Tivoli Netcool Supports Guide to Historical Reporting Gateways

by

Jim Hutchinson

Document release: 2.0

Table of Contents

1 Introduction

1.1 Overview

The Netcool/OMNIbus product provides three main historical reporting gateways:

- Oracle Gateway [Oracle libraries]
- ODBC Gateway [DataDirect ODBC drivers]
- JDBC Gateway [Database JDBC drivers]

The Oracle and ODBC gateway's are superseded by the JDBC gateway.

This document was written to discuss how the historical reporting gateways manage the event flow to the database and how best to manage the data based on reporting usage. Full event auditing is not discussed as this is considered to be unnecessary.

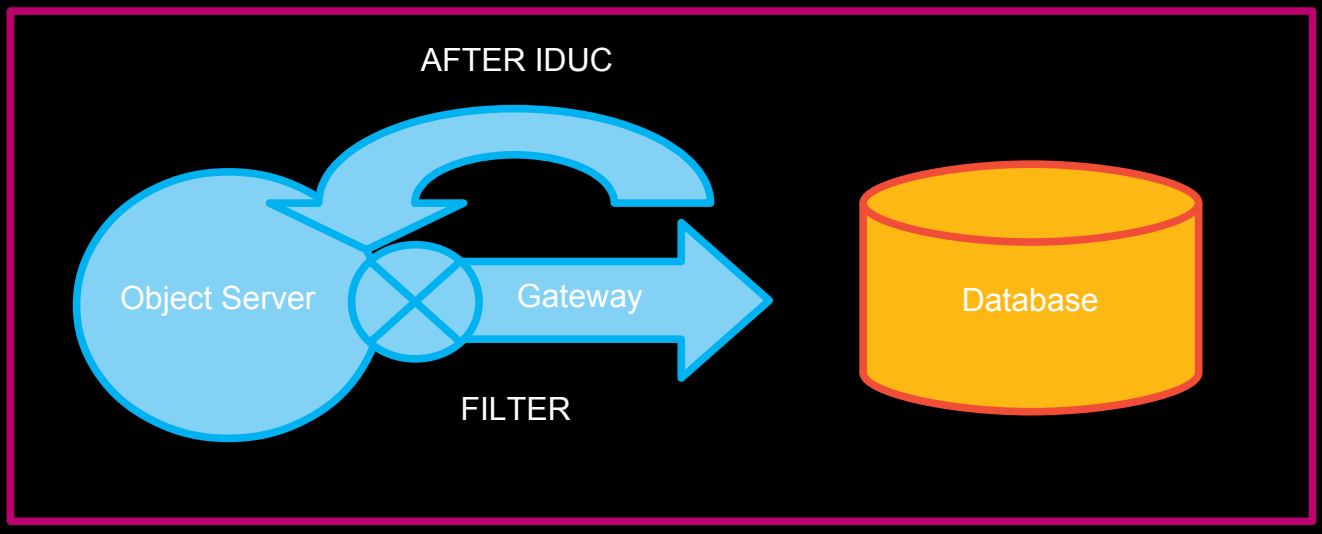
The historical reporting gateways forward three tables:

- alerts.status
- alerts.journal
- alerts.details

Typically alerts.details is not forwarded, and should not be, unless the data is actually required for reporting purposes.

The alerts.journal table can be forwarded, where event management auditing is required, otherwise it is best to add custom fields to the alerts.status table to record key event management states and actions.

All three gateways support a filter and 'after-iduc' option, which allows full control over the events forwarded to the database. When historical reporting gateway performance is an issue and the volume of events being stored is limited, it is best to control event flow using event filtering rather than relying on the standard event flow method alone.



2 Standard event flow control

All the historical reporting gateways have a built-in method to manage and control the flow of events without any major modification or customisation. These methods should be explored before attempting to implement the more complex customisations discussed within this document.

2.1 IDUC Flush Rate

The Historical reporting gateways use the IDUC Flush Rate property to control when events are forwarded to the database. The value of the IDUC Flush Rate is in seconds and when set to '0' or left unset, defaults to the object servers granularity [the default is 60 seconds].

Setting the IDUC Flush Rate to a high value reduces the update volumes the Historical reporting gateway has to manage, since an event may be updated many times within one IDUC period, resulting in only a single update for many fields being forwarded to the database, or in the case of a field being updated many times, only the last update in the IDUC period is forwarded.

2.2 Mapping file

The Historical reporting gateways use a mapping file to define which fields are forwarded to the database and how they are forwarded. The main table, alerts.status, is used to drive the event forwarding, by default. The two main properties of the mapping file are the field definition and the use of 'ON INSERT ONLY'.

For example:

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

Field values are generally only important when the event is inserted into the database, therefore using 'ON INSERT ONLY' can greatly reduce update volumes the Historical reporting gateway has to manage.

The general rule is to use ON INSERT ONLY' unless the field value must be kept updated, or is not set when the field is inserted into the database.

3 Event Filtering Flow Control

The historical reporting gateway will forward all events by default. However, if all events are not required, and the Historical reporting gateway has limited resources, event filtering flow control provides the solution.

In the following discussion the three historical reporting gateway flags are:
1 ORACLEGW
2 ODBCGW
3 JDBCGW

These are added to the source object servers, and set to a value based upon historical reporting requirements.

Flow control:

GWFLAG Value	Meaning
-1	Ready for forwarding using resynchronisation trigger
0	Do not forward to database
1	Forward to database
2	Forwarded to database

3.1 Historical Reporting Gateway Properties

3.1.1 Oracle Gateway

Example nco_g_oracle.props

```
Gate.ReaderFilter          : 'ORACLEGW>0'
Gate.ReaderAfterIDUC       : 'update alerts.status set ORACLEGW=2'
```

Fine tuning:

```
# alerts.status
Gate.ForwardStatusDel          : TRUE
Gate.ForwardStatusIns          : TRUE
Gate.ForwardStatusUpd          : TRUE
# alerts.journals
Gate.ForwardHistoricJournals   : TRUE
Gate.ForwardJournals           : TRUE
# alerts.details
Gate.ForwardDetails            : FALSE
Gate.ForwardHistoricDetails    : FALSE
```

3.1.2 ODBC Gateway

Example nco_g_odbc.props

```
Gate.ReaderFilter          : 'ODBCGW>0'
Gate.ReaderAfterIDUC       : 'update alerts.status set ODBCGW=2'
```

Fine tuning:

```
# alerts.status
Gate.ForwardStatusDel          : TRUE
Gate.ForwardStatusIns          : TRUE
Gate.ForwardStatusUpd          : TRUE
# alerts.journals
Gate.ForwardHistoricJournals   : TRUE
Gate.ForwardJournals           : TRUE
# alerts.details
Gate.ForwardDetails            : FALSE
Gate.ForwardHistoricDetails    : FALSE
```

3.1.3 JDBC Gateway

Example table replication definition jdbc.rdrwtr.tblrep.def

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'JDBCGW>0'
AFTER IDUC DO 'JDBCGW=2';

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

Fine tuning:

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'JDBCGW>0'
AFTER IDUC DO 'JDBCGW=2';

REPLICATE INSERT FROM TABLE 'alerts.journal'
USING MAP 'JournalMap';
```

You may need to set the IgnoreStatusFilter property:

```
# Ignore the status filter when required
Gate.Reader.IgnoreStatusFilter: TRUE
```

3.2 Minimum Updates

The historical reporting gateways can be configured to forward the minimum number of updates based upon reporting requirements, as well as on insert and on delete only. This minimum update method ensures that every event only accounts for two or three historical reporting gateway transactions.

For example:

```
Gate.ReaderFilter             : 'ODBCGW=1'
Gate.ReaderAfterIDUC          : 'update alerts.status set ODBCGW=2'

Fine tuning:
# alerts.status
Gate.ForwardStatusDel         : TRUE
Gate.ForwardStatusIns         : TRUE
Gate.ForwardStatusUpd         : FALSE
# alerts.journals
Gate.ForwardHistoricJournals  : FALSE
Gate.ForwardJournals          : FALSE
# alerts.details
Gate.ForwardDetails           : FALSE
Gate.ForwardHistoricDetails   : FALSE
```

In order to minimise the number of updates, the events are only inserted prior to their deletion, using a custom trigger.

```
-- Create temporal trigger for perform forwwarding and deletion
create or replace trigger nc_odbc_gateway_forward_on_delete
group nc_odbc_gateway
priority 1
comment 'Ensure events are forwarded upon deletion - extend period if required - ODBCGWDelete=1'
every 60 seconds
begin
        update alerts.status set ODBCGW = 1 , ODBCGWDelete = 2 where ODBCGWDelete = 1;
        delete from alerts.status where ODBCGW = 2 and ODBCGWDelete = 2;
end;
```

In this example ODBCGWDelete must be set to ‘1’ which could be performed in the default delete_clears trigger instead of performing the delete.

The configuration could be extended to allow new_row to perform the insert, so that the event is known about for reporting purposes [Setting ODBCGW=1]. Equally, a temporal trigger could be used to force the events current status, as required for reporting purposes.

Note that the alerts.status updates property would need to be set to true where event data needed to be updated rather than on insert/delete only:

```
Gate.ForwardStatusUpd         : TRUE
```

If journals need to be forwarded as well, then the journals insert trigger would need to be added and enabled, to ensure that ODBCGW was set to ‘1’ when a journal entry was added.

4 Controlled Synchronisation

4.1 Overview

When the historical reporting gateway connects to the source object server, as a specific user allocated to the gateway, all events that have the gateway flag set to ‘1’ for forwarding, has the value updated to ‘-1’, so that the event push trigger can be used to control the event rate. By default the period of the event push trigger is set to ‘1000’ events in ‘30’ seconds. Typically this value is lower than the historical reporting gateway is capable of, and would need to be tuned as required.

For example, setting the temporal trigger to ‘5000’ events in ‘10’ seconds may be more appropriate for the system. The number of events processed per period would be set according to the amount of memory available to the historical reporting gateway. The larger the value, the more memory the gateway process would use. The period of the trigger would be set to a value to ensure that the events are forwarded in a timely manner.

Consider a historical reporting gateway, whose peak event rate is 200 IDU’s per second, with the source object server having 20,000 standing events. Therefore in 10 seconds the historical reporting gateway could process 2,000 standing events, and in 100 seconds could process all the 20,000 standing events. However, if the object server had a high event turn-over, then the longer historical reporting gateway took to process events, the more standing events there would be.

i.e.

((standing events) / (gateway event rate)) = time to resynchronise

and

(gateway event rate) >> (event turn-over)

4.2 Object Server trigger solution

The Object Server trigger solution allows the object server to control how events are forwarded to the database when the historical reporting gateway resynchronises on start-up. Rather than forwarding all the events in one large block, the events can be blocked and forwarded periodically, to manage the historical reporting gateways loading and memory usage.

In the following examples the historical reporting gateway is the Oracle gateway, and uses the properties:

Gate.ReaderFilter : 'ORACLEGW>0'

Gate.ReaderAfterIDUC : 'update alerts.status set ORACLEGW=2'

To control which events are forwarded to the database.

4.2.1 Trigger : nc_oracle_gateway_resync

When the gateway user connects, the gateway processing flag is set [e.g. ORACLEGW] to a value [-1] which is used by the event push trigger [e.g. nc_push_oracle_resync_events].

For example:

```
if (%signal.username = 'oraclegw') then
              update alerts.status set ORACLEGW = -1 where ORACLEGW = 1;
end if;
```

4.2.2 Trigger : nc_push_oracle_resync_events

The event push trigger works through events where the gateway processing flag [e.g. ORACLEGW] is set to the resynchronisation value [-1]: The trigger sets the gateway processing flag to ‘1’ so that the historical reporting gateway will forward the event to the database. The period of the temporal trigger is every 30 seconds by default, and the amount of events processed each period is 1000. These values should be adjusted as required to meet with the systems requirements.

For example:

```
for each row push_event in alerts.status where push_event.ORACLEGW = -1
begin
    if (event_count < 1000) then
        update alerts.status set ORACLEGW = 1 where Identifier = push_event.Identifier;
        set event_count = event_count + 1;
    else
        cancel;
    end if;
end;
```

5 Journal considerations

The alerts.journal table holds the object server created journal entries. Usually these journal records are manually created, or created by a manual action, such as taking ownership of an event. Whilst these event handling records are useful they may not be required in the format given in the alerts.journal table, and may be difficult to produce reports on.

Therefore it is recommended that any event handling data is captured in the alerts.status stable, in a concise compact format, so that this data can be propagated through alerts.status and reported on.

5.1 The alerts.journal Table

The journals table contains the object server serial, the UID of the user who wrote the journal, along with the desktop timestamp of when the journal entry was recorded.
The historical reporting gateways mapping file has the following entry to propagate these details:

```
CREATE MAPPING JournalMap
(
    'SERIAL'            = '@Serial',
    'USERID'            = '@UID',
    'CHRONO'            = '@Chrono' CONVERT TO DATE,
    'TEXT1'             = '@Text1',
…
    'TEXT16'            = '@Text16',

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

Because the Text# fields can contain large blocks of text, the alerts.journal data is inefficient, and with the use of @UID instead of the users name, inaccurate for auditing. Some triggers and tools add the username to the Text# fields, but these cannot be reported on, therefore, the better method is to use alerts.status, to record key audit data that would normally be held in the alerts.journal table.

5.2 Example Event Handling Fields

With the loss of journals replication key points in the events life should be captured to allow data to be presented in custom reports. The new fields need to populated using customisations to existing tools and triggers, as well as custom tools and triggers.

The following fields could, for example be added to the Aggregation and display layer object servers to capture key data in an events life:

```
alter table alerts.status add FirstAcknowledged time;
alter table alerts.status add LastAcknowledged time;
alter table alerts.status add FirstResolved time;
alter table alerts.status add LastResolved time;

alter table alerts.status add FirstAssigned varchar(64);
alter table alerts.status add LastAssigned varchar(64);

alter table alerts.status add InitialSeverity int;
alter table alerts.status add FinalSeverity int;

alter table alerts.status add MaxSeverity int;

alter table alerts.status add DeletedBy varchar(64);
alter table alerts.status add ClearedBy varchar(64);
alter table alerts.status add ClosingComment varchar(255);
```

Once the fields are inserted, they can be populated as required. Notice that the username is captured rather than the UID [number], as the UID is linked to the username, and requires strict administration to be applied for successful event auditing.

As a minimum the new fields need to be added to the Aggregation bi-directional and historical reporting gateway as well, along with the new equivalent fields being added to the historical database.

If the system is complex, field names should be identified with a prefix, such as ‘Rep’, to allow administrators to identify reporting fields from other custom fields. This prefix should also be extended to custom triggers and tools.

6 The JDBC Gateway

The JDBC gateway is the latest historical reporting gateway product, and as such includes a number of features that can be used instead of the generic solutions discussed earlier within this document.

6.1 Gate.Jdbc.ResyncFilter

The Resync filter can be used to find open events in target. It can be set such that the events within the object server are selected for forwarding to the historical database. However, it should be noted that the 'AFTER IDUC DO' command is not run, therefore any filter should take this into account. For example setting the filter to 'JDBCGW>1' would allow all events in the object server to be forwarded, that have already been forwarded, or were set to be forwarded. Whilst setting the filter to 'JDBCGW>0' would cause events, to be forwarded, as well as, those events that had been forwarded, to be included in the resynchronisation process.

A better solution is to reassert the events through updating events where 'JDBCGW>0', setting 'JDBCGW=1', through a custom trigger or use the nc_push_jdbc_resync_events to perform the resynchronisation, and set the resync filter to an impossible value, or kept as the default, empty.

6.2 Gate.Jdbc.ResyncMode

The Resync mode takes the values of NONE, UNI, BI, AUTO, with the default being 'AUTO' mode. Setting the mode to 'NONE' with prevent any type of Synchronisation, whilst 'AUTO' is the same as 'UNI' unless the gateways cache is empty, in which case it behaves as if it was set to 'BI'. Please refer to the JDBC Gateways manual for complete descriptions of the modes.

Therefore if a custom resynchronisation is required the Resync mode should be set to 'NONE', otherwise it can be kept at its default value, 'AUTO'.

6.3 Gate.RdrWtr.IgnoreStatusFilter

The property allows the user to disable the status filter being applied to the alerts.details and alerts.journal data forwarding.
The default value is FALSE, which prevents the expected forwarding of other tables.
For example if alerts.journals needs to be forwarded, set to TRUE to allow alerts.journals to be forwarded as expected.

6.4 Historical Reporting Gateways Compatible properties

The following properties were added to allow backward compatibility with the earlier historical reporting gateway properties and behaviour:

Gate.Mapper.ForwardHistoricDetails: FALSE
Gate.Mapper.ForwardHistoricJournals: FALSE

The default behaviour is to prevent historic event forwarding, which are those rows that exist in the object server. Not forwarding historic data reduces the volume of reinserts and messages related to the row already existing in the target table.