# Netcool/OMNIbus ObjectServer Gateway
Version 7 Release 4

# *Reference Guide*

SC14-7531-00

# Netcool/OMNIbus ObjectServer Gateway
Version 7 Release 4

# *Reference Guide*

SC14-7531-00

**Note**
Before using this information and the product it supports, read the information in "Notices" on page 51.

**Edition notice**

This edition applies to version 7 release 4 of IBM Tivoli Netcool/OMNIbus ObjectServer Gateway (product number 5724-S44) and to all subsequent releases and modifications until otherwise indicated in new editions.

# Contents

# Chapter 1. ObjectServer Gateways

Use ObjectServer Gateways to replicate alerts and other data between ObjectServers. ObjectServer Gateways help you to improve the reliability and increase the scalability of your system. You can improve reliability by maintaining backup ObjectServers, and increase the scalability by establishing a multiered configuration.

ObjectServer Gateways can be unidirectional or bidirectional. ObjectServer Gateways consist of readers and writers. Readers extract alerts from a source ObjectServer. Writers send the alert data to a target ObjectServer.

The ObjectServer Gateway is installed with the Tivoli Netcool/OMNIbus installation package.

ObjectServer Gateways can replicate the data in any table between ObjectServers. Details of the tables to be replicated are stored in the table replication definition file and the map definition file.

You can improve the reliability of your system by setting up a pair of ObjectServers that are connected by a bidirectional gateway. All clients, except the bidirectional gateway, connect to the primary ObjectServer. The backup ObjectServer acts as a standby, and is kept up to date by the bidirectional gateway.

In a multitiered configuration, ObjectServer Gateways function as follows:

- Each collection layer ObjectServer has its own dedicated unidirectional ObjectServer Gateway that connects the ObjectServer to the aggregation layer.
- The aggregation layer includes one pair of ObjectServers that is connected by a bidirectional ObjectServer Gateway to keep them synchronized. The bidirectional ObjectServer Gateway runs on the backup host.
- Each display layer ObjectServer has its own dedicated unidirectional ObjectServer Gateway that connects the ObjectServer to the aggregation layer. Each display gateway reader connects to the virtual aggregation pair whereas each gateway writer connects, and is fixed, to its dedicated display ObjectServer. Therefore, although the readers can fail over and fail back between the primary and backup aggregation layer ObjectServers, the writer stays connected only to its dedicated display ObjectServer. (These gateway connections are the opposite of the gateway connections in the collection layer.)

For more information about configuring the multitiered architecture, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

## Bidirectional ObjectServer Gateways

The bidirectional ObjectServer Gateway enables alerts to flow in both directions between two ObjectServers. You can use bidirectional gateways to create a failover pair of ObjectServers. The executable file name of a bidirectional ObjectServer gateway is `nco_g_objserv_bi`.

Changes in one ObjectServer are replicated in the other ObjectServer. This replication ensures that both ObjectServers contain the same alerts and allows you to maintain a backup ObjectServer.

**Important:** Ensure that table data is changed in only one of the ObjectServers. If table data is changed in both ObjectServers, the gateway might not be able to reconcile the changes and the row data might be left in an indeterminate state. The configuration setup for a failover pair of ObjectServers ensures that only data in the primary ObjectServer is changed, except when the system fails over to the backup ObjectServer. For more information about failover, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

## Information flow

The following figure shows the structure of a bidirectional ObjectServer Gateway.

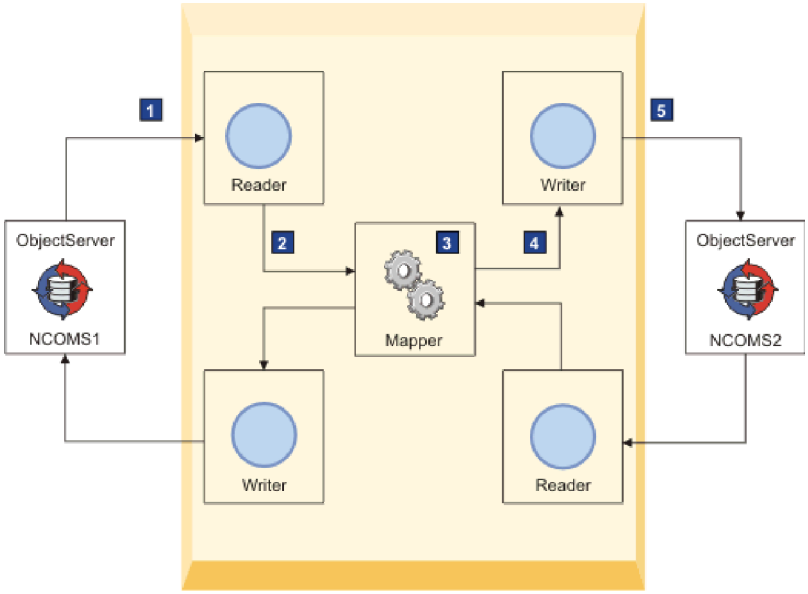


*Figure 1. Bidirectional ObjectServer Gateway*

The bidirectional gateway consists of a mapper and two reader/writer components (one for each ObjectServer). The flow of information between the components is as follows:

**1** : The reader reads data from one ObjectServer.

**2** : The data is passed to the gateway mapper.

**3** : The mapper transforms the data into a format that is appropriate for the second ObjectServer. To transform the data, the mapper uses the settings in the map definition file.

**4** : The mapper passes the data to the reader.

**5** : The writer writes the data to the second ObjectServer.

**Related tasks**:

"Configuring failback operations" on page 8

## Unidirectional ObjectServer Gateways

The unidirectional ObjectServer Gateway enables alerts to flow in one direction, from a source ObjectServer to a destination ObjectServer. The executable file name of a unidirectional ObjectServer gateway is `nco_g_objserv_uni`.

Changes made in the source ObjectServer are reflected in the destination ObjectServer, but changes in the destination ObjectServer are not reflected in the source ObjectServer.

### Information flow

The following figure shows the structure of a unidirectional ObjectServer Gateway.

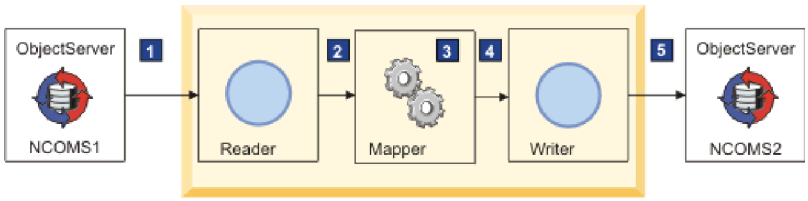


*Figure 2. Unidirectional ObjectServer Gateway*

The unidirectional gateway is comprised of mapper, reader, and writer components. The flow of information between the components is as follows:

**1** : The gateway reader reads data from the source ObjectServer, NCOMS1.

**2** : The data is passed to the gateway mapper.

**3** : The data is mapped from the tables and columns of the source ObjectServer to those in the target ObjectServer, based on the settings in the map definition file.

**4** : The mapper passes the data to the gateway writer.

**5** : The writer writes the data to the target ObjectServer, NCOMS2.

## Running ObjectServer Gateways

After you configured the gateway, you can issue the command to start it.

- To start a bidirectional gateway, enter the following command: `nco_g_objserv_bi -name`, where *name* is the name of the gateway.
- To start a unidirectional gateway, enter the following command: `nco_g_objserv_uni -name`, where *name* is the name of the gateway.

# Chapter 2. Configuring ObjectServer Gateway properties

To define the operational environment of the gateway, such as connection details and the location of the other configuration files, set the properties in the properties file. This is a text file that contains a set of properties and their corresponding values.

By default, the ObjectServer Gateway reads the properties file from the `$OMNIHOME/etc/NCO_GATE.props` file. You can use the `-name` command-line option to rename this file, or the `-propsfile` option to instruct the gateway to read a different properties file. Sample, writeable properties files are provided at `$NCHOME/omnibus/gates/objserv_uni/objserv_uni.props` for unidirectional gateways and `$NCHOME/omnibus/gates/objserv_bi/objserv_bi.props` for bidirectional gateways.

For information about the common properties and Interprocess Control (IPC), see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

Most configuration tasks are identical for unidirectional gateways and bidirectional gateways. However, if a configuration task requires different properties to be set, depending on whether the gateway is unidirectional or bidirectional, the tasks are described separately, for example setting up failback operations.

After you edit the properties file, restart the gateway so that the changes take effect.

The standard multitiered architecture includes sample configurations, including properties file, for gateways at the collection layer, aggregation layer, and display layer. These configurations are in `$NCHOME/omnibus/extensions/multitier/gateway`. For example, `$NCHOME/omnibus/extensions/multitier/gateway/AGG_GATE` contains configuration for an aggregation layer ObjectServer that can be deployed when you run a single primary ObjectServer to a backup ObjectServer.

## Controlling the size of the cache

The gateway uses a cache to store details of tables that require transferring from one ObjectServer to another. Change the size of the cache if the ObjectServer to which the gateway is linked handles large numbers of events.

The main function of the cache is to facilitate journal and details table insert operations. When a journal or detail is forwarded for insertion into a target ObjectServer, the gateway writer needs to know the corresponding status serial in the target ObjectServer. This information is found in the cache. The cache is also used for any other tables that are specified in the table replication definition table.

The cache increases performance by providing the gateway with an in-memory summarized view of the contents of the ObjectServers to which it is linked. As a result, the gateway does not have to query an ObjectServer to check for the existence of an event, or the Serial or Tally of an event. Instead, it can check the cache of the ObjectServer.

The cache is implemented by a hash table. Performance decreases if the number of rows in the ObjectServer status table is many times the number of entries in the hash table.

To control the size of the hash table cache, change the value of the **Gate.CacheHashTblSize** property. The default is 5023 elements, or rows. If a status table has many rows, for example, over 20,000, increase the number. For efficiency, specify a value that is a prime number.

### What to do next

After you edited the properties file, restart the gateway.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Setting the level of debug messages

You can troubleshoot problems with the gateway by consulting error messages. The gateway has configurable error handling, which is provided by the Netcool/OMNIbus Gateway Toolkit (NGTK) library. You can specify which messages are included in the debug files.

1. To specify that the NGTK library logs debug messages, set the **Gate.NGtkDebug** property to TRUE.
2. To specify which debug messages are included in the debug files, set the following properties to TRUE or FALSE, as required:
   - **Gate.Mapper.Debug**
   - **Gate.ObjectServerB.Debug**
   - **Gate.ObjectServerA.Debug**

### What to do next

After you edit the properties file, restart the gateway.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Checking whether the gateway runs under process control

The gateway can be put under Process Agent (PA) control, in which case, the PA specifies how the gateway runs. You can check whether a gateway is under PA control by checking the gateway properties.

To check whether a gateway is under PA control:

- To check whether the gateway is PA aware, check the value of the **Gate.PAAware** property. If the property is set to 0, the gateway is not PA aware.
- For the name of the PA agent that is running the gateway, check the value of the **Gate.PAAwareName** property

**Important:** Do not change these properties in the gateway properties file. These properties are maintained automatically by the PA server and provide information

only.
For more information about PA control, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Changing the buffer size

The buffer size controls the number of entries that the gateway stores in its buffer before flushing them to the ObjectServer. The optimum value for the buffer size depends on the average event size and the speed of the network. The default buffer size is often sufficient. To reduce latency, you can adjust the buffer size.

The gateway uses separate buffers for the source and destination ObjectServers

You can adjust the buffer size as follows:

1. To determine the most efficient setting for your system, keep a record of the time that it takes for resynchronization operations to complete. Then, change the resynchronization properties and resynchronize again, and compare the timing figures. Repeat as required.
2. To change the buffer size for the source and destination ObjectServer, change the values of the following properties accordingly:
   - **Gate.ObjectServerA.Buffersize**
   - **Gate.ObjectServerB.Buffersize**

### What to do next

After you edi the properties file, restart the gateway.

**Related tasks**:

"Configuring resynchronization" on page 12

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Changing the authentication mechanism

The gateway supports standard UNIX authentication or Pluggable Authentication Modules (PAM) authentication. Standard UNIX authentication is the default. PAM authentication is required to run the gateway in FIPS 140-2 mode.

For more information about authentication, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

To change the authentication mechanism to PAM, set the **Gate.UsePamAuth** to `TRUE`.

### What to do next

1. Configure Tivoli Netcool/OMNIbus to use PAM for external authentication. The service names are `nco_g_objserv_uni` and `nco_g_objserv_bi`. Both have the module type `account`. For more information about the configuration for PAM, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.
2. Restart the gateway.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Connecting to secure ObjectServers

When an ObjectServer is running in secure mode, the gateway must make its connection either as a known ObjectServer user or as the root user. The user must have permission to access the tables that are being replicated, and also some of the system tables. The default ObjectServer configuration includes a group that is called Gateway, which has the required permissions.

For more information about running the ObjectServer is secure mode, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Resynchronizing security data with secure ObjectServers

If you want to resynchronize security data when the ObjectServers are running in secure mode, run the gateway as the root user. If you do not, when you attempt to resynchronize, the gateway quits and no security data is transferred to the destination ObjectServer.

No security data is transferred because the gateway deletes the destination permissions and so cannot insert rows copied from the source table. If you run the gateway as the root user, this problem is overcome, because no permissions need to be set explicitly. For more information, about the ObjectServer, see *IBM Tivoli Netcool/OMNIbus Administration Guide*.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Configuring failback operations

Use the gateway properties to configure how a backup ObjectServer fails back to a primary ObjectServer after the primary ObjectServer is restored. If the failback function is enabled, when the connection to the primary ObjectServer is lost, the gateway connects to the backup ObjectServer. When the primary ObjectServer becomes active, the gateway reconnects to it.

**Important:** Although you can use ObjectServer Gateways to control failback, to minimize event loss, client failback behavior must be controlled by a failover pair of ObjectServers instead of the clients themselves. Event loss can occur if clients fail back to a primary ObjectServer before resynchronization is completed.

## Configuring failback operations for unidirectional ObjectServer Gateways

Two ObjectServers can be set up as a pair, with one acting as the primary and the other as the backup. You can specify how the backup ObjectServer fails back to the primary ObjectServer.

These instructions are applicable only to gateway readers or writers that connect to a virtual ObjectServer pair.

1. In the backup ObjectServer, set the **BackupObjectServer** property to TRUE and restart the ObjectServer, if required.
2. In the gateway properties file, set the **Gate.Reader.Failback** property and/or the **Gate.Writer.Failback** property to TRUE.
3. To specify the frequency with which the reader and writer parts of the gateway poll the failed primary ObjectServer, set the **Gate.Reader.FailbackTimeout** and/or **Gate.Writer.FailbackTimeout** properties.
4. To specify the Object Server pairs, set the **Gate.ObjectServerA.Server** property and the **Gate.ObjectServerB.Server** property to the virtual Object Server name.

### Results

When the primary ObjectServer fails, the reader and writer fail over to the backup ObjectServer without shutting down. When the reader or writer detects that they are now connected to a backup ObjectServer, they periodically poll for the return of the primary ObjectServer. When the primary ObjectServer is detected again, the reader or writer automatically fails back to the primary ObjectServer.

### Example

The following figure shows an example unidirectional gateway failback configuration.

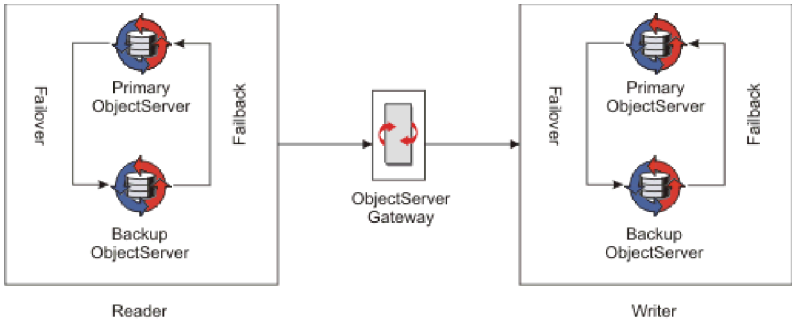


*Figure 3. Configuration of failback for unidirectional ObjectServer Gateways*

Typically, in a multitiered architecture, the collection layer has a single ObjectServer as the reader and a pair of ObjectServers as the writer. The aggregation layer has a pair of ObjectServers as the reader and the writer. The display layer has a single ObjectServer as the writer and a pair of ObjectServers as the reader.

### What to do next

After you edit the properties file, restart the gateway.

# Configuring failback operations for bidirectional ObjectServer Gateways

Provided that data is changed in only one ObjectServer in a virtual pair, you can set up a bidirectional gateway for failback operations. The operation is the same as for a unidirectional gateway, but you need to configure different properties.

**Restriction:** Failback does not occur when the bidirectional gateway connects to an ObjectServer that is not in a virtual pair. For example, if you are using a bidirectional gateway to maintain a backup ObjectServer, failback does not occur.

To configure failback operations:

1. In the backup ObjectServer, set the **BackupObjectServer** property to TRUE and restart the ObjectServer, if required.
2. Set the following properties in the gateway properties file.
   - If ObjectServer A has a backup ObjectServer, set the **Gate.ObjectServerA.Failback** property to TRUE.
   - If ObjectServer B has a backup ObjectServer, set the **ObjectServerGate.ObjectServerB.Failback** property to TRUE
3. To specify the frequency with which the gateway polls the failed ObjectServer, set the **Gate.ObjectServerA.FailbackTimeout** and **Gate.ObjectServerB.FailbackTimeout** properties.
4. To specify the Object Server pairs, set the **Gate.ObjectServerA.Server** property and the **Gate.ObjectServerB.Server** property to the virtual Object Server name.

## Results

When the primary ObjectServer fails, the gateway fails over to the backup ObjectServer without shutting down. When the gateway is connected to a backup ObjectServer, it periodically polls for the return of the primary ObjectServer. When the primary ObjectServer is detected again, the gateway automatically fails back to the primary ObjectServer.

## Example

The following figure shows a sample failback operation for a bidirectional gateway.

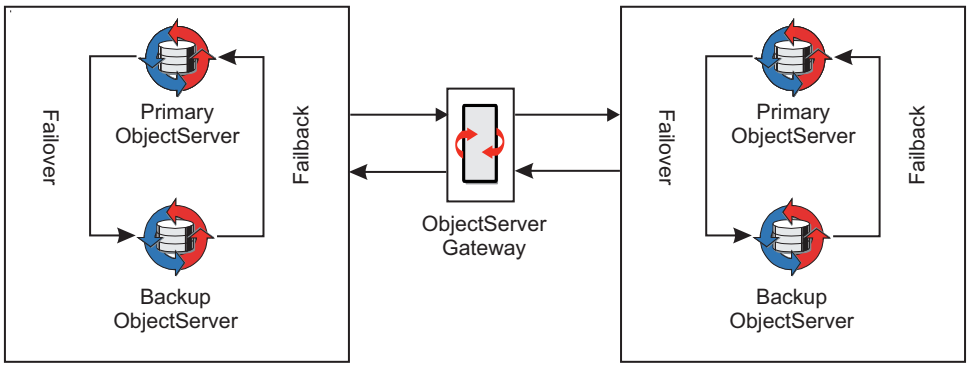


*Figure 4. Bidirectional ObjectServer Gateway*

## What to do next

After you edit the properties file, restart the gateway.

# Configuring store-and-forward operations

You can configure bidirectional and unidirectional gateways to store and forward the data of replicated tables if the destination ObjectServer goes offline.

## Configuring store-and-forward operations for unidirectional ObjectServer Gateways

Unidirectional gateways store and forward on any table when the destination ObjectServer goes offline. You can configure the gateway to store and forward either all or none of the tables that are replicated.

To configure the store-and-forward function:

1. Set the **Gate.Writer.SAF** property to TRUE.
2. To specify the file to which alerts are written while the destination ObjectServer is offline, set the **Gate.Writer.SAFFile** property.
3. Restart the gateway.

### What to do next

To deactivate the store and forward function, set the **Gate.Writer.SAF** property to FALSE and restart the gateway.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Configuring store-and-forward operations for bidirectional ObjectServer Gateways

The gateway supports store and forward on any table when the destination ObjectServer goes offline. You can configure the gateway to store and forward either all or none of the tables that are replicated.

1. To configure the store-and-forward function for ObjectServer A:
   a. Set the **Gate.ObjectServerA.SAF** property to TRUE.
   b. To specify the file to which alerts are written while the destination ObjectServer is offline, set the **Gate.ObjectServerA.SAFFile** property
2. To configure the store-and-forward function for ObjectServer B:
   a. Set the **Gate.ObjectServerB.SAF** property to TRUE.
   b. To specify the file to which alerts are written while the destination ObjectServer is offline, set the **Gate.ObjectServerB.SAFFile** property
3. After you reset the properties, restart the gateway.

### What to do next

To deactivate the store and forward function, set the **GateObjectServerA.SAF** property to FALSE, set the **Gate.ObjectServerB.SAF** property to FALSE, and restart the gateway.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

# Configuring resynchronization

Use the gateway properties to specify how the gateway resynchronizes with its target ObjectServers. You can configure resynchronization for both bidirectional and unidirectional ObjectServer Gateways.

**Related tasks**:

"Changing the buffer size" on page 7

## Configuring resynchronization for unidirectional ObjectServer Gateways

To specify how unidirectional gateways resynchronize with the ObjectServers, set the resynchronization properties that are in the properties file. Unidirectional ObjectServer Gateways have fewer resynchronization properties than bidirectional ObjectServer Gateways.

To configure resynchronization, set the properties that are described in the following table.

*Table 1. Resynchronization properties and command line options of unidirectional ObjectServer Gateways*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Resync.Enable** *boolean* | `-resyncenable` *boolean* | Use this property to specify that the gateway uses resynchronization.<br><br>The default is `TRUE`. |
| **Gate.Resync.Type** *string* | `-resynctype` *string* | Use this property to specify the how the gateway resynchronizes table data between ObjectServers when the gateway starts or restores a lost connection. The gateway resynchronizes the tables that are defined in the table replication definition file.<br><br>See the information that follows this table for the values to which you can set this property.<br><br>The default is `NORMAL`. |
| **Gate.Resync.LockType** *string* | `-resynclocktype` *string* | Use this property to specify the locking option on the source and destination ObjectServers while resynchronizing events.<br><br>You have the following options:<br>• `FULL`: The gateway locks both the source and target ObjectServers.<br>• `PARTIAL`: The gateway only locks the destination ObjectServer.<br>• `NONE`: The gateway locks neither the source nor the target ObjectServer.<br><br>The default is `FULL`. |

You can set the **Gate.Resync.Type** property to one of the following values:

- NORMAL: For each table, the gateway deletes all the data from the slave ObjectServer. Then, the gateway retransfers the full set of tables from the master to the slave. With this type of resynchronization, the master and slave are fully synchronized. However, table rows that are in the slave but not in the master are lost. Additionally, if table rows are in the master and the slave, the copy of the row that is on the master is retained on both the master and the slave. Any prior updates to the row on the slave are lost.
- UPDATE: For each table, the gateway builds a cache that contains all rows in the master and slave ObjectServers. Then, the gateway examines the contents of the cache for each table and compares the row data from the master with the row data from the slave. The data is resynchronized as follows:
  - Rows in the slave that are also in the master are updated with the data from the master, if the data in the master is different from the slave.
  - Rows that are in the master but not in the slave are copied to the slave.
  - Rows in the slave that are not in the master are retained.

  With this type of resynchronization, no events are lost, but the master and slave ObjectServers might not be fully synchronized.
- MINIMAL: This option behaves in the same way as UPDATE. In addition, events (that is, rows in the alerts.status table) that are in the slave but not in the master are marked for deletion. To mark these events for deletion, the gateway behaves as follows:
  1. For each row of the alerts.status table in the slave ObjectServer that is not in the master, the OldRow field is set to 1.
  2. The pass_deletes trigger runs on the slave ObjectServer and deletes all rows in which the OldRow field is set to 1.

  The benefit of a MINIMAL resynchronization is that the master and slave ObjectServers are fully synchronized but less data is sent during the resynchronization process. MINIMAL resynchronization is less data-intensive because all the rows are not deleted and then recopied, unlike a NORMAL resynchronization.

### What to do next

After you edit the properties file, restart the gateway.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Configuring resynchronization for bidirectional ObjectServer Gateways

To specify how bidirectional gateways resynchronize with the ObjectServers, set the resynchronization properties that are in the properties file. Bidirectional ObjectServer Gateways have more resynchronization properties than unidirectional ObjectServer Gateways.

To configure resynchronization, set the properties that are described in the following table.

*Table 2. Resynchronization properties and command line options of bidirectional ObjectServer Gateways*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Resync.Enable** *boolean* | -resyncenable *boolean* | Use this property to specify that the gateway uses resynchronization.<br><br>The default is TRUE. |
| **Gate.Resync.Master** *string* | -resyncmaster *string* | Use this property to specify which ObjectServer the gateway should always treat as the master during resynchronization.<br><br>Valid values are ObjectServerA and ObjectServerB.<br><br>The default is "".<br>**Note:** If you omit this property, the gateway always treats the ObjectServer that has been running the longest as the master. |
| **Gate.Resync.Preferred** *string* | -resyncpreferred *string* | Use this property to specify which ObjectServer the gateway should treat as the master during resynchronization if the **Gate.Resynch.Master** has been omitted and both ObjectServers have been running for the same length of time.<br><br>Valid values are ObjectServerA and ObjectServerB.<br><br>The default is "". |
| **Gate.Resync.Type** *string* | -resynctype *string* | Use this property to specify the how the gateway resynchronizes table data between ObjectServers when the gateway starts or restores a lost connection. The gateway resynchronizes the tables that are defined in the table replication definition file.<br><br>See the information that follows this table for the values to which you can set this property.<br><br>The default is NORMAL. |

*Table 2. Resynchronization properties and command line options of bidirectional ObjectServer Gateways (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Resync.LockType** *string* | -resynclocktype *string* | Use this property to specify the locking option on the source and destination ObjectServers while resynchronizing events.<br><br>You have the following options:<br>• FULL: The gateway locks both the source and target ObjectServers.<br>• PARTIAL: The gateway only locks the destination ObjectServer.<br>• NONE: The gateway locks neither the source nor the target ObjectServer.<br><br>The default is FULL. |

You can set the **Gate.Resync.Type** property to one of the following values:

- NORMAL: For each table, the gateway deletes all the data from the slave ObjectServer. Then, the gateway retransfers the full set of tables from the master to the slave. With this type of resynchronization, the master and slave are fully synchronized. However, table rows that are in the slave but not in the master are lost. Additionally, if table rows are in the master and the slave, the copy of the row that is on the master is retained on both the master and the slave. Any prior updates to the row on the slave are lost.
- UPDATE: For each table, the gateway builds a cache that contains all rows in the master and slave ObjectServers. Then, the gateway examines the contents of the cache for each table and compares the row data from the master with the row data from the slave. The data is resynchronized as follows:
  - Rows in the slave that are also in the master are updated with the data from the master, if the data in the master is different from the slave.
  - Rows that are in the master but not in the slave are copied to the slave.
  - Rows in the slave that are not in the master are retained.

  With this type of resynchronization, no events are lost, but the master and slave ObjectServers might not be fully synchronized.
- MINIMAL: This option behaves in the same way as UPDATE. In addition, events (that is, rows in the alerts.status table) that are in the slave but not in the master are marked for deletion. To mark these events for deletion, the gateway behaves as follows:
  1. For each row of the alerts.status table in the slave ObjectServer that is not in the master, the OldRow field is set to 1.
  2. The pass_deletes trigger runs on the slave ObjectServer and deletes all rows in which the OldRow field is set to 1.

  The benefit of a MINIMAL resynchronization is that the master and slave ObjectServers are fully synchronized but less data is sent during the resynchronization process. MINIMAL resynchronization is less data-intensive because all the rows are not deleted and then recopied, unlike a NORMAL resynchronization.

## What to do next

After you edit the properties file, restart the gateway.

**Related reference**:

"Bidirectional gateway properties" on page 25

"Unidirectional gateway properties" on page 19

## Generic ObjectServer Gateway properties

Certain properties are shared by unidirectional and bidirectional ObjectServer Gateways.

For information about the generic gateway properties and Interprocess Communication (IPC) properties, see the *IBM Netcool/OMNIbus Probe and Gateway Guide*. The following table describes the properties that are shared by unidirectional and bidirectional ObjectServer Gateways.

*Table 3. Generic ObjectServer Gateway properties and command-line options*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.CacheHashTblSize** *integer* | `-chashtblsize` *integer* | Use this property to specify the size (in elements) that the gateway allocates for the hash table cache.<br><br>The default is `5023`. |
| **Gate.MapFile** *string* | `-mapfile` *string* | Use this property to specify the location of the map definition file.<br><br>The default is `$OMNIHOME/gates/objserv_uni/objserv_uni.map` for unidirectional gateways and `$OMNIHOME/gates/objserv_bi/objserv_bi.map` for bidirectional gateways. |
| **Gate.StartupCmdFile** *string* | `-startupcmdfile` *string* | Use this property to specify the location of the startup command file.<br><br>The default is `$OMNIHOME/objserv_uni/objserv_uni.startup.cmd` for unidirectional gateways and `$OMNIHOME/objserv_bi/objserv_bi.startup.cmd` for bidirectional gateways. |
| **Gate.Transfer.FailoverSyncRate** *integer* | `-fsyncrate` *integer* | Use this property to specify the duration (in seconds) of the failover synchronization.<br><br>The default is `60`, which means that. every 60 seconds, the gateway transfers the contents of any table defined for failover synchronization from the source ObjectServer to the target ObjectServer. |

*Table 3. Generic ObjectServer Gateway properties and command-line options (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.NGtkDebug** *boolean* | -ngtkdebug *boolean* | Use this property to specify whether the NGTK library logs debug messages.<br><br>The default is TRUE.<br>**Tip:** Use the following properties to specify which debug messages are included in the debug log file:<br>• **Gate.Mapper.Debug**<br>• **Gate.Reader.Debug**<br>• **Gate.Writer.Debug** |
| **Gate.PAAware** *integer* | -paaware *integer* | This property is reserved for use by the Process Agent (PA) and specifies whether the gateway is PA aware. It is included in the properties file for information only. Do not change this property.<br><br>The default is " ", which means the gateway is not PA aware. |
| **Gate.PAAwareName** *string* | -paname *string* | This property is reserved for use by the Process Agent (PA) and specifies the name of the PA that controls the gateway. It is included in the properties file for information only. Do not change this property.<br><br>The default is " ", which means no PA controls the gateway. |
| **Gate.UsePamAuth** *boolean* | -usepamauth *boolean* | UNIX operating systems only: Use this property to specify whether PAM authentication is used.<br><br>The default is FALSE.<br>**Note:** To run the gateway in FIPS 140-2 mode, set this property to TRUE. |
| **Gate.UnixAdminGrp** *string* | -unixadmingroup *string* | UNIX operating systems only: Use this property to specify the administration group to which the gateway must belong if standard UNIX authentication is used.<br><br>The default is ncoadmin. |

*Table 3. Generic ObjectServer Gateway properties and command-line options (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **MaxLogFileSize** *integer* | -maxlogfilesize *integer* | Use this property to specify the size, in KB that the gateway allocates for the log file. When the log file reaches this size, the gateway renames the log file by appending the name with the characters .old and creates a new log file.<br><br>The default is 1024. |
| **OldTimeStamp** *boolean* | -oldtimestamp *boolean* | Use this property to specify old-style timestamp format the gateway uses in the log file. Options are as follows:<br>• TRUE: Specifies the locale-specific timestamp format that is used by Tivoli Netcool/OMNIbus V7.2.1 or earlier. For example, dd/MM/YYYY hh:mm:ss AM or dd/MM/YYYY hh:mm:ss PM when the locale is set to en_GB.<br>• FALSE: Displays the timestamp in ISO 8601 format, which is YYYY-MM-DDThh:mm:ss, where T separates the date and time, and hh is in 24-hour clock.<br><br>The default is FALSE.<br>**Important:** Do not set the **OldTimeStamp** property to TRUE when running in UTF-8 mode. |
| N/A | -utf8enabled *boolean* | |
| **Gate.Mapper.Debug** *boolean* | -mapperdebug *boolean* | Use this property to specify whether the gateway includes mapper debug messages in the debug log.<br><br>The default is TRUE. |
| **Gate.Mapper.Forward HistoricDetails** *boolean* | -mapperforhistdtls *boolean* | Use this property to specify whether the gateway forwards all historic details on converted update.<br><br>The default is FALSE. |
| **Gate.Mapper.Forward HistoricJournals** *boolean* | -mapperforhistjrnl *boolean* | Use this property to specify whether the gateway forwards all historic journals on converted update.<br><br>The default is FALSE. |

**Related concepts**:

Chapter 4, “Additional gateway runtime commands,” on page 41

## Unidirectional gateway properties

In addition to the generic ObjectServer Gateway properties, unidirectional gateways have specific properties.

For information about the common properties and Interprocess Communication (IPC) properties, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* SC14-7530. The following table describes the common gateway properties.

*Table 4. Properties and command line options used by unidirectional gateways*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Reader.CommonNames** *string* | `-readercommonnames` *string* | If the gateway is connecting to an ObjectServer using SSL, and the Common Name field of the received certificate does not match the name specified by the **Gate.Reader.Server** property (for example, in a failover pair or a virtual server setting), use this property to specify a comma-separated list of acceptable SSL Common Names.<br><br>The default setting is to use the **Gate.Reader.Server** property. |
| **Gate.Reader.Debug** *boolean* | `-readerdebug` *boolean* | Use this property to specify whether the gateway includes gateway reader debug messages in the debug log.<br><br>The default is `TRUE`. |
| **Gate.Reader.Description** *string* | `-readerdescription` *string* | Use this property to specify the application description for the reader connection. This description is used in triggers and allows you to determine which component of the gateway attempted to perform an action.<br><br>The default is `"Gateway Reader"`. |
| **Gate.Reader.DetailsTableName** *string* | `-readerdetailstblname` *string* | Use this property to specify the name of the details table that the gateway reads.<br><br>The default is `alerts.details`. |
| **Gate.Reader.FailbackEnabled** *boolean* | `-readerfailbackenabled` *boolean* | Use this property to specify failback for this ObjectServer.<br><br>The default is `TRUE`. |
| **Gate.Reader.FailbackTimeout** *integer* | `-readerfailbacktimeout` *integer* | Use this property to specify the period of time (in seconds) that the gateway must wait before it enters failback mode.<br><br>The default is `30`. |

*Table 4. Properties and command line options used by unidirectional gateways (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Reader.IDUCFlushRate** *integer* | `-readeriducflushrate` *integer* | Use these properties to control how often the gateway looks for changes in the ObjectServer.<br><br>The default is `0`.<br><br>If you leave the property set to `0` the ObjectServer can notify the gateway that changes are pending. If you need the gateway to capture more detailed changes to events, set the property to a different value. If you set the property to a value that is not `0`, the load on the ObjectServer might be increased.<br><br>For more information, see the description of the **Granularity** ObjectServer property in the *IBM Tivoli Netcool/OMNIbus Administration Guide*. |
| **Gate.Reader.JournalTableName** *string* | `-readerjournaltblname` *string* | Use this property to specify the name of the journal table that the gateway reads.<br><br>The default is `alerts.journal`. |
| **Gate.Reader.LogOSSql** *boolean* | `-readerlogossql` *boolean* | Use this property to specify whether the gateway logs all SQL commands sent to the ObjectServer in debug mode.<br><br>The default is `FALSE`. |
| **Gate.Reader.Password** *string* | `-readerpassword` *string* | Use this property to specify the password associated with the user specified by the **Gate.Reader.Username** property.<br><br>The default is `""`.<br>**Note:** If the ObjectServer to which the gateway reads/writes alerts is running on IBM Tivoli Netcool/OMNIbus V7, V7.1, V7.2 or V7.2.1, this password must be encrypted by the `nco_g_crypt` utility. If the ObjectServer from which the gateway reads alerts is running Tivoli Netcool/OMNIbus V7.2.1 in FIPS 140-2 mode, this password must be in plain text or encrypted by the **nco_aes_crypt** utility. For more information about the encryption utilities, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*. |
| **Gate.Reader.ReconnectTimeout** *integer* | `-readerreconntimeout` *integer* | Use this property to specify the time (in seconds) between each reconnection poll attempt that the gateway makes if the connection to the ObjectServer is lost.<br><br>The default is `30`. |

*Table 4. Properties and command line options used by unidirectional gateways (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Reader.Server** *string* | -readerserver *string* | Use this property to specify the name of the ObjectServer from which the gateway reads alerts.<br><br>The default is NCOMS. |
| **Gate.Reader. StatusTableName** *string* | -readerstatustblname *string* | Use this property to specify the name of the status table that the gateway reads.<br><br>The default is alerts.status. |
| **Gate.Reader. TblReplicateDefFile** *string* | -readertblrepdeffile *string* | Use this property to specify the path to the table replication definition file.<br><br>The default is $OMNIHOME/gates/ objserv_uni/ objserv_uni.reader.tblrep.def. |
| **Gate.Reader.Username** *string* | -readerusername *string* | Use this property to specify the username that is used to authenticate the ObjectServer connection.<br><br>The default is root. |
| **Gate.Resync.Enable** *boolean* | -resyncenable *boolean* | Use this property to make the gateway resychronize the ObjectServers after the gateway establishes or restablishes a connection to both ObjectServers. For more information, see "Configuring resynchronization" on page 12.<br><br>The default is TRUE. |
| **Gate.Resync.LockType** *string* | -resynclocktype *string* | Use this property to specify the locking option on the source and destination ObjectServers while resynchronizing events.<br><br>You have the following options:<br>• FULL: The gateway locks both the source and target ObjectServers.<br>• PARTIAL: The gateway only locks the destination ObjectServer.<br>• NONE: The gateway locks neither the source nor the target ObjectServer.<br><br>The default is FULL. |

*Table 4. Properties and command line options used by unidirectional gateways (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Resync.Type** *string* | -resynctype *string* | Use this property to specify the how the gateway resynchronizes table data between ObjectServers when the gateway starts or restores a lost connection. The gateway resynchronizes the tables that are defined in the table replication definition file.<br><br>For more information about the values to which you can set this property, see "Gate.Resync.Type options" on page 24.<br><br>The default is NORMAL. |
| **Gate.Writer.Buffersize** *integer* | -writerbufsize *integer* | Use these properties to specify the maximum number of entries that the gateway stores in the buffer for this ObjectServer before flushing, if buffering is enabled. The gateway flushes the buffer when the end of a batch of SQL statements has been reached regardless of the buffer size. This property can be used to fine-tune the efficiency of the gateway.<br><br>The default is 25. |
| **Gate.Writer. CommonNames** *string* | -writercommonnames *string* | If the gateway is connecting to an ObjectServer using SSL, and the Common Name field of the received certificate does not match the name specified by the **Gate.Writer.Server** property (for example, in a failover pair or a virtual server setting), use this property to specify a comma-separated list of acceptable SSL Common Names.<br><br>The default setting is to use the **Gate.Writer.Server** property. |
| **Gate.Writer.Debug** *boolean* | -writerdebug *boolean* | Use this property to specify whether the gateway includes gateway writer debug messages in the debug log.<br><br>The default is TRUE. |
| **Gate.Writer. Description** *string* | -writerdescription *string* | Use this property to specify the application description for the writer connection. This description is used in triggers and allows you to determine which component of the gateway attempted to perform an action.<br><br>The default is "Gateway Writer". |
| **Gate.Writer.Failback Enabled** *boolean* | -writerfailback enabled *boolean* | Use this property to specify failback for this ObjectServer.<br><br>The default is TRUE. |

*Table 4. Properties and command line options used by unidirectional gateways (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Writer.Failback Timeout** *integer* | `-writerfailback timeout` *integer* | Use this property to specify the time (in seconds) that the gateway allows before checking for the return of the master ObjectServer and failing back.<br><br>The default is `30`. |
| **Gate.Writer.LogOSSql** *boolean* | `-writerlogossql` *boolean* | Use this property to specify whether the gateway logs all SQL commands sent to the ObjectServer in debug mode.<br><br>The default is `FALSE`. |
| **Gate.Writer.Password** *string* | `-writerpassword` *string* | Use this property to specify the password associated with the user specified by the **Gate.Writer.Username** property.<br><br>The default is `""`.<br>**Note:** If the ObjectServer to which the gateway reads/writes alerts is running on IBM Tivoli Netcool/OMNIbus V7, V7.1, V7.2 or V7.2.1, this password must be encrypted by the `nco_g_crypt` utility. For more information about the encryption utilities, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*. |
| **Gate.Writer. ReconnectTimeout** *integer* | `-writerreconntimeout` *integer* | Use this property to specify the time (in seconds) between each reconnection poll attempt if the gateway loses the connection to the ObjectServer.<br><br>The default is `30`. |
| **Gate.Writer. RefreshCache OnUpdate** *boolean* | `-writerrefcacheonupd` *boolean* | Use this property to specify whether the hash table cache for this ObjectServer is refreshed.<br><br>You have the following options:<br>• `TRUE`: The cache is resynchronized with the target ObjectServer prior to processing each collection of row updates for a table in the current IDUC window.<br>• `FALSE`: The gateway assumes that its cache is accurate and does not resynchronize it.<br><br>The default is `TRUE`. |
| **Gate.Writer.SAF** *boolean* | `-writersaf` *boolean* | Use this property to specify that the gateway stores all table entries if the destination ObjectServer is unavailable and forwards them when the ObjectServer becomes available again.<br><br>The default is `FALSE`. |

*Table 4. Properties and command line options used by unidirectional gateways (continued)*

| Property name | Command line option | Description |
|---|---|---|
| **Gate.Writer.SAFFile** *string* | -writersaffile *string* | Use this property to specify the name of the file that the gateway uses to store table entries while the destination ObjectServer is unavailable.<br><br>The default is $OMNIHOME/var/objserv_uni_NCO_GATE_Writer.store. **Note:** This file is only used if the **Gate.Writer.SAF** property is set to TRUE. |
| **Gate.Writer.Server** *string* | -writerserver *string* | Use this property to specify the name of the ObjectServer to which the gateway writes alerts.<br><br>The default is REMOTE. |
| **Gate.Writer.Username** *string* | -writerusername *string* | Use this property to specify the username that is used to authenticate the ObjectServer connection. This username is used to establish both the writer's IDUC connection and the subsidiary SQL command connection.<br><br>The default is root. |
| **Gate.Writer.SAF ReplayOnResync** *boolean* | -writersafreplay onresync *boolean* | Use this property to specify how store-and-forward (SAF) replays on resynchronization.<br><br>You have the following options:<br>• TRUE: SAF replays regardless of whether **Gate.Resync.Enable** has been set to TRUE.<br>• FALSE: SAF replays only when **Gate.Resync.Enable** has been set to FALSE.<br><br>The default is FALSE. |
| **Gate.Writer. UseBulkInsCmd** *boolean* | -usebulkinscmd *boolean* | Use this property to specify bulk inserts for faster resynchronization.<br><br>You have the following options:<br>• TRUE: The gateway changes the format of the insert statement to enable the ObjectServer to process bulk inserts more efficiently.<br>• FALSE: The gateway makes no changes to the insert statement before sending events to the ObjectServer.<br><br>The default is FALSE. |

## Gate.Resync.Type options

You can set the **Gate.Resync.Type** property to one of the following values:

- NORMAL: For each table, the gateway deletes all the data from the slave ObjectServer. Then, the gateway retransfers the full set of tables from the master to the slave. With this type of resynchronization, the master and slave are fully synchronized. However, table rows that are in the slave but not in the master are lost. Additionally, if table rows are in the master and the slave, the copy of the row that is on the master is retained on both the master and the slave. Any prior updates to the row on the slave are lost.
- UPDATE: For each table, the gateway builds a cache that contains all rows in the master and slave ObjectServers. Then, the gateway examines the contents of the cache for each table and compares the row data from the master with the row data from the slave. The data is resynchronized as follows:
  - Rows in the slave that are also in the master are updated with the data from the master, if the data in the master is different from the slave.
  - Rows that are in the master but not in the slave are copied to the slave.
  - Rows in the slave that are not in the master are retained.

  With this type of resynchronization, no events are lost, but the master and slave ObjectServers might not be fully synchronized.
- MINIMAL: This option behaves in the same way as UPDATE. In addition, events (that is, rows in the alerts.status table) that are in the slave but not in the master are marked for deletion. To mark these events for deletion, the gateway behaves as follows:
  1. For each row of the alerts.status table in the slave ObjectServer that is not in the master, the OldRow field is set to 1.
  2. The pass_deletes trigger runs on the slave ObjectServer and deletes all rows in which the OldRow field is set to 1.

  The benefit of a MINIMAL resynchronization is that the master and slave ObjectServers are fully synchronized but less data is sent during the resynchronization process. MINIMAL resynchronization is less data-intensive because all the rows are not deleted and then recopied, unlike a NORMAL resynchronization.

## Bidirectional gateway properties

In addition to the generic ObjectServer Gateway properties, bidirectional gateways have specific properties.

*Table 5. Properties and command-line options used by bidirectional gateways*

| Property name | Command-line option | Description |
|---|---|---|
| **Gate.ObjectServerA.Buffersize** *integer*<br><br>**Gate.ObjectServerB.Buffersize** *integer* | -objectserverabufsize *integer*<br><br>-objectserverbbufsize *integer* | Use these properties to specify the maximum number of entries that the gateway stores in the buffer for this ObjectServer before flushing, if buffering is enabled. The gateway flushes the buffer when the end of a batch of SQL statements has been reached regardless of the buffer size. This property can be used to fine-tune the efficiency of the gateway.<br><br>The default is 25. |

*Table 5. Properties and command-line options used by bidirectional gateways (continued)*

| Property name | Command-line option | Description |
|---|---|---|
| **Gate.ObjectServerA.CommonNames** *string*<br><br>**Gate.ObjectServerB.CommonNames** *string* | -objectservera commonnames *string*<br><br>-objectserverb commonnames *string* | If the gateway is connecting to an ObjectServer using SSL, and the Common Name field of the received certificate does not match the name specified by the **Gate.ObjectServerA.Server** property or **Gate.ObjectServerB.Server** (for example, in a failover pair or a virtual server setting), use these properties to specify a comma-separated list of acceptable SSL Common Names.<br><br>The default setting is to use the **Gate.ObjectServerA.Server** or **Gate.ObjectServerB.Server** property. |
| **Gate.ObjectServerA.Debug** *boolean*<br><br>**Gate.ObjectServerB.Debug** *boolean* | -objectserveradebug *boolean*<br><br>-objectserverbdebug *boolean* | Use these properties to specify whether the gateway includes debug messages for this ObjectServer in the gateway debug log.<br><br>The default is TRUE. |
| **Gate.ObjectServerA.DeleteIfNoDedup** *boolean*<br><br>**Gate.ObjectServerB.DeleteIfNoDedup** *boolean* | -objectservera delifnodedup *boolean*<br><br>-objectserverb delifnodedup *boolean* | Use these properties to specify how the gateway forwards deletes.<br><br>You have the following options:<br>• FALSE: The delete is always applied<br>• TRUE: The delete is not applied if the event in the target server indicates that the event has occurred again since the delete was issued.<br><br>The default is FALSE. |
| **Gate.ObjectServerA.Description** *string*<br><br>**Gate.ObjectServerB.Description** *string* | -objectservera description *string*<br><br>-objectserverb description *string* | Use these properties to specify an application description for the connection to ObjectServer A. This description is used in triggers and allows you to determine which component of the gateway attempted to perform an action.<br><br>The default is "Gateway Reader/Writer".<br><br>In a bidirectional ObjectServer gateway configuration, where the gateway connects a primary ObjectServer and a backup ObjectServer, set this property to failover_gate. |
| **Gate.ObjectServerA.DetailsTableName** *string*<br><br>**Gate.ObjectServerB.DetailsTableName** *string* | -objectservera detailstblname *string*<br><br>-objectserverb detailstblname *string* | Use these properties to specify the name of the details table that the gateway reads.<br><br>The default is alerts.details. |

*Table 5. Properties and command-line options used by bidirectional gateways (continued)*

| Property name | Command-line option | Description |
|---|---|---|
| **Gate.ObjectServerA.FailbackEnabled** *boolean*<br><br>**Gate.ObjectServerB.FailbackEnabled** *boolean* | -objectservera failbackenabled *boolean*<br><br>-objectserverb failbackenabled *boolean* | Use these properties to enable failback for this ObjectServer.<br><br>The default is FALSE. |
| **Gate.ObjectServerA.FailbackTimeout** *integer*<br><br>**Gate.ObjectServerB.FailbackTimeout** *integer* | -objectservera failbacktimeout*integer*<br><br>-objectserverb failbacktimeout*integer* | Use these properties to specify the period of time (in seconds) that the gateway must wait before it checks whether the master ObjectServer is back up, so that the gateway can fail back to the master ObjectServer.<br><br>The default is 30. |
| **Gate.ObjectServerA.IDUCFlushRate** *integer*<br><br>**Gate.ObjectServerB.IDUCFlushRate** *integer* | -objectservera iducflushrate *integer*<br><br>-objectserverb iducflushrate *integer* | Use these properties to control how often the gateway looks for changes in the ObjectServer.<br><br>The default is 0.<br><br>If you leave the property set to 0, the ObjectServer can notify the gateway that changes are pending. If you need the gateway to capture more detailed changes to events, set the property to a different value. If you set the property to a value that is not 0, the load on the ObjectServer might be increased. |
| **Gate.ObjectServerA.JournalTableName** *string*<br><br>**Gate.ObjectServerB.JournalTableName** *string* | -objectservera journaltblname *string*<br><br>-objectserverb journaltblname *string* | Use these properties to specify the name of the journal table that the gateway reads.<br><br>The default is alerts.journal. |
| **Gate.ObjectServerA.LogOSSql** *boolean*<br><br>**Gate.ObjectServerB.LogOSSql** *boolean* | -objectservera logossql *boolean*<br><br>-objectserverb logossql *boolean* | Use these properties to specify whether the gateway logs all SQL commands sent to this ObjectServer in debug mode.<br><br>The default is FALSE. |

*Table 5. Properties and command-line options used by bidirectional gateways (continued)*

| Property name | Command-line option | Description |
|---|---|---|
| **Gate.ObjectServerA. Password** *string*<br><br>**Gate.ObjectServerB. Password** *string* | -objectservera password *string*<br><br>-objectserverb password *string* | Use these properties to specify the password associated with the user specified by the **Gate.ObjectServerA.Username** property or the **Gate.ObjectServerB.Password** property.<br><br>The default is "".<br>**Note:** If the ObjectServer to which the gateway reads/writes alerts is running on IBM Tivoli Netcool/OMNIbus V7, V7.1, V7.2 or V7.2.1, this password must be encrypted by the nco_g_crypt utility. If the ObjectServer from which the gateway reads alerts is running Tivoli Netcool/OMNIbus V7.2.1 in FIPS 140-2 mode, this password must be in plain text or encrypted by the **nco_aes_crypt** utility. For more information about the encryption utilities, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*. |
| **Gate.ObjectServerA. ReconnectTimeout** *integer*<br><br>**Gate.ObjectServerB. ReconnectTimeout** *integer* | -objectservera reconntimeout *integer*<br><br>-objectserverb reconntimeout *integer* | Use these properties to specify the time, in seconds, between each reconnection poll attempt if the connection to this ObjectServer is lost.<br><br>The default is 30. |
| **Gate.ObjectServerA. RefreshCacheOn Update** *boolean*<br><br>**Gate.ObjectServerB. RefreshCacheOn Update** *boolean* | -objectservera refcacheonupd *boolean*<br><br>-objectserverb refcacheonupd *boolean* | Use these properties to specify how the hash table cache is refreshed for this ObjectServer.<br><br>You have the following options:<br>• TRUE: The cache is resynchronized with the target ObjectServer before each collection of row updates for a table in the current IDUC window is processed<br>• FALSE: The gateway assumes that its cache is accurate and does not resynchronize it<br><br>The default is FALSE. |
| **Gate.ObjectServerA. SAF** *boolean*<br><br>**Gate.ObjectServerB. SAF** *boolean* | -objectserverasaf *boolean*<br><br>-objectserverbsaf *boolean* | Use these properties to specify whether the gateway stores all changes if the destination ObjectServer is unavailable and to forward them when the ObjectServer becomes available again.<br><br>The default is FALSE. |

*Table 5. Properties and command-line options used by bidirectional gateways (continued)*

| Property name | Command-line option | Description |
|---|---|---|
| **Gate.ObjectServerA.SAFFile** *string*<br><br>**Gate.ObjectServerB.SAFFile** *string* | -objectserverasaffile *string*<br><br>-objectserverbsaffile *string* | Use these properties to specify the name of the file in which the gateway stores changes while the destination ObjectServer is unavailable. The default is as follows:<br>• For the **Gate.ObjectServerA.SAFFile** property, the default is $OMNIHOME/var/objserv_bi/NCO_GATE_ObjectServerA.store.<br>• For the **Gate.ObjectServerB.SAFFile**, property, the default is $OMNIHOME/var/objserv_bi/NCO_GATE_ObjectServerB.store.<br>This file is used only if the **Gate.ObjectServerA.SAF** property or **Gate.ObjectServerB.SAF** property is set to TRUE. |
| **Gate.ObjectServerA.SAFReplayOnResync** *boolean*<br><br>**Gate.ObjectServerB.SAFReplayOnResync** *boolean* | -objectservera safreplayonresync *boolean*<br><br>-objectserverb safreplayonresync *boolean* | Use these properties to specify how store-and-forward (SAF) file for ObjectServerA replays before resynchronization.<br>You have the following options:<br>• TRUE: SAF replays regardless of whether **Gate.Resync.Enable** has been set to TRUE.<br>• FALSE: SAF replays only when **Gate.Resync.Enable** has been set to FALSE<br>The default is FALSE. |
| **Gate.ObjectServerA.Server** *string*<br><br>**Gate.ObjectServerB.Server** *string* | -objectserveraserver *string*-objectserverbserver *string* | Use these properties to specify the name of the ObjectServer that the gateway connects to.<br>The default is NCOMS. |
| **Gate.ObjectServerA.StatusTableName** *string*<br><br>**Gate.ObjectServerB.StatusTableName** *string* | -objectservera statustblname *string*<br><br>-objectserverb statustblname *string* | Use these properties to specify the name of the status table that the gateway reads.<br>The default is alerts.status. |

*Table 5. Properties and command-line options used by bidirectional gateways (continued)*

| Property name | Command-line option | Description |
|---|---|---|
| **Gate.ObjectServerA. TblReplicateDefFile** *string*<br><br>**Gate.ObjectServerB. TblReplicateDefFile** *string* | -objectservera tblrepdeffile *string*<br><br>-objectserverb tblrepdeffile *string* | Use these properties to specify the path to the table replication definition file.<br><br>The default is as follows:.<br>• For the **Gate.ObjectServerA. TblReplicateDefFile** property, the default is $OMNIHOME/gates/objserv_bi/objserv_bi.objectservera.tblrep.def<br>• For the **Gate.ObjectServerB. TblReplicateDefFile** property, the default is $OMNIHOME/gates/objserv_bi/objserv_bi.objectservera.tblrep.def |
| **Gate.ObjectServerA. UseBulkInsCmd** *boolean*<br><br>**Gate.ObjectServerB. UseBulkInsCmd** *boolean* | -usebulkinscmd *boolean*<br><br>-usebulkinscmd *boolean* | Use these properties to specify bulk inserts for faster resynchronization.<br><br>You have the following options:<br>• TRUE: The gateway changes the format of the insert statement it sends to ObjectServerA, which allows ObjectServerA to process the inserts more efficiently.<br>• FALSE: The gateway makes no changes to the insert statement before sending events to ObjectServerA.<br><br>The default is FALSE. |
| **Gate.ObjectServerA. Username** *string*<br><br>**Gate.ObjectServerB. Username** *string* | -objectservera username *string*<br><br>-objectserverb username *string* | Use these properties to specify the user name that is used to authenticate the connection to the ObjectServer.<br><br>The default is root. |
| **Gate.Resync.Enable** *boolean* | -resyncenable *boolean* | Use this property to make the gateway resychronize the ObjectServers after the gateway establishes or restablishes a connection to both ObjectServers. For more information, see “Configuring resynchronization” on page 12.<br><br>The default is TRUE. |
| **Gate.Resync.LockType** *string* | -resynclocktype *string* | Use this property to specify the locking option on the source and destination ObjectServers while resynchronizing events.<br><br>You have the following options:<br>• FULL: The gateway locks both the source and target ObjectServers.<br>• PARTIAL: The gateway only locks the destination ObjectServer.<br>• NONE: The gateway locks neither the source nor the target ObjectServer.<br><br>The default is FULL. |

*Table 5. Properties and command-line options used by bidirectional gateways (continued)*

| Property name | Command-line option | Description |
|---|---|---|
| **Gate.Resync.Master** *string* | -resyncmaster *string* | Use this property to specify which ObjectServer the gateway uses as the master during resynchronization. To use the ObjectServer that has been running the longest, leave the property as an empty string. To use a named ObjectServer, regardless of which ObjectServer has been running the longest, set the property to ObjectServerA or ObjectServerB. |
| **Gate.Resync.Preferred** *string* | -resyncpreferred *string* | Use this property to specify which ObjectServer the gateway uses as the master during resynchronization if the **Gate.Resynch.Master** is left empty and both ObjectServers have been running for the same length of time. To use a named ObjectServer, set the property to ObjectServerA or ObjectServerB. |
| **Gate.Resync.Type** *string* | -resynctype *string* | Use this property to specify the how the gateway resynchronizes table data between ObjectServers when the gateway starts or restores a lost connection. The gateway resynchronizes the tables that are defined in the table replication definition file.<br><br>For more information about the values to which you can set this property, see "Gate.Resync.Type options."<br><br>The default is NORMAL. |

## Gate.Resync.Type options

You can set the **Gate.Resync.Type** property to one of the following values:

- NORMAL: For each table, the gateway deletes all the data from the slave ObjectServer. Then, the gateway retransfers the full set of tables from the master to the slave. With this type of resynchronization, the master and slave are fully synchronized. However, table rows that are in the slave but not in the master are lost. Additionally, if table rows are in the master and the slave, the copy of the row that is on the master is retained on both the master and the slave. Any prior updates to the row on the slave are lost.
- UPDATE: For each table, the gateway builds a cache that contains all rows in the master and slave ObjectServers. Then, the gateway examines the contents of the cache for each table and compares the row data from the master with the row data from the slave. The data is resynchronized as follows:
  - Rows in the slave that are also in the master are updated with the data from the master, if the data in the master is different from the slave.
  - Rows that are in the master but not in the slave are copied to the slave.
  - Rows in the slave that are not in the master are retained.

  With this type of resynchronization, no events are lost, but the master and slave ObjectServers might not be fully synchronized.

- `MINIMAL`: This option behaves in the same way as `UPDATE`. In addition, events (that is, rows in the alerts.status table) that are in the slave but not in the master are marked for deletion. To mark these events for deletion, the gateway behaves as follows:
  1. For each row of the alerts.status table in the slave ObjectServer that is not in the master, the OldRow field is set to 1.
  2. The pass_deletes trigger runs on the slave ObjectServer and deletes all rows in which the OldRow field is set to 1.

  The benefit of a MINIMAL resynchronization is that the master and slave ObjectServers are fully synchronized but less data is sent during the resynchronization process. `MINIMAL` resynchronization is less data-intensive because all the rows are not deleted and then recopied, unlike a NORMAL resynchronization.

# Chapter 3. ObjectServer Gateway mapping

The gateway can replicate any table in the ObjectServer. To do this, the gateway maps data to the appropriate fields in the ObjectServer using a map definition file.

To replicate user related system tables, for example, SecurityUsers, SecurityGroups, SecurityRoles, SecurityRoleGrants, and SecurityGroupMembers, include details of these mappings in the map file.

The path of the map definition file is determined by the **Gate.Mapfile** property in the properties file.

The gateway reads this table only at startup. If you edit this table while the gateway is running, restart the gateway so that the changes take effect.

The following map definition file conversion functions can be used in the map definition file:

- TO_STRING ( <column_name>)
- TO_INTEGER ( <column_name>)
- TO_TIME ( <column_name>)

## Syntax

Mappings for use with the ObjectServer writer must adhere to the following syntax:

```
CREATE MAPPING mappingname
(
'dest_fieldname' = '@src_fieldname' | simple_expression | attribute
[ ON INSERT ONLY ] [ CONVERT TO type ] [ NOT NULL '@src_fieldname'],
...
) ;
```

Where:

- *mappingname* is the name of the mapping to be created.
- *dest_fieldname* is the name of the field to be written in the destination ObjectServer.
- *src_fieldname* is the name of a field in the ObjectServer alerts.status table.
- *simple_expression* is a string, boolean or integer; or sequence of strings or integers joined by the operators +, -, *, /. All operators concatenate strings. All operators work from left to right when operating on integers.
- *attribute* is an attribute name.

The following commands are optional:

**ON INSERT ONLY**
Controls the updating of the field during the life of the alert. If you omit this command, the field is updated when any change in the state of the alert occurs. If you include this command, the field is created once for the alert, but is never updated. Use this command only when setting the values of variables.

**CONVERT TO**
Defines a forced conversion if a source field may not match the type of the destination field. Possible options are INTEGER, STRING, or DATE.

**NOT NULL**
Provides an alterative value that is used if the source value is zero or the empty string. The alterative value can be a column or constant, but not an expression

# Mapping attributes

You use attribute names to include additional data in mapping definitions. You can specify two types of attribute: cache value access attributes or dynamic attributes.

## Cache value access attributes

The gateway uses cache value attributes to access values that are stored in the cross-reference cache. The following table describes the cache value attributes that can be used in mapping definitions.

*Table 6. Cache value access attributes*

| Attribute name | Description |
|---|---|
| STATUS.SERIAL | Cached serial number for the status table row that is associated with the current journal or details table row. |
| STATUS.SERVER_SERIAL | Cached server serial number for the status table row that is associated with the current journal or details table row. |
| STATUS.SERVER_NAME | Cached server name for the status table row that is associated with the current journal or details table row. |
| STATUS.IDENTIFIER | Cached identifier for the status table row that is associated with the current journal or details table row. |
| JOURNAL.SERIAL | Cached serial number of the journal table row. |
| DETAILS.IDENTIFIER | Cached identifier of the details table row. |

## Dynamic attributes

Dynamic attributes enable the gateway to access dynamic values that are automatically generated by the gateway. The following table describes the dynamic attributes that can be used in mapping definitions.

*Table 7. Dynamic attributes*

| Attribute name | Description |
|---|---|
| ACTION_CODE | This attribute displays a single character string that specifies the type of operation performed. Valid values are:<br>• I: Insert<br>• U: Update<br>• D : Delete |
| ACTION_TIME | This attribute displays the time in UTC that the action occurred. |

*Table 7. Dynamic attributes (continued)*

| Attribute name | Description |
|---|---|
| DELETEDAT | This attribute displays the date on which the row was deleted, if applicable. |

## Example mapping

Mappings define how the gateways replicate tables by assigning data to appropriate fields in the ObjectServer.

The following example shows the mappings for the ObjectServer tables into which the gateway writes. In this example, the fields on the left side of the equals sign (=) are the target ObjectServer (ObjectServer B) and the fields on the right side of the equals sign are the source ObjectServer (ObjectServer A).

```
CREATE MAPPING StatusMap
(
 'Identifier' =  '@Identifier'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',
 'FirstOccurrence' = '@FirstOccurrence'  ON INSERT ONLY,
 'LastOccurrence' =  '@LastOccurrence',
 'InternalLast' =  '@InternalLast',
 'Poll'  =  '@Poll'  ON INSERT ONLY,
 'Type'  =  '@Type'  ON INSERT ONLY,
 'Tally'  =  '@Tally',
 'ProbeSubSecondId' =  '@ProbeSubSecondId',
 'Class'  =  '@Class'  ON INSERT ONLY,
 'Grade'  =  '@Grade' ON INSERT ONLY,
 'Location' =  '@Location' ON INSERT ONLY,
 'OwnerUID' =  '@OwnerUID',
 'OwnerGID' =  '@OwnerGID',
 'Acknowledged' =  '@Acknowledged',
 'BSM_Identity' = '@BSM_Identity'
 'Flash' =  '@Flash',
 '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',
 'NmosObjInst' =  '@NmosObjInst',
 'NmosCauseType' =  '@NmosCauseType',
 'NmosDomainName' =  '@NmosDomainName',
 'NmosEntityId' =  '@NmosEntityId',
 'NmosManagedStatus' =  '@NmosManagedStatus',
 'NmosEventMap' =  '@NmosEventMap',
 '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'  ON INSERT ONLY,
 'X733SpecificProb' =  '@X733SpecificProb' ON INSERT ONLY,
 'X733CorrNotif' =  '@X733CorrNotif'  ON INSERT ONLY,
 'URL' =  '@URL'   ON INSERT ONLY,
 'ExtendedAttr' =  '@ExtendedAttr'  ON INSERT ONLY,
 'ServerName' =  '@ServerName'  ON INSERT ONLY,
 'ServerSerial' =  '@ServerSerial'  ON INSERT ONLY
);


CREATE MAPPING JournalMap
(
 'KeyField' = TO_STRING(STATUS.SERIAL) + ":" +
    TO_STRING('@UID') + ":" +
     TO_STRING('@Chrono')ON INSERT ONLY,
 'Serial' =  STATUS.SERIAL,
 'Chrono' =  '@Chrono',
 'UID'  =  TO_INTEGER('@UID'),
 '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'
);


CREATE MAPPING DetailsMap
(
 'KeyField' = '@Identifier' + '####' +
   TO_STRING('@Sequence') ON INSERT ONLY,
 'Identifier' =  '@Identifier',
 'AttrVal' =  '@AttrVal',
 'Sequence' =  '@Sequence',
 'Name'  =  '@Name',
 'Detail' =  '@Detail'
);


CREATE MAPPING IducMap
(
 'ServerName' = '@ServerName' ON INSERT ONLY,
 'AppName' = '@AppName',
 'AppDesc' = '@AppDesc' ON INSERT ONLY,
 'ConnectionId' = '@ConnectionId' ON INSERT ONLY,
 'LastIducTime' = '@LastIducTime'
);


# CREATE MAPPING SecurityUsersMap
# (
#  'UserID' = '@UserID' ON INSERT ONLY,
```

```
#  'UserName' = '@UserName',
#  'SystemUser' = '@SystemUser',
#  'FullName' = '@FullName',
#  'Passwd' = '@Passwd',
#  'UsePAM' = '@UsePAM',
#  'Enabled' = '@Enabled'
# );
#
#
# CREATE MAPPING SecurityGroupsMap
# (
#  'GroupID' = '@GroupID'  ON INSERT ONLY,
#  'GroupName' = '@GroupName',
#  'SystemGroup' = '@SystemGroup',
#  'Description' = '@Description'
# );
#
#
# CREATE MAPPING SecurityRolesMap
# (
#  'RoleID' = '@RoleID' ON INSERT ONLY,
#  'RoleName' = '@RoleName',
#  'SystemRole' = '@SystemRole',
#  'Description' = '@Description',
#  'RoleScope' = '@RoleScope'
# );
#
#
# CREATE MAPPING SecurityRoleGrantsMap
# (
#  'GranteeType' = '@GranteeType' ON INSERT ONLY,
#  'GranteeID' = '@GranteeID' ON INSERT ONLY,
#  'RoleID' = '@RoleID' ON INSERT ONLY
# );
#
#
# CREATE MAPPING SecurityGroupMembersMap
# (
#  'UserID' =  '@UserID' ON INSERT ONLY,
#  'GroupID' =  '@GroupID'ON INSERT ONLY,
#  'Compat' =  '@Compat'
# );
#
#
# CREATE MAPPING CatalogRestrictionFiltersMap
# (
#  'RestrictionName' =  '@RestrictionName' ON INSERT ONLY,
#  'TableName'   =  '@TableName',
#  'DatabaseName'   =  '@DatabaseName',
# 'ConditionText'   =  '@ConditionText',
# 'CreationText'   =  '@CreationText'
# );
#
#
# CREATE MAPPING SecurityRestrictionFiltersMap
# (
# 'GranteeType'  = '@GranteeType' ON INSERT ONLY,
# 'GranteeID'  = '@GranteeID' ON INSERT ONLY,
# 'RestrictionName' = '@RestrictionName',
# 'DatabaseName'  =  '@DatabaseName',
# 'TableName'  = '@TableName'
# );
#
#
# CREATE MAPPING SecurityPermissionsMap
# (
# 'ApplicationID'  = '@ApplicationID' ON INSERT ONLY,
```

```
# 'ObjectType'  = '@ObjectType' ON INSERT ONLY,
# 'Object'  = '@Object' ON INSERT ONLY,
# 'GranteeType'  = '@GranteeType' ON INSERT ONLY,
# 'GranteeID'  = '@GranteeID' ON INSERT ONLY,
# 'Allows'  = '@Allows',
# 'Denies'  = '@Denies',
# 'GrantOptions'  = '@GrantOptions'
# );
#
#
# CREATE MAPPING ToolsMenusMap
# (
# 'MenuID'  = '@MenuID' ON INSERT ONLY,
# 'Name'   = '@Name',
# 'Owner'   = '@Owner',
# 'Enabled'  =  '@Enabled'
# );
#

# # CREATE MAPPING ToolsMenuItemsMap
# (
# 'KeyField'  = TO_STRING('@MenuID') + ":" +
#     TO_STRING('@MenuItemID')
#       ON INSERT ONLY,
# 'MenuID'  = '@MenuID' ON INSERT ONLY,
# 'MenuItemID'  = '@MenuItemID' ON INSERT ONLY,
# 'Title'   = '@Title',
# 'Description'  = '@Description',
# 'Enabled'  = '@Enabled',
# 'InvokeType'  = '@InvokeType',
# 'InvokeID'  = '@InvokeID',
# 'Position'  = '@Position',
# 'Accelerator'  =  '@Accelerator'
# );
#
#
# CREATE MAPPING ToolsActionsMap
# (
# 'ActionID'  = '@ActionID' ON INSERT ONLY,
# 'Name'   = '@Name',
# 'Owner'   = '@Owner',
# 'Enabled'  = '@Enabled',
# 'Description1'  = '@Description1',
# 'Description2'  = '@Description2',
# 'Description3'  = '@Description3',
# 'Description4'  = '@Description4',
# 'HasInternal'  = '@HasInternal',
# 'InternalEffect1' = '@InternalEffect1',
# 'InternalEffect2' = '@InternalEffect2',
# 'InternalEffect3' = '@InternalEffect3',
# 'InternalEffect4' = '@InternalEffect4',
# 'InternalForEach' = '@InternalForEach',
# 'HasExternal'  = '@HasExternal',
# 'ExternalEffect1' = '@ExternalEffect1',
# 'ExternalEffect2' = '@ExternalEffect2',
# 'ExternalEffect3' = '@ExternalEffect3',
# 'ExternalEffect4' = '@ExternalEffect4',
# 'ExternalForEach' = '@ExternalForEach',
# 'RedirectOut'  = '@RedirectOut',
# 'RedirectErr'  = '@RedirectErr',
# 'Platform'  = '@Platform',
# 'JournalText1'  = '@JournalText1',
# 'JournalText2'  = '@JournalText2',
# 'JournalText3'  = '@JournalText3',
# 'JournalText4'  = '@JournalText4',
# 'JournalForEach' = '@JournalForEach',
# 'HasForcedJournal' = '@HasForcedJournal'
```

```
# );
#
#
# CREATE MAPPING ToolsActionAccessMap
# (
# 'ActionAccessID' = '@ActionAccessID' ON INSERT ONLY,
# 'ActionID'  =  '@ActionID',
# 'GID'   = '@GID',
# 'ClassID'  = '@ClassID'
# );
#
#
# CREATE MAPPING ToolsMenuDefsMap
# (
# 'Name'   = '@Name' ON INSERT ONLY,
# 'DatabaseName'  =  '@DatabaseName',
# 'TableName'  = '@TableName',
# 'ShowField'  = '@ShowField',
# 'AssignField'  = '@AssignField',
# 'OrderbyField'  = '@OrderbyField',
# 'WhereClause'  = '@WhereClause'
# );
#
# CREATE MAPPING ToolsPromptDefsMap
# (
# 'Name'   = '@Name' ON INSERT ONLY,
# 'Prompt'  =  '@Prompt',
# 'Default'  = '@Default',
# 'Value'   = '@Value',
# 'Type'   = '@Type'
# );
#
#
# CREATE MAPPING AlertsConversionsMap
# (
# 'KeyField'  = '@KeyField' ON INSERT ONLY,
# 'Colname'  =  '@Colname' ON INSERT ONLY,
# 'Value'   = '@Value' ON INSERT ONLY,
# 'Conversion'  = '@Conversion'
# );
#
# CREATE MAPPING AlertsColVisualsMap
# (
# 'Colname'  = '@Colname' ON INSERT ONLY,
# 'Title'   =  '@Title',
# 'DefWidth'  = '@DefWidth',
# 'MaxWidth'  = '@MaxWidth',
# 'TitleJustify'  = '@TitleJustify',
# 'DataJustify'  = '@DataJustify'
# );
#
#
# CREATE MAPPING AlertsColorsMap
# (
# 'Severity'  = '@Severity' ON INSERT ONLY,
# 'AckedRed'  =  '@AckedRed',
# 'AckedGreen'  = '@AckedGreen',
# 'AckedBlue'  = '@AckedBlue',
# 'UnackedRed'  = '@UnackedRed',
# 'UnackedGreen'  = '@UnackedGreen',
# 'UnackedBlue'  = '@UnackedBlue'
# );
#
#
# CREATE MAPPING MasterServergroupsMap
# (
```

```
# 'ServerName'  ='@ServerName' ON INSERT ONLY,
# 'GroupID'  ='@GroupID',
# 'Weight'='@Weight'
# );
```

# Chapter 4. Additional gateway runtime commands

On UNIX operating systems, you can use the SQL Interactive Interface (the **nco_sql** utility) to run commands against the gateway. In addition to the ObjectServer SQL commands, you can run the following runtime commands against the gateway: SHOW PROPS, GET CONFIG, and FAILOVER SYNCH. These commands are in the gateway command file.

The following information is important for running SQL commands against the gateway:

- To use the **nco_sql** utility against the gateway, the `-server` command-line option must specify the gateway server name that is given in the data connections file, `omni.dat`.
- If the **Gate.UsePamAuth** property is set to `FALSE`, the user that runs the gateway process must have permission to read the user database. Depending on your operating system configuration, the user needs permission to read `etc/passwd`, `etc/shadow`, and `etc/group`.
- If the **Gate.UsePamAuth** property is set to `TRUE`, the `nco_g_objserv_bi` PAM service or the `nco_g_objserv_uni` PAM service must be configured for the `auth` and `account` modules.
- The user name that is used to start the **nco_sql** utility must be a UNIX user that is specified by the **Gate.UnixAdminGrp** property.

**Restriction:** You cannot use the SQL Interactive Interface to run commands against the gateway if your environment is in FIPS 140-2 mode.

For more information about the SQL Interactive Interface, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

**Related reference**:

"Generic ObjectServer Gateway properties" on page 16

## GET CONFIG

Use the **GET CONFIG** command to display the current configuration of the gateway by listing all properties and their values.

**GET CONFIG** is identical to the **SHOW PROPS** command; it may be removed from later versions of the ObjectServer gateway.

### Syntax

```
GET CONFIG ;
```

### Example

```
GET CONFIG ;
go
```

## SHOW PROPS

Use the **SHOW PROPS** command to display the current configuration of the gateway by listing all properties and their values.

### Syntax

**SHOW PROPS ;**

### Example

```
SHOW PROPS ;
go
```

## FAILOVER SYNCH

Use the **FAILOVER SYNCH** command to synchronize data between primary and backup ObjectServers. The command specifies which master tables are transferred during the data transfer operation.

For information about ObjectServer failover, see the *IBM Tivoli Netcool/OMNIbus Administration Guide* (SC14-7527).

### Syntax

**FAILOVER_SYNC [ ADD 'TABLENAME' TO | REMOVE 'TABLENAME' FROM ] WRITERNAME ;**

### Example

```
FAILOVER_SYNC ADD 'master.names' TO ObjectServerA;
FAILOVER_SYNC ADD 'master.groups'TO ObjectServerA;
FAILOVER_SYNC ADD 'master.members' TO ObjectServerA;
FAILOVER_SYNC ADD 'master.permissions' TO ObjectServerA;
FAILOVER_SYNC ADD 'master.profiles'TO ObjectServerA;
FAILOVER_SYNC ADD 'tools.actions' TO ObjectServerA;
FAILOVER_SYNC ADD 'tools.action_access' TO ObjectServerA;
FAILOVER_SYNC ADD 'tools.menus' TO ObjectServerA;
FAILOVER_SYNC ADD 'tools.menu_defs' TO ObjectServerA;
FAILOVER_SYNC ADD 'tools.menu_items' TO ObjectServerA;
FAILOVER_SYNC ADD 'tools.prompt_defs'TO ObjectServerA;
FAILOVER_SYNC ADD 'alerts.conversions' TO ObjectServerA;
FAILOVER_SYNC ADD 'alerts.col_visuals' TO ObjectServerA;
FAILOVER_SYNC ADD 'alerts.colors' TO ObjectServerA;
FAILOVER_SYNC ADD 'alerts.objclass' TO ObjectServerA;
FAILOVER_SYNC ADD 'alerts.objmenus' TO ObjectServerA;
FAILOVER_SYNC ADD 'alerts.objmenuitems'TO ObjectServerA;
```

## SET LOG LEVEL

The SET LOG LEVEL command can be used in the startup command file. The command does not take effect immediately; an alternative is to configure the properties file.

### Syntax

```
SET LOG LEVEL TO FATAL|ERROR|WARNING|INFORMATION|DEBUG;
```

### Example

```
SET LOG LEVEL TO ERROR ;
go
```

**Related tasks**:
"Setting the level of debug messages" on page 6

# Chapter 5. Table replication definition file

ObjectServer gateways can replicate the data in any table between ObjectServers. Details of the tables to be replicated are stored in the table replication definition file. The table replication definition file defines the tables in the source ObjectServer that the ObjectServer Gateway replicates in the target ObjectServer. The gateway reads this table only at startup. If you edit this table while the gateway is running, restart the gateway so that the changes take effect.

For unidirectional gateways, the **Gate.Reader.TblReplicateDefFile** property specifies the location of the table replication definition file.

For bidirectional gateways, the **Gate.ObjectServerA.TblReplicateDefFile** property and the **Gate.ObjectServerB.TblReplicateDefFile** property specify the locations of the files.

## Syntax

The syntax of the table replication definition file is as follows:

```
REPLICATE {ALL | INSERTS, UPDATES, DELETES,   FT_INSERTS, FT_UPDATES, FT_DELETES}
FROM TABLE sourcetable
 USING MAP mapname
[FILTER WITH filter_clause] [INTO destinationtable ] [ ORDER BY order_string]
[ASC | DESC] [WITH NORESYNC]
 [RESYNC DELETES FILTER condition]
[SET UPDTOINS CHECK TO {ENABLED|DISABLED|FORCED}]
[AFTER IDUC DO command]
[CACHE FILTER condition]
```

The following table describes the variable in the syntax.

| Variable | Description |
|---|---|
| *sourcetable* | The table to be replicated in the target ObjectServer. |
| *mapname* | The map definition that the defines the table. |
| *filter_clause* | The filter the that gateway uses to select rows for replicating. By default, filtering is inclusive, which means that the filter sends only those events that match the filter definition.<br><br>To send events that do not match the filter definition, precede the equals sign (=) with an exclamation mark (!). For example, the following filter clause sends all events whose severity is not set to 5: `FILTER with 'Severity !=5'`. |
| *destinationtable* | The table to receive the replicated table. If this clause is omitted, the name of the destination table is the same as the value of *sourcetable*. |

| Variable | Description |
|---|---|
| *order_string* | A comma-separated list of column names. Each column name can be followed by ASC or DESC, to indicate whether the values in the column are to be in ascending or descending order. |
| *condition* | The SQL condition that the gateway adds to the SELECT statement when limiting the cache entries that the gateway retrieves during a cache refresh. |
| *propertyname* | The property that the gateway uses to filter the table data. Only rows that satisfy the filter are replicated. |
| *propertyvalue* | The argument to be used in the filter. |
| *targettable* | The name of the table in which to replicate the data. |
| *delete_filter_clause* | The resynchronization delete filter that the gateway issues to the target ObjectServer. |

The following table describes the options for the REPLICATE command.

| Option | Description |
|---|---|
| `ALL` | The equivalent of `INSERTS, UPDATES, DELETES`. |
| `WITH NORESYNC` | Optional: Specifies the tables that you do not want to be resynchronized. |
| `ORDER BY` | Optional: Specifies the order in which the rows are returned to the gateway from the ObjectServer. For each column, you can specify how the rows are sorted.<br>• `ASC`: Specifies that the values in the column are sorted in ascending order.<br>• `DESC`: Specifies that the values in the column are sorted in descending order.<br>If you specify neither, `ASC` is used.<br>To define multiple columns for sorting, specify a comma-separated list; for example:<br>`ORDER BY 'Serial DESC, StateChange ACS'` |
| `RESYNC DELETE FILTER` | Optional: Defines a resynchronization deletion filter that specifies which rows to remove before insertion into the target table. This filter is used when the rows in the target and source tables are not an exact match. |

| Option | Description |
| --- | --- |
| SET UPDTOINS CHECK TO | Optional: allows you to configure the update-to-insert functionality.<br>• ENABLED: The gateway performs normal update-to-insert conversions. If an update from the source ObjectServer contains a table row that does not exist in the target ObjectServer, the update is converted to an insert, so that the table is repopulated in the target ObjectServer. This setting is the default.<br>• DISABLED: For each update received from the source ObjectServer, the gateway always sends an update to the destination ObjectServer. If the update contains rows that do not exist in the target, these rows are dropped.<br>• FORCED: The gateway converts all updates from the source ObjectServer to an insert on the target ObjectServer. If the row already exists in the target ObjectServer, the row is deduplicated. This behavior is identical to the behavior of probes. |
| AFTER IDUC DO | Optional: Specifies a column and associated value that the gateway applies to all rows that are inserted, updated, or deleted. |
| CACHE FILTER | Optional: Reduces the amount of data that the gateway retrieves during a unidirectional gateway cache refresh. The condition is added to the end of the select statement that it uses to retrieve cache entries.<br>If you want to use the CACHE FILTER option, it must be the last entry in the table replication definition. |

## Effects of delete forwarding on memory size

The memory usage of the gateway can be affected by whether you enable or disable delete forwarding.

The following table describes the effects on memory of enabling or disabling delete forwarding.

*Table 8. Effects of memory of delete forwarding*

| Status of delete forwarding | Sample command | Effect on memory |
| --- | --- | --- |
| Enabled | REPLICATE INSERTS, UPDATES | If the target ObjectServer is not fast enough to keep up with the flow of data that is sent to it, the memory usage of the gateway increases. |

*Table 8. Effects of memory of delete forwarding (continued)*

| Status of delete forwarding | Sample command | Effect on memory |
|---|---|---|
| Disabled | REPLICATE ALL | The gateway mapper drops the deletion details and does not pass them to the writer. To stop the entries for the deleted rows from remaining in the cache, the gateway mapper acts on behalf of the gateway writer by deleting the entries for deleted rows from the cache of the target ObjectServer. This behavior helps to keep memory usage stable. |

## Example table replication definition file

The table replication definition file defines how the ObjectServer gateway replicates tables between the source and target ObjectServers. Use this example to familiarize yourself with how the file works.

The following example shows a table replication definition file:

```
REPLICATE INSERT, DELETE FROM TABLE 'alerts.status'
 USING MAP 'StatusMap'
ORDER BY 'Serial ASC'
FILTER WITH 'Severity=!5'
SET UPDTOINS CHECK TO FORCED
AFTER IDUC DO 'Location=\'PASSED BY GW\''
CACHE FILTER  'ServerName IN (\'NCOMBS_P\',\'NCOMBS_B\')';

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

REPLICATE ALL FROM TABLE 'alerts.details'
 USING MAP 'DetailsMap';


##############################################################################
# NOTE: If replication of the user related system tables is required, uncomment
# the replication definitions below. The associated maps will also need to be
# uncommented.
##############################################################################

# REPLICATE ALL FROM TABLE 'security.users'
# USING MAP 'SecurityUsersMap'
# INTO 'transfer.users';
#
# REPLICATE ALL FROM TABLE 'security.groups'
# USING MAP 'SecurityGroupsMap'
# INTO 'transfer.groups';
#
# REPLICATE ALL FROM TABLE 'security.roles'
# USING MAP 'SecurityRolesMap'
# INTO 'transfer.roles';
#
# REPLICATE ALL FROM TABLE 'security.role_grants'
# USING MAP 'SecurityRoleGrantsMap'
# INTO 'transfer.role_grants';
```

```
#
# REPLICATE ALL FROM TABLE 'security.group_members'
# USING MAP 'SecurityGroupMembersMap'
# INTO 'transfer.group_members';
```

# Notices

This information was developed for products and services offered in the U.S.A.

IBM® may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
958/NH04
IBM Centre, St Leonards
601 Pacific Hwy
St Leonards, NSW, 2069
Australia

IBM Corporation
896471/H128B
76 Upper Ground
London SE1 9PZ
United Kingdom

IBM Corporation
JBF1/SOM1
294 Route 100
Somers, NY, 10589-0100
United States of America

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Portions of this product include software developed by Daniel Veillard.

- `libxml2-2.7.8`

The `libxml2-2.7.8` software is distributed according to the following license agreement:
© Copyright 1998-2003 Daniel Veillard.
All Rights Reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE DANIEL VEILLARD BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Except as contained in this notice, the name of Daniel Veillard shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from him.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Trademarks

AIX®, IBM, the IBM logo, ibm.com®, Netcool®, Netcool/OMNIbus, and Tivoli® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.


Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

# Index

Printed in USA

SC14-7531-00