs to be stored in the object server, before it is deleted. This means that any issues with event processing or the JDBC gateway will have less impact on the final reports.

The recommended filter is to use a custom column as flag and set this columns value after the event is forwarded to the historical database;

e.g.

```
FILTER WITH 'HistoricalReporting>=1'
AFTER IDUC DO 'HistoricalReporting=2'
```

It is best to forward all journals and purge them as required form the historical database, so as to reduce the risk of journal loss, provided that journals are important in the reporting process. Otherwise only forward alerts.status and add custom columns to store key event life parameters, such as which user deleted the event, when the event was acknowledged, etc.

In this example the delete triggers would check that the HistoricalReporting column was either ‘0’ or ‘2’.

6.1.2 Properties file

```
Name: 'G_JDBC'
# Reporting mode properties
Gate.Jdbc.Mode: 'REPORTING'
# Table properties
Gate.Jdbc.StatusTableName: 'REPORTER_STATUS'
Gate.Jdbc.JournalTableName: 'REPORTER_JOURNAL'
Gate.Jdbc.DetailsTableName: 'REPORTER_DETAILS'
# JDBC Connection properties
# Setting CLASSPATH as multiple Oracle jars are installed - otherwise it's not required
Gate.Java.ClassPath:
'/opt/nrv731/tivoli/netcool/omnibus/gates/G_JDBC/java/classes12.jar:/opt/nrv731/tivoli/netcool/omnibus/gates/G_JDBC/java/ojdbc14.jar:/opt/nrv731/tivoli/netcool/omnibus/gates/java/ngtktk.jar:/opt/nrv731/tivoli/netcool/omnibus/gates/java/ngjava.jar:/opt/nrv731/tivoli/netcool/omnibus/gates/java/nco_g_jdbc.jar'
Gate.Jdbc.Url: 'jdbc:oracle:thin:@server.uk.ibm.com:1521:ORACLE11G'
Gate.Jdbc.Driver: 'oracle.jdbc.driver.OracleDriver'
Gate.Jdbc.Username: 'REPORTS'
Gate.Jdbc.Password: 'netcool'
Gate.Jdbc.ReconnectTimeout: 30
Gate.Jdbc.InitializationString: ''
# ObjectServer Connection properties
Gate.RdrWtr.Server: 'JDBC_COMS'
Gate.RdrWtr.Username: 'jdbcgw'
Gate.RdrWtr.Password: 'netcool'
Gate.UsePamAuth: TRUE
# Files
Gate.StartupCmdFile: '$NCHOME/omnibus/gates/G_JDBC/reporting.jdbc.startup.cmd'
Gate.MapFile: '$NCHOME/omnibus/gates/G_JDBC/reporting.jdbc.map'
Gate.RdrWtr.TblReplicateDefFile: '$NCHOME/omnibus/gates/G_JDBC/reporting.jdbc.rdrwtr.tblrep.def'
# Resynchronisation
Gate.Jdbc.ResyncMode: 'UNI'
Gate.Jdbc.ResyncFilter: 'LastOccurrence > (getdate - 36000)'
# Logging
MessageLog: '$NCHOME/omnibus/log/G_JDBC.log'
MaxLogFileSize: 102400
MessageLevel: 'warn'
# Failover/Failback
Gate.RdrWtr.Description: 'JDBC_ORACLE_gateway'
Gate.RdrWtr.FailbackEnabled: FALSE
Gate.RdrWtr.FailbackTimeout: 30
Gate.Transfer.FailoverSyncRate: 60
Gate.G_JDBC.FailbackEnabled: FALSE
Gate.G_JDBC.FailbackTimeout: 30
# Other settings
Gate.Reader.IgnoreStatusFilter : TRUE
#EOF
```

6.1.3 Map file

```
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',
    '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',
    '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
);
```

```
CREATE MAPPING JournalMap
(
    'SERIAL'            = '@Serial',
    'USERID'            = '@UID',
    'CHRONO'            = '@Chrono' CONVERT TO DATE,
    'TEXT1'             = '@Text1',
    'TEXT2'             = '@Text2',
    'TEXT3'             = '@Text3',
    'TEXT4'             = '@Text4',
    'TEXT5'             = '@Text5',
    'TEXT6'             = '@Text6',
    'TEXT7'             = '@Text7',
    'TEXT8'             = '@Text8',
    'TEXT9'             = '@Text9',
    'TEXT10'            = '@Text10',
    'TEXT11'            = '@Text11',
    'TEXT12'            = '@Text12',
    'TEXT13'            = '@Text13',
    'TEXT14'            = '@Text14',
    'TEXT15'            = '@Text15',
    'TEXT16'            = '@Text16',
# NB do not concatenate additional values for ServerName and ServerSerial !
    'SERVERNAME'        = STATUS.SERVER_NAME,
    'SERVERSERIAL'      = STATUS.SERVER_SERIAL
);

CREATE MAPPING DetailsMap
(
    'IDENTIFIER'        = '@Identifier',
    'ATTRVAL'           = '@AttrVal',
    'SEQUENCE'          = '@Sequence',
    'NAME'              = '@Name',
    'DETAIL'            = '@Detail',
# NB do not concatenate additional values for ServerName and ServerSerial !
    'SERVERNAME'        = STATUS.SERVER_NAME,
    'SERVERSERIAL'      = STATUS.SERVER_SERIAL
);
```

```
# The following maps can be used when the gateway is run in REPORTER mode
# and the example TRANSFER commands in the default nco_g_oracle.startup.cmd
# are required - and uncommented.

CREATE MAPPING NamesMap
(
    'NAME'                  = '@Name',
    'OWNERUID'              = '@UID',
    'OWNERGID'              = '@GID',
    'PASSWORD'              = '@Passwd',
    'TYPE'                  = '@Type'
);

CREATE MAPPING GroupsMap
(
    'NAME'                 = '@Name',
    'OWNERGID'             = '@GID'
);

CREATE MAPPING MembersMap
(
#    'OWNERKEY'             = TO_STRING('@UID') + TO_STRING('@GID'),
    'OWNERUID'             = '@UID',
    'OWNERGID'             = '@GID'
);

CREATE MAPPING ConversionsMap
(
    'CONVERSION_KEY'     = '@KeyField',
    'COLUMN_NAME'        = '@Colname',
    'VALUE'              = '@Value',
    'CONVERSION'         = '@Conversion'
);

CREATE MAPPING ObjectClassesMap
(
    'CLASS'              = '@Tag',
    'NAME'               = '@Name',
    'ICON'               = '@Icon',
    'MENU'               = '@Menu'
);
```

6.1.4 Table replication file

```
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
```

6.1.5 Start-up file

```
# TRANSFER static tables on JDBC Gateway start-up
TRANSFER FROM 'alerts.conversions' TO 'REPORTER_CONVERSIONS' DELETE USING TRANSFER_MAP ConversionsMap;
TRANSFER FROM 'master.groups' TO 'REPORTER_GROUPS' DELETE USING TRANSFER_MAP GroupsMap;
TRANSFER FROM 'master.members' TO 'REPORTER_MEMBERS' DELETE USING TRANSFER_MAP MembersMap;
TRANSFER FROM 'master.names' TO 'REPORTER_NAMES' DELETE USING TRANSFER_MAP NamesMap;
#EOF
```

6.2 Solaris : AUDIT

Platform	: Solaris 10
Database	: Sybase 15
Environment	: Netcool/OMNIbus v7.3.1

6.2.1 Configuration

With the AUDIT mode a new row is created in the historical database whenever Inserts, Updates and Deletes happen in the object server. In general all events need to be forwarded in this mode, so filtering and event reduction is not performed at the gateway. Instead the historical database prunes the data periodically, so as to reduce the data to a manageable amount, or else the data is archived.

6.2.2 Properties file

```
# Java
# Setting CLASSPATH as multiple versions of Sybase Jars are installed
Gate.Java.ClassPath: '/opt/nrv731/omnibus/tivoli/netcool/omnibus/gates/G_JSYB/java/jtds.jar:/opt
/nrv731/omnibus/tivoli/netcool/omnibus/gates/java/ngtktk.jar:/opt
/nrv731/omnibus/tivoli/netcool/omnibus/gates/java/ngjava.jar:/opt
/nrv731/omnibus/tivoli/netcool/omnibus/gates/java/nco_g_jdbc.jar'
# Limiting Java memory usage
Gate.Java.Arguments: '-Xmx1024m'
# Object Server & gateway
Name: 'G_JSYB'
Gate.RdrWtr.Server: 'JDBC_COMS'
Gate.RdrWtr.Username: 'jdbcgw'
Gate.RdrWtr.Password: 'netcool'
Gate.StartupCmdFile: '/opt /nrv731/omnibus/tivoli/netcool/omnibus/gates/G_JSYB/audit.jdbc.startup.cmd'
Gate.RdrWtr.TblReplicateDefFile: '/opt
/nrv731/omnibus/tivoli/netcool/omnibus/gates/G_JSYB/audit.jdbc.rdrwtr.tblrep.def'
Gate.MapFile: '/opt/ nrv731/omnibus/tivoli/netcool/omnibus/gates/G_JSYB/audit.jdbc.map'
MessageLog: '/opt/ nrv731/omnibus/tivoli/netcool/omnibus/log/G_JSYB15.log'
Ipc.SSLCertificate: '/opt/ nrv731/omnibus/tivoli/netcool/omnibus/gates/G_JSYB/audit.JDBC.crt'
Ipc.SingleThreaded: FALSE
Gate.RdrWtr.UseBulkInsCmd: FALSE
Gate.RdrWtr.IducFlushRate: 11
Gate.Mapper.ForwardHistoricJournals: TRUE
Gate.Mapper.ForwardHistoricDetails: FALSE
Gate.RdrWtr.IgnoreStatusFilter: TRUE
# JDBC
Gate.Jdbc.Url: 'jdbc:jtds:sybase://server.uk.ibm.com:5000;DatabaseName=audit_gw'
Gate.Jdbc.Driver: 'net.sourceforge.jtds.jdbc.Driver'
Gate.Jdbc.Username: 'jdbc_audit'
Gate.Jdbc.Password: 'jdbc_audit'
Gate.Jdbc.Connections: 7
Gate.Jdbc.MaxBatchSize: 100
Gate.Jdbc.Mode: 'AUDIT'
Gate.Jdbc.ActionCodeField: 'ActionCode'
Gate.Jdbc.ActionTimeField: 'ActionTime'
Gate.Jdbc.ResyncMode: 'BI'
Gate.Jdbc.DeletedAtField: 'DeletedAt'
Gate.Jdbc.ServerNameField: 'ServerName'
Gate.Jdbc.ServerSerialField: 'ServerSerial'
Gate.Jdbc.StatusTableName: 'jdbc_audit_status'
Gate.Jdbc.JournalTableName: 'jdbc_audit_journal'
Gate.Jdbc.DetailsTableName: 'jdbc_audit_details'
# Failover/Failback
Gate.RdrWtr.Description: 'Sybase JDBC gateway'
Gate.RdrWtr.FailbackEnabled: FALSE
Gate.RdrWtr.FailbackTimeout: 30
Gate.Transfer.FailoverSyncRate: 60
Gate.G_JDBC.FailbackEnabled: FALSE
Gate.G_JDBC.FailbackTimeout: 30
# Logging
MessageLevel: 'warn'
# EOF
```

6.2.3 Map file

```
CREATE MAPPING StatusMap
(
    'ActionCode'        = ACTION_CODE,
    'ActionTime'        = ACTION_TIME CONVERT TO DATE,
    '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',
    'StateChange'       = '@StateChange'        CONVERT TO DATE,
    'FirstOccurrence'   = '@FirstOccurrence'    ON INSERT ONLY CONVERT TO DATE,
    'LastOccurrence'    = '@LastOccurrence'     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',
    'Flash'             = '@Flash'              ON INSERT ONLY,
    'EventId'           = '@EventId'            ON INSERT ONLY,
    'ExpireTime'        = '@ExpireTime'         ON INSERT ONLY,

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

```
CREATE MAPPING JournalMap
(
    'Serial'            = '@Serial',
    'UID'               = '@UID',
    'Chrono'            = '@Chrono' CONVERT TO DATE,
    'Text1'             = '@Text1',
    'Text2'             = '@Text2',
    'Text3'             = '@Text3',
    'Text4'             = '@Text4',
    'Text5'             = '@Text5',
    'Text6'             = '@Text6',
    'Text7'             = '@Text7',
    'Text8'             = '@Text8',
    'Text9'             = '@Text9',
    'Text10'            = '@Text10',
    'Text11'            = '@Text11',
    'Text12'            = '@Text12',
    'Text13'            = '@Text13',
    'Text14'            = '@Text14',
    'Text15'            = '@Text15',
    'Text16'            = '@Text16',

# NB do not concatenate additional values for ServerName and ServerSerial !
    'ServerName'        = STATUS.SERVER_NAME,
    'ServerSerial'      = STATUS.SERVER_SERIAL
);

CREATE MAPPING DetailsMap
(
    'Identifier'        = '@Identifier',
    'AttrVal'           = '@AttrVal',
    'Sequence'          = '@Sequence',
    'Name'              = '@Name',
    'Detail'            = '@Detail',

# NB do not concatenate additional values for ServerName and ServerSerial !
    'ServerName'        = STATUS.SERVER_NAME,
    'ServerSerial'      = STATUS.SERVER_SERIAL
);
```

6.2.4 Table replication file

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

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

6.3 Same Second Updates : Linux/Oracle AUDIT mode

The AUDIT uses the ActionTime which is a second timestamp. Because of this any Updates in the same second will return unique constraint errors. Sometimes it s necessary to capture every event update, even when they are in the same second. The workaround for this is to use a counter field in the database tables to allow same second events. The counter automatically increases whenever a new row is inserted, thereby ensuring every inserted event is unique. The downside with this is that synhronisation events will also be inserted as unique events. This might be useful under strict auditing environments, but could be managed through the gateways Gate.Jdbc.ResyncMode property. The rest of the gateways configuration is the same as normal.

6.3.1 Example Oracle Schema

The following SQLPLUS commands are for a test system, you should consult with your Oracle DBA to determine the best tablespace settings for a production system.

```
sqlplus system@ORACLE


CREATE TABLESPACE AUDITSEQ
DATAFILE 'AUDITSEQ_DATA' SIZE 256M
AUTOEXTEND ON NEXT 128M
DEFAULT STORAGE (
                INITIAL 256M
                NEXT   128M
                MINEXTENTS 1
                MAXEXTENTS 999
                PCTINCREASE 0)
                LOGGING
                ONLINE;
COMMIT;

CREATE TEMPORARY TABLESPACE AUDITSEQ_TEMP
TEMPFILE 'AUDITSEQ_TEMP_DATA' SIZE 256M
AUTOEXTEND ON NEXT 128M MAXSIZE UNLIMITED
UNIFORM SIZE 128M;

COMMIT;

-- Create user.

CREATE USER AUDITSEQ IDENTIFIED BY AUDITSEQ
DEFAULT TABLESPACE AUDITSEQ
TEMPORARY TABLESPACE AUDITSEQ_TEMP
PROFILE DEFAULT ACCOUNT UNLOCK;
GRANT CONNECT TO AUDITSEQ;
GRANT RESOURCE TO AUDITSEQ;
GRANT SELECT_CATALOG_ROLE TO AUDITSEQ;
GRANT SELECT ANY TABLE TO AUDITSEQ;
GRANT UNLIMITED TABLESPACE TO AUDITSEQ;
```

```
sqlplus AUDITSEQ@ORACLE/AUDITSEQ

CREATE TABLE STATUS
(
        ActionTime              DATE NOT NULL,
        ActionCode              CHAR(1) NOT NULL,
        StatusSequence          NUMBER(24) NOT NULL,
        Identifier              VARCHAR2(255) NULL,
        Serial                  NUMBER(16) NULL,
        Node                    VARCHAR2(64) NULL,
        NodeAlias               VARCHAR2(64) NULL,
        Manager                 VARCHAR2(64) NULL,
        Agent                   VARCHAR2(64) NULL,
        AlertGroup              VARCHAR2(255) NULL,
        AlertKey                VARCHAR2(255) NULL,
        Severity                NUMBER(4) NULL,
        Summary                 VARCHAR2(255) NULL,
        StateChange             DATE NULL,
        FirstOccurrence         DATE NULL,
        LastOccurrence          DATE NULL,
        InternalLast            DATE NULL,
        Poll                    NUMBER(16) NULL,
        Type                    NUMBER(16) NULL,
        Tally                   NUMBER(16) NULL,
        Class                   NUMBER(16) NULL,
        Grade                   NUMBER(16) NULL,
        Location                VARCHAR2(64) NULL,
        OwnerUID                NUMBER(16) NULL,
        OwnerGID                NUMBER(16) NULL,
        Acknowledged            NUMBER(16) NULL,
        Flash                   NUMBER(16) NULL,
        ExpireTime              NUMBER(16) NULL,
        SuppressEscl            NUMBER(16) NULL,
        Customer                VARCHAR2(64) NULL,
        Service                 VARCHAR2(64) NULL,
        ServerName              VARCHAR2(64) NOT NULL,
        ServerSerial            NUMBER(16) NOT NULL
);

CREATE UNIQUE INDEX  STATUS_idx ON
STATUS(ActionTime,ActionCode,ServerSerial,ServerName);

commit;

ALTER TABLE STATUS
ADD PRIMARY KEY (
ActionTime,
ActionCode,
ServerSerial,
ServerName
)
;

commit;
```

```
CREATE TABLE JOURNAL (
JournalSequence 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
)
;

commit;

-- Create the Indexes for JOURNAL

CREATE INDEX JOURNAL_INDEX
ON JOURNAL (
ServerSerial,
ServerName
)
;

commit;

-- Create the Constraints for JOURNAL

ALTER TABLE JOURNAL
ADD PRIMARY KEY (
JournalSequence,
ServerSerial,
ServerName,
UserID,
Chrono
)
;

commit;
```

```
create sequence status_sequence
start with 1
increment by 1
maxvalue 1000000
cycle
cache 1000;

commit;


CREATE OR REPLACE TRIGGER status_counter
BEFORE INSERT ON STATUS
FOR EACH ROW
BEGIN
SELECT status_sequence.NEXTVAL
INTO :new.StatusSequence
FROM dual;
END;
/
commit;

create sequence journal_sequence
start with 1
increment by 1
maxvalue 1000000
cycle
cache 1000;
commit;


CREATE OR REPLACE TRIGGER journal_counter
BEFORE INSERT ON JOURNAL
FOR EACH ROW
BEGIN
SELECT journal_sequence.NEXTVAL
INTO :new.JournalSequence
FROM dual;
END;
/

commit;
```

7 Event Data Flow

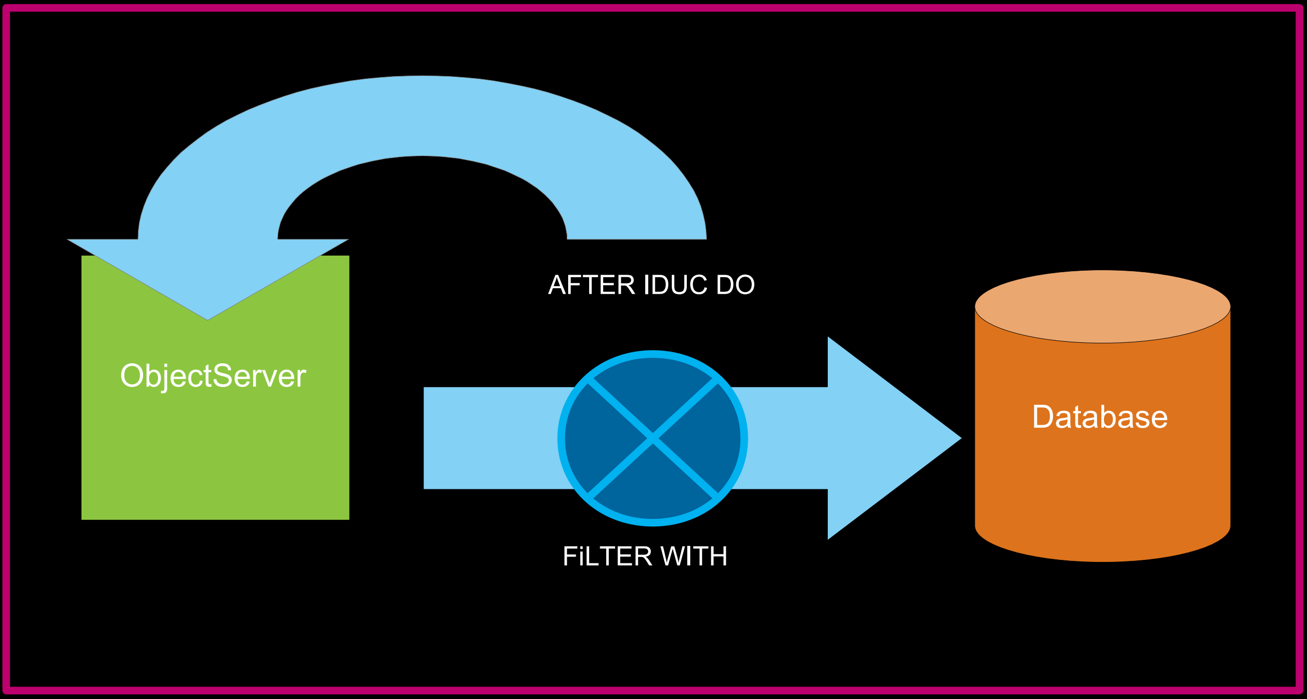
The design of the Target Database is dependent upon how the historical data is to be used.

7.1 *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.

7.2 *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.



7.2.1 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 JDBC gateway. The JDBC 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',
```

7.2.2 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 JDBC 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;
```

8 Static Table Updates

The JDBC gateway can be used to copy static tables to the database, using the TRANSFER command. This command can be run from the nco_sql command line or in the jdbc.startup.cmd to push static data to the historical database, for use with reporting tools.

The JDBC gateways' TRANSFER command has the same syntax as other gateways.

The syntax of the TRANSFER command is:

```
TRANSFER 'tablename' FROM readername TO writername
[ AS 'tableformat' ]
{ DELETE | DELETE condition | DO NOT DELETE }
[ USE TRANSFER_MAP ] [ USING FILTER filter_clause ];
```

The JDBC gateway includes the mapping definitions and the commented out TRANSFER commands in the jdbc.startup.cmd file:

e.g.

```
TRANSFER FROM 'alerts.conversions' TO 'REPORTER_CONVERSIONS' DELETE USING TRANSFER_MAP ConversionsMap;
TRANSFER FROM 'master.groups' TO 'REPORTER_GROUPS' DELETE USING TRANSFER_MAP GroupsMap;
TRANSFER FROM 'master.members' TO 'REPORTER_MEMBERS' DELETE USING TRANSFER_MAP MembersMap;
TRANSFER FROM 'master.names' TO 'REPORTER_NAMES' DELETE USING TRANSFER_MAP NamesMap;
```

However, not all databases support this command format.

8.1 DB2 Update Script Example

It may be required to run the TRANSFER commands periodically to ensure that the data is up to date. If this is required, create a shell-script, test that it works and then add the script to UNIX cron.

e.g.

```
vi $OMNIHOME/gates/G_DB2/tranfer_static_tables_to_db2.sh
#! /bin/sh
# Delete tables ready for TRANSFER
NCHOME=/opt/nrv73
OMNIHOME=$NCHOME/omnibus
LANG=C
LC_ALL=C
export NCHOME OMNIHOME LANG LC_ALL
#
DB2DATABASE=TCRMODEL
DB2USER=db2v95
DB2PASSWORD=netcool
export DB2DATABASE DB2USER DB2PASSWORD

echo "Purging static tables"
db2 CONNECT TO $DB2DATABASE USER $DB2USER USING $DB2PASSWORD
db2 -td@ << EOF
DELETE from REPORTER_CONVERSIONS @
DELETE from REPORTER_CLASSES @
DELETE from REPORTER_GROUPS @
DELETE from REPORTER_MEMBERS @
DELETE from REPORTER_NAMES @
COMMIT WORK @
exit
EOF
# TRANSFER tables from object server
GWHOST=localhost
GWUSER=root
GWPASSWORD=netcool
export GWHOST GWUSER GWPASSWORD

echo "Attempting to transfer static tables"
$OMNIHOME/bin/nco_g_icmd -hostname $GWHOST -username $GWUSER -password '' << EOF
TRANSFER FROM 'alerts.conversions' TO 'REPORTER_CONVERSIONS' USING TRANSFER_MAP ConversionsMap;
go
TRANSFER FROM 'alerts.objclass' TO 'REPORTER_CLASSES' USING TRANSFER_MAP ObjectClassesMap;
go
TRANSFER FROM 'master.groups' TO 'REPORTER_GROUPS' USING TRANSFER_MAP GroupsMap;
go
TRANSFER FROM 'master.members' TO 'REPORTER_MEMBERS' USING TRANSFER_MAP MembersMap;
go
TRANSFER FROM 'master.names' TO 'REPORTER_NAMES' USING TRANSFER_MAP NamesMap;
go
quit
EOF
echo " Transfer to DB2 script completed"
# Check tables
echo "Checking static tables"
db2 -td@ << EOF
CONNECT TO $DB2DATABASE USER $DB2USER USING $DB2PASSWORD @
select count(*)from REPORTER_CONVERSIONS @
select count(*)from REPORTER_CLASSES @
select count(*)from REPORTER_GROUPS @
select count(*)from REPORTER_MEMBERS @
select count(*)from REPORTER_NAMES @
exit
EOF
#EOF
:wq

chmod 755 $OMNIHOME/gates/G_DB2/tranfer_static_tables_to_db2.sh
```

Test that the script works as expected from the command line, ensuring the user running the script has the correct permissions to access the DB2 database. Afterwards, configure the script to run periodically using UNIX cron, as the same user.

8.2 Command line access

The nco_sql command is used to login to the gateway process and run commands, including the TRANSFER command.

In order to authenticate the UNIX user that is used to login to the JDBC gateway the gateway process must be run as root or else have the appropriate PAM configuration.

For PAM configuration on linux:-

Create a PAM login file for the nco_g_jdbc biinary;
cd /etc/pam.d
cp login nco_g_jdbc

Set the gateway to use PAM in the property file;
Gate.UsePamAuth: TRUE

The JDBC gateway process user and nco_sql login user must be the same;
e.g.

```
nco_sql -server G_JDBC -user netcool -password ********
1> get props;
2> go
```

Note : You should be using libngtktk version 3.2 or above

9 Common Issues

9.1 Status updates are not happening

There are no dedicated threads for alerts.status replication. The JDBC 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
```

Try increasing the buffer size and thread pool properties, whose default settings are:

```
Gate.Jdbc.Connections: 3
Gate.RdrWtr.BufferSize: 25
```

You can try increasing the settings, for example:

```
# Threads available for forwarding data to the database
Gate.Jdbc.Connections: 12
# Buffer and batch settings
Gate.RdrWtr.BufferSize: 500
```

If this does not help 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 old historical gateways like the ODBC Gateway used 7 threads [**`Gate.Jdbc.Connections:7`**].
- With these increased settings, the JDBC's gateway memory allocation will be increased.
- Use the Gate.Jdbc.ResyncFilter to manage the start-up synchronisation behaviour

9.2 Event Re-Awaken issues

In a multi tier 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 JDBC gateway.

Note : Reinserts in the aggregation layer will 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 any 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 is always longer than the CollectionExpireTime;
e.g.

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

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

This problem occurs as all events are described by the 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.

Note : July 2015 - APAR IV74891 covers the default behaviour for the multitier environment

9.3 Replicating Custom tables

For a table called custom.alerts, and a table in the database called TARGET_ALERTS_TABLE , you can use the TRANSFER feature to synchronise the tables on start-up.

It is possible to add a filter to the TRANSFER command to minimise the volume of reinserts seen when the JDBC gateway is started, depending upon the available columns in the custom table.

Here is a generic example of how to perform a single custom table replication:

jdbc.startup.cmd:

```
TRANSFER FROM 'custom.alerts' TO 'TARGET_ALERTS_TABLE' USING TRANSFER_MAP
CustomAlertsMap;
```

jdbc.map:

```
CREATE MAPPING CustomAlertsMap
(
'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,
'Summary' = '@Summary',
'Location' = '@Location' ON INSERT ONLY,
'Class' = '@Class' ON INSERT ONLY,
'Poll' = '@Poll' ON INSERT ONLY,
'Type' = '@Type' ON INSERT ONLY,
'Tally' = '@Tally',
'Severity' = '@Severity',
'OwnerUID' = '@OwnerUID',
'OwnerGID' = '@OwnerGID',
'Acknowledged' = '@Acknowledged',
'LastModified' = '@StateChange' CONVERT TO DATE,
'FirstOccurrence' = '@FirstOccurrence' ON INSERT ONLY CONVERT TO DATE,
'LastOccurrence' = '@LastOccurrence' CONVERT TO DATE,
'OriginalSeverity' = '@Severity' ON INSERT ONLY,
'ServerName' = '@ServerName' ON INSERT ONLY,
'ServerSerial' = '@ServerSerial' ON INSERT ONLY
);
```

jdbc.def:

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

REPLICATE ALL FROM TABLE 'custom.alerts'
USING MAP 'CustomAlertsMap' INTO 'TARGET_ALERTS_TABLE'
SET UPDTOINS CHECK TO FORCED
;
```

9.4 Recommended SQLSTATE settings

Development do not recommend any SQLSTATE settings other than the ones provided with the JDBC 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 JDBC 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 JDBC 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'
```

9.5 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.

9.6 Encrypting passwords and other properties

The JDBC Gateway supports AES encryption for encrypting passwords and other secret properties.

For example:

Generate the AES key for the JDBC Gateway:

```
$OMNIHOME/bin/nco_keygen -o $NCHOME/etc/security/keys/jdbc_gw.key
```

Obtain the encrypted string using the AES key:

```
$OMNIHOME/bin/nco_aes_crypt -c AES -k $NCHOME/etc/security/keys/jdbc_gw.key netcool
@44:etSQg2r6xBRjOj0g8cYvQM+VM5oaYPEQBJQkAnBeH3k=@


ConfigCryptoAlg : 'AES'
ConfigKeyFile : '$NCHOME/etc/security/keys/jdbc_gw.key'
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 would look like this:

```
# AES Encryption properties
ConfigCryptoAlg : 'AES'
ConfigKeyFile : '$NCHOME/etc/security/keys/jdbc_gw.key'
# Database connection user/password
Gate.Jdbc.Username: 'reportdb'
Gate.Jdbc.Password: '@44:etSQg2r6xBRjOj0g8cYvQM+VM5oaYPEQBJQkAnBeH3k=@'
# ObjectServer Connection properties
Name: 'G_JDBC'
Gate.RdrWtr.Server: 'AGG_V'
Gate.RdrWtr.Username: 'jdbcgw'
Gate.RdrWtr.Password: '@44:etSQg2r6xBRjOj0g8cYvQM+VM5oaYPEQBJQkAnBeH3k=@'
```

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

9.7 Creating an Object Server Gateway User

The JDBC gateway user needs to be a member of the Gateway group. If command line access is required to check access to custom tables, then the ISQL group can be added as well. The UserID (e.g. 300) should be a free UserID within the administration or gateway user ranges.

```
create user 'jdbcgw' id 300 full name 'JDBC ORACLE Gateway' password 'netcool';
go
alter group 'Gateway' assign members  'jdbcgw';
go
alter group 'ISQL' assign members  'jdbcgw';
go
```

9.8 Setting the Gateways Java path

The Gateways Java is picked up from the environmentfile.

You can find the file using UNIX find.
e.g.
find $NCHOME -name nco_g_jdbc.env

You can then edit the file and add the NCO_GATEWAY_JRE and echo messages to check the new Java path is used, as shown below:

```
# Top of file
NCO_GATEWAY_JRE=/opt/ibm-java-x86_64-80
export NCO_GATEWAY_JRE
echo "*** NCO_GATEWAY_JRE = $NCO_GATEWAY_JRE"

…
echo "*** NCO_GATEWAY_JRE = $NCO_GATEWAY_JRE"
echo "*** JRE_DIR = $JRE_DIR"
#EOF
```

9.9 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
Ipc.Timeout: 600
Gate.Jdbc.ResyncMode: 'UNI'
# Debugging settings
Gate.Jdbc.MaxBatchSize: 1
Gate.Jdbc.Connections: 1

# Debugging
MessageLevel: 'debug'
MaxLogFileSize: 10240
Gate.G_JDBC.Debug: TRUE
Gate.Java.Debug: TRUE
Gate.Mapper.Debug: TRUE
Gate.NGtkDebug: TRUE
Gate.RdrWtr.Debug: TRUE
Gate.RdrWtr.LogOSSql: TRUE
#EOF
```

Add the DB2JccConfiguration.properties files directory path to the CLASSPATH.

```
setenv CLASSPATH
/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/java/DB2/db2jcc.jar:/opt/nrv81/IBM/tivoli
/netcool/omnibus/gates/java
```

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.MaxBatchSize: 250
Gate.Jdbc.Connections: 12
# Debugging
MessageLevel: 'info'
MaxLogFileSize: 10240
Gate.G_JDBC.Debug: FALSE
Gate.Java.Debug: FALSE
Gate.Mapper.Debug: FALSE
Gate.NGtkDebug: 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
```

9.10 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;
```

9.11 Database Down Data loss

The JDBC 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 JDBC Gateway is still running and is not shutdown befre the Database returns fully, and the JDBC 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: 11
```

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 JDBC 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.