MICROMUSE™
NETCOOL® SOLUTIONS

Micromuse

Netcool®/OMNIbus™

Supporting Products

ObjectServer Gateway

Netcool®/OMNIbus™
Document Version 2.1
Supporting Products
ObjectServer Gateway
13 October 2004
Status: Approved

Micromuse Disclaimer of Warranty and Statement of Limited Liability

Micromuse provides this document "as is", without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose or non-infringement. This document may contain technical inaccuracies or typographical errors. Micromuse may make improvements and changes to the programs described in this document or this document at any time without notice. Micromuse assumes no responsibility for the use of the programs or this document except as expressly set forth in the applicable Micromuse agreement(s) and subject to terms and conditions set forth therein. Micromuse does not warrant that the functions contained in the programs will meet your requirements, or that the operation of the programs will be uninterrupted or error-free. Micromuse shall not be liable for any indirect, consequential or incidental damages arising out of the use or the ability to use the programs or this document.

Micromuse specifically disclaims any express or implied warranty of fitness for high risk activities.

Micromuse programs and this document are not certified for fault tolerance, and are not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems ("High Risk Activities") in which the failure of programs could lead directly to death, personal injury, or severe physical or environmental damage.

Compliance with Applicable Laws; Export Control Laws

Use of Micromuse programs and documents is governed by all applicable federal, state and local laws. All information therein is subject to U.S. export control laws and may also be subject to the laws of the country where you reside.

All Micromuse programs and documents are commercial in nature. Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7015 and FAR 52.227-19.

Trademarks and Acknowledgements

Micromuse and Netcool are registered trademarks of Micromuse.

Other Micromuse trademarks include but are not limited to: Netcool/OMNIbus, Netcool/OMNIbus for Voice Networks, Netcool/Reporter, Netcool/Internet Service Monitors, Netcool/NT Service Monitors, Netcool/Wireless Service Monitors, Netcool/Usage Service Monitors, Netcool/Fusion, Netcool/Data Center Monitors, Netcool/Impact, Netcool/Visionary, Netcool/Precision for IP Networks, Netcool/Precision for Transmission Networks, Netcool/Firewall, Netcool/Webtop, Netcool/SM Operations, Netcool/SM Configuration, Netcool/OpCenter, Netcool/System Service Monitors, Netcool/Application Service Monitors, Netcool for Asset Management, Netcool for Voice over IP, Netcool for Security Management, Netcool/Portal 2.0 Premium Edition, Netcool ObjectServer, Netcool/Software Developers Kit, Micromuse Alliance Program and Network Slice.

Micromuse acknowledges the use of I/O Concepts Inc. X-Direct 3270 terminal emulators and hardware components and documentation in Netcool/Fusion. X-Direct ©1989-1999 I/O Concepts Inc. X-Direct and Win-Direct are trademarks of I/O Concepts Inc.

Netcool/Fusion contains IBM Runtime Environment for AIX®, Java™ Technology Edition Runtime Modules © Copyright IBM Corporation 1999. All rights reserved.

Micromuse acknowledges the use of MySQL in Netcool/Precision for IP Networks. Copyright © 1995, 1996 TcX AB & Monty Program KB & Detron HB Stockholm SWEDEN, Helsingfors FINLAND and Uppsala SWEDEN. All rights reserved.

Micromuse acknowledges the use of the UCD SNMP Library Netcool/ISMs. Copyright © 1989, 1991, 1992 by Carnegie Mellon University. Derivative Work - Copyright © 1996, 1998, 1999, 2000 The Regents of the University of California. All rights reserved.

Portions of the Netcool/ISMs code are copyright (c) 2001, Cambridge Broadband Ltd. All rights reserved.

Portions of the Netcool/ISMs code are copyright (c) 2001, Networks Associates Technology, Inc. All rights reserved.

Micromuse acknowledges the use of Viador Inc. software and documentation for Netcool/Reporter. Viador © 1997-1999 is a trademark of Viador Inc.

Micromuse acknowledges the use of software developed by the Apache Group for use in the Apache HTTP server project. Copyright © 1995-1999 The Apache Group. Apache Server is a trademark of the Apache Software Foundation. All rights reserved.

Micromuse acknowledges the use of software developed by Edge Technologies, Inc. ©2003 Edge Technologies, Inc. and Edge enPortal are trademarks or registered trademarks of Edge Technologies Inc. All rights reserved.

Micromuse acknowledges the use of Acme Labs software in Netcool/SM Operations, Netcool/SM Configuration and Netcool/OpCenter. Copyright 1996, 1998 Jef Poskanzer jef@acme.com. All rights reserved.

Micromuse acknowledges the use of WAP and MMS stacks in Netcool/SM as powered by http://www.serialio.com.

Micromuse acknowledges the use of Merant drivers. Copyright © MERANT Solutions Inc., 1991-1998.

The following product names are trademarks of Tivoli Systems or IBM Corporation: AIX, IBM, OS/2, RISC System/6000, Tivoli Management Environment, and TME10.

IBM, NetView/6000 and WebSphere are either trademarks or registered trademarks of IBM Corporation. VTAM is a trademark of IBM Corporation.

Omegamon is a trademark of Candle Corporation.

Netspy is a trademark of Computer Associates International Inc.

The Sun logo, Sun Microsystems, SunOS, Solaris, SunNet Manager, Java are trademarks of Sun Microsystems Inc.

SPARC is a registered trademark of SPARC International Inc. Programs bearing the SPARC trademark are based on an architecture developed by Sun Microsystems Inc. SPARCstation is a trademark of SPARC International Inc., licensed exclusively to Sun Microsystems Inc.

UNIX is a registered trademark of the X/Open Company Ltd.

Sybase is a registered trademark of Sybase Inc.

Action Request System and Remedy are registered trademarks of Remedy Corporation.

Peregrine System and ServiceCenter are registered trademarks of Peregrine Systems Inc.

HP, HP-UX and OpenView are trademarks of Hewlett-Packard Company.

Netcool®/OMNIbus™
Document Version 2.1
Supporting Products
ObjectServer Gateway
13 October 2004
Status: Approved

Micromuse Inc., 139 Townsend Street, San Francisco, USA CA 94107

www.micromuse.com

Document Control Page

This document is supplied solely as an electronic copy. To obtain the most recent version, see the Micromuse support site.

```
http://support.micromuse.com/documentation/supplementals/index.html
```

Before you read this document you should be familiar with the Netcool/OMNIbus product documentation.

Document Details

Document Name	ObjectServer Gateway
Print Date	13 October 2004
Current Status	Approved

Modification History

Document Version No.	Comments	Print Date
1.0	Updated for Netcool/OMNIbus 3.4.1	10 December 2001
1.1	Updated	6 August 2002
1.2	Updated	2 December 2002
1.3	Updated	24 February 2003
1.4	Updated	29 July 2003
2.0	Gateway rewritten using the Netcool Gateway Toolkit	6 April 2004
2.1	Gateway name recognition functionality added. Update-to-insert and order by functionality added to the table replication definition language. Errors in examples corrected.	13 October 2004

Table of Contents

ObjectServer Gateway

ObjectServer gateways are used to replicate table data (for example, alert-related data) between different Netcool/OMNIbus ObjectServers. ObjectServer gateways consist of *readers* and *writers*. Readers extract alerts from a source ObjectServer. Writers send the alert data to a target ObjectServer.

ObjectServer gateways can be used to:

- Maintain a backup ObjectServer.
- Replicate alerts between different Network Operations Centers (NOCs).
- Create a tiered architecture.

Installation

For information about installing gateways, refer to the Netcool/OMNIbus Installation Guide.

About the ObjectServer Gateway

Unlike the ObjectServer Gateway for Netcool/OMNIbus v3.x, the ObjectServer Gateway for Netcool/OMNIbus v7 is not a writer within the gateway server binary (`nco_gate`). Instead, there are two binaries: one for unidirectional gateways (`nco_g_objserv_uni`) and one for bidirectional gateways (`nco_g_objserv_bi`). Both binaries use the Netcool Gateways Toolkit (NGTK) library, which provides the basic framework for the gateway process and are configured independently of each other. The NGTK library is installed as a part of Netcool/OMNIbus v7.

Note: Separate licenses are required for the unidirectional and bidirectional ObjectServer gateways. You also require a resynchronizing license if you want to use the resynchronization feature of the gateway. For details about licensing for the ObjectServer Gateway, contact the Micromuse License Generation team.

Unidirectional Gateways

The unidirectional ObjectServer Gateway enables alerts to flow in one direction, from a source ObjectServer to a destination ObjectServer. 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.

Figure 1 shows the configuration of a unidirectional ObjectServer gateway.

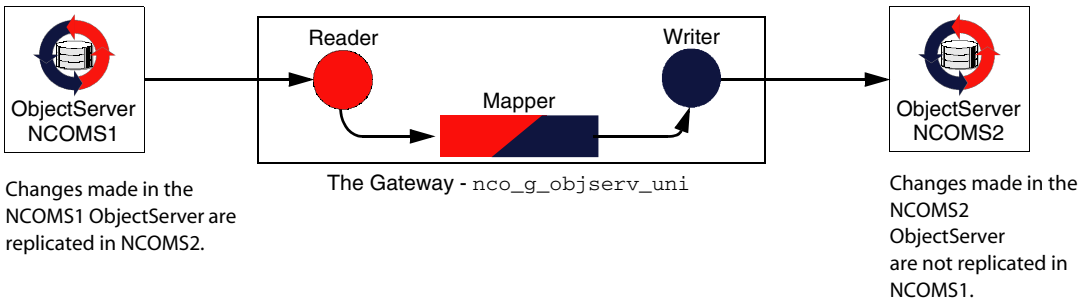


Figure 1: Unidirectional ObjectServer Gateway

Information Flow

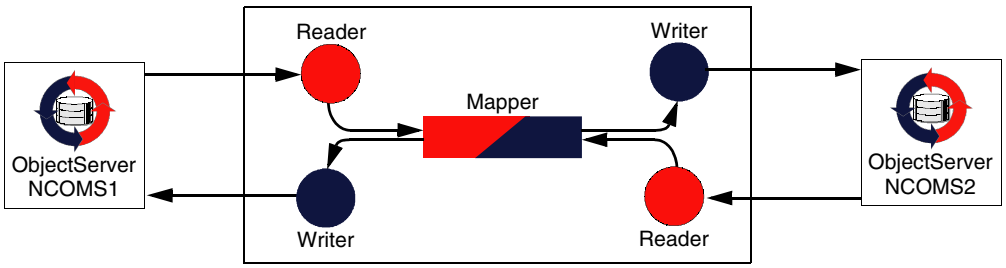
The unidirectional gateway is comprised of mapper, reader, and writer components. The flow of information between the components is as follows:

1. The reader component reads from the source ObjectServer and passes the data to the mapper component.
2. The mapper component receives data from the reader component, transforms the data into an appropriate form for the target writer using the map definition file, and passes the data to the writer component.
3. The writer component receives the source data from the mapper component and writes it to the destination ObjectServer.

Bidirectional Gateways

The bidirectional ObjectServer Gateway enables alerts to flow in both directions between a source and a destination ObjectServer. Any changes made in the source ObjectServer are replicated in the destination ObjectServer, and changes in the destination ObjectServer are replicated in the source ObjectServer. This ensures that both ObjectServers contain the same alerts and allows you to maintain a backup ObjectServer.

Figure 2 shows the configuration of a bidirectional ObjectServer gateway:



Changes made in the NCOMS1 ObjectServer are replicated in NCOMS2.

The Gateway - `nco_g_objserv_bi`

Changes made in the NCOMS2 ObjectServer are replicated in NCOMS1.

Figure 2: Bidirectional ObjectServer Gateway

Information Flow

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

1. A source reader/writer component reads from the source ObjectServer and passes the data to the mapper component.
2. The mapper component receives data from the source reader/writer component, transforms the data into an appropriate form using the map definition file, and passes the data to the target reader/writer component.
3. A second reader/writer component receives the source data from the mapper component and writes it to the destination ObjectServer.

Note: Bidirectional gateways can be used to create a failover pair of ObjectServers.

Features

This section describes the features of the ObjectServer Gateway.

Passing Table Data

The gateway can replicate the data in any table between ObjectServers. Details of the tables to be replicated are stored in the table definition file; for details of the format of this file, see *Table Replication Definition File* on page 43.

Centralized Property Management

The gateway uses centralized property management and separates properties from data processing configuration. Configuration of the unidirectional and bidirectional gateways is performed using configuration files; for details, see *Configuring the ObjectServer Gateway* on page 6.

Supports Backward Compatibility with Netcool/OMNIbus v3.x

To support backward compatibility with Netcool/OMNIbus v3.x, the ObjectServer Gateway for Netcool/OMNIbus v7 provides the following functionality:

- The v7 ObjectServer supports the same relationship between the `alerts.status`, `alerts.journal`, and `alerts.details` tables as v3.x.
- If a reader connects to a v7 ObjectServer, it uses IDUC version 2 to monitor all tables, including the `alerts.status`, `alerts.journal` and `alerts.details` tables.
- If a reader connects to a v3.x ObjectServer, it uses IDUC version 1 to monitor the `alerts.status`, `alerts.journal` and `alerts.details` tables.

Configuring the ObjectServer Gateway

The ObjectServer Gateway uses a centralized property management library; this separates properties from data processing configuration. The following sections describe the configuration files for both the unidirectional and bidirectional gateways:

- *Properties File for the Unidirectional Gateway* on page 7
- *Properties File for the Bidirectional Gateway* on page 18
- *Map Definition File* on page 34
- *Startup Command File* on page 39
- *Table Replication Definition File* on page 43

Note: All users, groups, roles, and restriction filters must be identical in the master ObjectServer and slave ObjectServer.

Properties File for the Unidirectional Gateway

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

The properties files for the unidirectional and bidirectional gateways contain similar properties; however, they are described separately for clarity. For details of the properties file for bidirectional gateways, see *Bidirectional Gateway Properties and Command Line Options* on page 19.

The default location for the property file for the unidirectional and bidirectional gateways is:

```
$OMNIHOME/etc/server_name.props
```

Note: The default properties file must be copied to the `$OMNIHOME/etc` folder.

Starting the Gateway

To start the unidirectional gateway, enter the following command on the command line:

```
nco_g_objserv_uni -name
```

Unidirectional Gateway Properties and Command Line Options

The unidirectional gateway properties file is divided into the following sets of properties:

- Common gateway properties
- Gateway mapper properties
- Gateway reader properties
- Gateway writer properties
- Resynchronization properties

Table 1 describes the properties specific to the unidirectional ObjectServer Gateway. For information about the properties specific to the bidirectional ObjectServer Gateway, see *Bidirectional Gateway Properties and Command Line Options* on page 19. For information about the common properties and Interprocess Communication (IPC) properties, see the Netcool/OMNIbus Probe and Gateway Guide.

Table 1: Properties Used by the Unidirectional ObjectServer Gateway

Property Name	Command Line Options	Description
Common Gateway Properties		
`Gate.CacheHashTblSize` *integer*	`-chashtblsize` *integer*	Size (in elements) that the gateway allocates for the hash table cache. The default is `5023`.
`Gate.MapFile` *string*	`-mapfile` *string*	Location of the map definition file. For details of the format of this file, see *Map Definition File* on page 34. The default is `$OMNIHOME/gates/objserv_uni/objserv_uni.map`.
`Gate.StartupCmdFile` *string*	`-startupcmdfile` *string*	Location of the startup command file. For details of the format of the this file, see *Startup Command File* on page 39. The default is `$OMNIHOME/objserv_uni/objserv_uni.startup.cmd`.
`Gate.Transfer.Failover SyncRate` *integer*	`-fsyncrate` *integer*	Rate (in seconds) of the failover synchronization. The default is `60`.

Table 1: Properties Used by the Unidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.NGtkDebug` *boolean*	`-ngtkdebug` *boolean*	Specifies whether the NGTK library should log debug messages. The default is `TRUE`. **Note:** You can specify which debug messages are included in the debug log file using the `Gate.Mapper.Debug`, `Gate.Reader.Debug`, and `Gate.Writer.Debug` properties.
`Gate.PAAware` *integer*	`-paaware` *integer*	Indicates whether the gateway is Process Agent (PA) aware. The default is `0` (not PA aware). **Note:** This property is maintained by the PA server and is included in the properties file for information only.
`Gate.PAAwareName` *string*	`-paname` *string*	Indicates the name of the Process Agent controlling the gateway. The default is "". **Note:** This property is maintained by the PA server and is included in the properties file for information only.
Gateway Mapper Properties		
`Gate.Mapper.Debug` *boolean*	`-mapperdebug` *boolean*	Specifies whether the gateway includes mapper debug messages in the debug log. The default is `TRUE`.
`Gate.Mapper.Forward HistoricDetails` *boolean*	`-mapperforhistdtls` *boolean*	Specifies whether the gateway forwards all historic details on converted update. The default is `FALSE`.
`Gate.Mapper.Forward HistoricJournals` *boolean*	`-mapperforhistjrnl` *boolean*	Specifies whether the gateway forwards all historic journals on converted update. The default is `FALSE`.
Gateway Reader Properties		
`Gate.Reader.Debug` *boolean*	`-readerdebug` *boolean*	Specifies whether the gateway includes gateway reader debug messages in the debug log. The default is `TRUE`.

Table 1: Properties Used by the Unidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.Reader.Description` *string*	`-readerdescription` *string*	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. The default is "".
`Gate.Reader.DetailsTableName` *string*	`-readerdetailstblname` *string*	Name of the details table that the gateway reads. The default is `alerts.details`.
`Gate.Reader.FailbackEnabled` *boolean*	`-readerfailbackenabled` *boolean*	Enables failback for this ObjectServer. For details about failback functionality, see *Failback* on page 31. The default is `TRUE`.
`Gate.Reader.FailbackTimeout` *integer*	`-readerfailbacktimeout` *integer*	Time (in seconds) that the gateway allows before entering failback mode. The default is `30`.
`Gate.Reader.IDUCFlushRate` *integer*	`-readeriducflushrate` *integer*	Rate (in seconds) of the granularity of the reader. If you set this property to `0`, the reader gets its updates at the same granular rate as that of the ObjectServer to which it is connected. The default is `0`. **Warning:** If you set this property to a value greater than `0`, the reader issues automatic IDUC flush requests to the ObjectServer with this frequency. This enables the reader to run at a faster granularity than that of the ObjectServer, thus enabling the gateway to capture more detailed event changes in systems where the ObjectServer itself has high granularity settings.
`Gate.Reader.JournalTableName` *string*	`-readerjournaltblname` *string*	Name of the journal table that the gateway reads. The default is `alerts.journal`.
`Gate.Reader.LogOSSql` *boolean*	`-readerlogossql` *boolean*	Specifies whether the gateway logs all SQL commands sent to the ObjectServer in debug mode. The default is `FALSE`.

Table 1: Properties Used by the Unidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.Reader.Password` *string*	`-readerpassword` *string*	Password associated with the user specified by the `Gate.Reader.Username` property. This password *must* be encrypted by the `nco_g_crypt` utility. For details about this utility, see the Netcool/OMNIbus Administration Guide. Plain text passwords are not accepted as the gateway assumes that all passwords are encrypted by this tool for security. The default is "".
`Gate.Reader.Reconnect Timeout` *integer*	`-readerreconntimeout` *integer*	Time (in seconds) between each reconnection poll attempt that the gateway makes if the connection to the ObjectServer is lost. The default is `30`.
`Gate.Reader.Server` *string*	`-readerserver` *string*	Name of the ObjectServer from which the gateway reads alerts. The default is `NCOMS`.
`Gate.Reader.Status TableName` *string*	`-readerstatustblname` *string*	Name of the status table that the gateway reads. The default is `alerts.status`.
`Gate.Reader.Tbl ReplicateDefFile` *string*	`-readertblrepdeffile` *string*	Path to the table replication definition file. The default is `$OMNIHOME/gates/objserv_uni/objserv_uni.reader.tblrep.def`. **Note:** For details about this file, see *Table Replication Definition File* on page 43.
`Gate.Reader.Username` *string*	`-readerusername` *string*	Username used to authenticate the ObjectServer connection. The default is `root`.
Gateway Writer Properties		
`Gate.Writer.Buffersize` *integer*	`-writerbufsize` *integer*	Number of entries that the gateway stores in the buffer before flushing, if buffering is enabled. This property can be used to fine-tune the efficiency of the gateway. For details, see *Buffer Size* on page 15. The default is `25`. **Note:** The gateway flushes the buffer when the end of a batch of SQL statements has been reached regardless of the buffer size.

Table 1: Properties Used by the Unidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.Writer.Debug` *boolean*	`-writerdebug` *boolean*	Specifies whether the gateway includes gateway writer debug messages in the debug log. The default is `TRUE`.
`Gate.Writer.` `Description` *string*	`-writerdescription` *string*	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. The default is "".
`Gate.Writer.Failback` `Enabled` *boolean*	`-writerfailbackenabled` *boolean*	Enables failback for this ObjectServer. For details about failback functionality, see *Failback* on page 31. The default is `TRUE`.
`Gate.Writer.Failback` `Timeout` *integer*	`-writerfailbacktimeout` *integer*	Time (in seconds) that the gateway allows before checking for the return of the master ObjectServer and failing back. The default is `30`.
`Gate.Writer.LogOSSql` *boolean*	`-writerlogossql` *boolean*	Specifies whether the gateway logs all SQL commands sent to the ObjectServer in debug mode. The default is `FALSE`.
`Gate.Writer.Password` *string*	`-writerpassword` *string*	Password associated with the user specified by the `Gate.Writer.Username` property. This password *must* be encrypted by the `nco_g_crypt` utility. For details about this utility, see the Netcool/OMNIbus Administration Guide. Plain text passwords are not accepted as the gateway assumes that all passwords are encrypted by this tool for security. The default is "".
`Gate.Writer.Reconnect` `Timeout` *integer*	`-writerreconntimeout` *integer*	If the gateway loses the connection to the ObjectServer, this property defines the time (in seconds) between each reconnection poll attempt. The default is `30`.

Table 1: Properties Used by the Unidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.Writer.Refresh CacheOnUpdate` *boolean*	`-writerrefcacheonupd` *boolean*	This property controls the refreshing of the hash table cache for this ObjectServer: `TRUE` - Cache is resynchronized with the target ObjectServer prior to processing each collection of row updates for a table in the current `IDUC` window. `FALSE` - Gateway assumes that its cache is accurate and does not resynchronize it. The default is `TRUE`.
`Gate.Writer.SAF` *boolean*	`-writersaf` *boolean*	Instructs the gateway to store all table entries if the destination ObjectServer is unavailable and to forward them when the ObjectServer becomes available again. The default is `FALSE`.
`Gate.Writer.SAFFile` *string*	`-writersaffile` *string*	Name of the file that the gateway uses to store table entries while the destination ObjectServer is unavailable. 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*	Name of the ObjectServer to which the gateway writes alerts. The default is `REMOTE`.
`Gate.Writer.Username` *string*	`-writerusername` *string*	Username used to authenticate the ObjectServer connection. This username is used to establish both the writer's IDUC connection and the subsidiary SQL command connection. The default is `root`.

Table 1: Properties Used by the Unidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
Resynchronization Properties		
`Gate.Resync.Enable` *boolean*	`-resyncenable` *boolean*	Allows you to specify that the gateway uses resynchronization. The default is `TRUE`.
`Gate.Resync.Type` *string*	`-resynctype` *string*	Type of resynchronization that is required: `NORMAL` - Gateway checks the contents of the two ObjectServers specified by the `Gate.Reader.Server` property and `Gate.Writer.Server` property, and, if necessary, resynchronizes them. `UDPATE` - Gateway performs the resynchronization with updates. This option ensures that updates made in the upper tier are not lost during the resynchronization. The default is `NORMAL`.

Hash Table Cache

The gateway uses a hash table cache to store details of tables that require transferring from one ObjectServer to another. 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. It is also used for any other tables specified using the table replication definition table.

The cache aids performance optimization by providing the gateway with a summarized, in-memory view of the contents of the ObjectServers to which it is linked. This means that the gateway does not have to query an ObjectServer to check for the existence of an event, or the `Serial` value or `Tally` value of an event; it can check the cache of the target ObjectServer instead.

You can control the size of the hash table cache by using the `Gate.CacheHashTblSize` property. By default, the size of the hash table cache is `5023` elements (or rows). This can be increased if the status table has a large number of rows (for example, in excess of 20,000).

Note: To maximize efficiency, you should specify a prime number for the `Gate.CacheHashTblSize` property.

Error Handling

The gateway provides configurable error handling. Error handling is provided by the Netcool/OMNIbus Gateway Toolkit (NGTK) library. To specify that the NGTK library logs debug messages, set the `Gate.NGtkDebug` property to `TRUE`.

You can specify which debug messages are included in the debug files by using the `Gate.Mapper.Debug`, `Gate.Reader.Debug`, and `Gate.Writer.Debug` properties; these can be set to `TRUE` or `FALSE` as appropriate.

Process Agent Control

The gateway can be run under Process Agent (PA) control. The `Gate.PAAware` property indicates whether the gateway is PA aware. The `Gate.PAAwareName` property indicates which PA is running the gateway.

These properties are maintained automatically by the PA server and provide information only. They should not be changed manually.

Store and Forward

The gateway supports store and forward on any table when the destination ObjectServer goes offline. This feature can be configured to store and forward either all or none of the tables being replicated. To activate the store and forward function, set the `Gate.Writer.SAF` property to `TRUE`. Then, specify the file to which the alerts are written while the destination ObjectServer is offline using the `Gate.Writer.SAFFile` property. To deactivate the store and forward function, set the `Gate.Writer.SAF` property to `FALSE`.

Buffer Size

The buffer size controls the number of entries that the gateway stores in its buffer before flushing them to the ObjectServer. This can be set using the `Gate.Writer.Buffersize` property. This property can adjusted to fine-tune the efficiency of the gateway.

The optimum value for the buffer size depends upon the average event size and the speed of the network. The default value has proved to be efficient for many installations. To determine the most efficient setting for your system, compare the timing figures for resynchronization operations performed using different settings for this property.

Resynchronization

The gateway supports two resynchronization modes:

- `NORMAL` - Gateway deletes matching data from the destination table and inserts data from the source table.

- `UPDATE` - Gateway inserts rows that are not in the destination table and updates rows in the destination table with the corresponding values in the source table. When events already exist in the destination table, the gateway can be configured to use some of the destination column values in preference to data from the source table. This filtered column data can then be written back to the source table so that both sides are consistent.

To specify that the gateway uses resynchronization, set the `Gate.Resync.Enable` property to `TRUE`. To specify which type of resynchronization the gateway uses, set the `Gate.Resync.Type` property.

Failback

Failback functionality comes into operation when a gateway loses its connection to the primary ObjectServer; this enables the gateway to connect to a backup ObjectServer. Failback functionality also enables the gateway to reconnect to the primary ObjectServer when it becomes active again. Figure 3 shows the failover-failback relationship between the primary and secondary ObejctServers to which an ObjectServer gateway is connected.

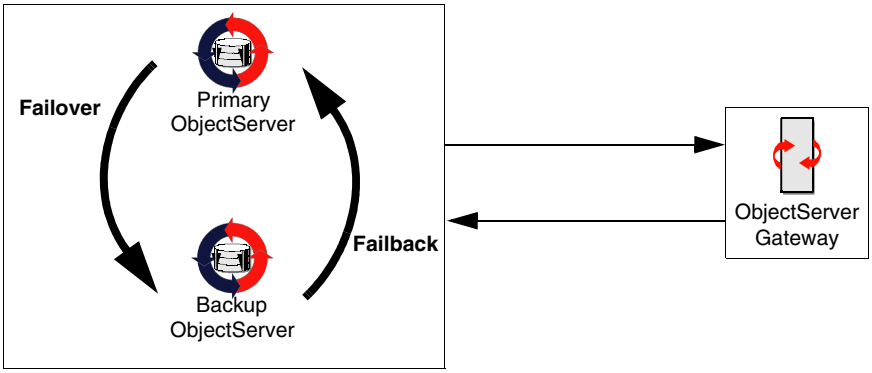


Figure 3: Failback Functionality

Setting Up Failback

To set up failback, set the ObjectServer's `BackupObjectServer` property for the backup ObjectServer to `TRUE`. For details about setting ObjectServer properties, see the Netcool/OMNIbus Administration Guide.

To enable failback, in the gateway properties file, set the `Gate.Reader.Failback` and `Gate.Writer.Failback` properties to `TRUE`. When the primary ObjectServer fails, the reader and writer fail over to the backup ObjectServer without shutting down. When the reader or writer have detected that they are now connected to a backup ObjectServer, they periodically poll for the return of the primary ObjectServer. When the primary ObjectServer has been detected again, the reader or writer automatically fail back to the primary ObjectServer.

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 `Gate.Writer.FailbackTimeout` properties.

Properties File for the Bidirectional Gateway

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

The properties files for the unidirectional and bidirectional gateways contain similar properties; however, they are described separately for clarity. For details of the properties file for unidirectional gateways, see *Unidirectional Gateway Properties and Command Line Options* on page 8.

The default location for the property file for the unidirectional and bidirectional gateways is:

```
$OMNIHOME/etc/server_name.props
```

Note: The default properties file must be copied to the `$OMNIHOME/etc` folder.

Starting the Gateway

To start the bidirectional gateway, enter the following command on the command line:

```
nco_g_objserv_bi -name
```

Bidirectional Gateway Properties and Command Line Options

The bidirectional gateway properties file is divided into the following sets of properties:

- Common gateway properties
- Gateway mapper properties
- ObjectServer properties
- Resynchronization properties

Table 2 describes the properties specific to the bidirectional ObjectServer Gateway. For information about the properties specific to the unidirectional ObjectServer Gateway, see *Unidirectional Gateway Properties and Command Line Options* on page 8. For information about the common properties and Interprocess Control (IPC), see the Netcool/OMNIbus Administration Guide.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway

Property Name	Command Line Options	Description
Common Gateway Properties		
`Gate.CacheHashTblSize` *integer*	`-chashtblsize` *integer*	Size (in elements) that the gateway allocates for the hash table cache. The default is `5023`.
`Gate.MapFile` *string*	`-mapfile` *string*	Location of the map definition file. For details of the format of the this file, see *Map Definition File* on page 34. The default is `$OMNIHOME/gates/objserv_bi/objserv_bi.map`.
`Gate.StartupCmdFile` *string*	`-startupcmdfile` *string*	Location of the startup command file. For details of the format of this file, see *Startup Command File* on page 39. The default is `$OMNIHOME/objserv_bi/objserv_bi.startup.cmd`.
`Gate.Transfer.FailoverSyncRate` *integer*	`-fsyncrate` *integer*	Rate (in seconds) of the failover synchronization. The default is `60`.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.NGtkDebug` *boolean*	`-ngtkdebug` *boolean*	Specifies whether the NGTK library should log debug messages. The default is `TRUE`. **Note:** You can specify which debug messages are included in the debug log file using the `Gate.Mapper.Debug`, `Gate.Reader.Debug`, and `Gate.Writer.Debug` properties.
`Gate.PAAware` *integer*	`-paaware` *integer*	Specifies whether the gateway is Process Agent (PA) aware. The default is `0`. **Note:** This property is maintained by the PA server and is for information only.
`Gate.PAAwareName` *string*	`-paname` *string*	Specifies the name of the Process Agent controlling the gateway. The default is "". **Note:** This property is maintained by the PA server and is for information only.
Gateway Mapper Properties		
`Gate.Mapper.Debug` *boolean*	`-mapperdebug` *boolean*	Specifies whether the gateway includes mapper debug messages in the debug log. The default is `TRUE`.
`Gate.Mapper.Forward HistoricDetails` *boolean*	`-mapperforhistdtls` *boolean*	Specifies whether the gateway forwards all historic details on converted update. The default is `FALSE`.
`Gate.Mapper.Forward HistoricJournals` *boolean*	`-mapperforhistjrnl` *boolean*	Specifies whether the gateway forwards all historic journals on converted update. The default is `FALSE`.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
ObjectServer Properties		
`Gate.ObjectServerA.Buffersize` *integer*	`-objectserverabufsize` *integer*	Number of entries that the gateway stores in the buffer for this ObjectServer before flushing, if buffering is enabled. This property can be used to fine-tune the efficiency of the gateway. For details, see *Buffer Size* on page 33. The default is `25`. **Note:** The gateway flushes the buffer when the end of a batch of SQL statements has been reached regardless of the buffer size.
`Gate.ObjectServerA.Debug` *boolean*	`-objectserveradebug` *boolean*	Specifies whether the gateway includes debug messages for this ObjectServer in the gateway debug log. The default is `TRUE`.
`Gate.ObjectServerA.DeleteIfNoDedup` *boolean*	`-objectserveradelif nodedup` *boolean*	If the gateway forwards deletes, this attribute controls the deletion: `FALSE` - Delete is always applied. `TRUE` - Prevents the deletion from being applied if the event in the target server indicates that the event has occurred again since the delete was issued. The default is `FALSE`. **Note:** In most cases, this property is set to `FALSE`. For details of using an alternative deletion strategy, see *Alternative Deletion Strategy* on page 33.
`Gate.ObjectServerA.Description` *string*	`-objectservera description` *string*	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. The default is "".
`Gate.ObjectServerA.FailbackEnabled` *boolean*	`-objectservera failbackenabled` *boolean*	Enables failback for this ObjectServer. For details about failback functionality, see *Failback* on page 31. The default is `FALSE`.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.ObjectServerA.FailbackTimeout` *integer*	`-objectservera failbacktimeout` *integer*	Time (in seconds) that the gateway allows before checking for the return of the master ObjectServer and failing back. The default is `30`.
`Gate.ObjectServerA.LogOSSql` *boolean*	`-objectservera logossql` *boolean*	Specifies whether the gateway logs all SQL commands sent to this ObjectServer in debug mode. The default is `FALSE`.
`Gate.ObjectServerA.Password` *string*	`-objectservera password` *string*	Password associated with the user specified by the `Gate.ObjectServerA.Username` property. This password *must* be encrypted by the `nco_g_crypt` utility. For details about this utility, see the Netcool/OMNIbus Administration Guide. Plain text passwords are not accepted as the gateway assumes that all passwords are encrypted by this tool for security. The default is "".
`Gate.ObjectServerA.ReconnectTimeout` *integer*	`-objectserverare conntimeout` *integer*	Time (in seconds) between each reconnection poll attempt if the connection to this ObjectServer is lost. The default is `30`.
`Gate.ObjectServerA.RefreshCacheOnUpdate` *boolean*	`-objectserveraref cacheonupd` *boolean*	Controls the refreshing of the hash table cache for this ObjectServer: `TRUE` - Cache is resynchronized with the target ObjectServer prior to processing each collection of row updates for a table in the current `IDUC` window. `FALSE` - Gateway assumes that its cache is accurate and does not resynchronize it. The default is `FALSE`.
`Gate.ObjectServerA.SAF` *boolean*	`-objectserverasaf` *boolean*	Instructs the gateway to store all table entries if the destination ObjectServer is unavailable and to forward them when the ObjectServer becomes available again. The default is `FALSE`.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.ObjectServerA.SAFFile` *string*	`-objectserverasaffile` *string*	Name of the file that the gateway uses to store table entries while the destination ObjectServer is unavailable. The default is `$OMNIHOME/var/objserv_bi/NCO_GATE_ObjectServerA.store`. **Note:** This file is only used if the `Gate.ObjectServerA.SAF` property is set to `TRUE`.
`Gate.ObjectServerA.Server` *string*	`-objectserveraserver` *string*	Name of this ObjectServer. The default is `NCOMS`.
`Gate.ObjectServerA.Username` *string*	`-objectservera username` *string*	Username used to authenticate connection to this ObjectServer. This username is used to establish both the writer's IDUC connection and the subsidiary SQL command connection. The default is `root`.
`Gate.ObjectServerA.TblReplicateDefFile` *string*	`-objectserveratblrep deffile` *string*	Path to the table replication definition file. The default is `$OMNIHOME/gates/objserv_bi/objserv_bi.objectservera.tblrep.def`. **Note:** For details about this file, see *Table Replication Definition File* on page 43.
`Gate.ObjectServerA.StatusTableName` *string*	`-objectservera statustblname` *string*	Name of the status table that the gateway reads. The default is `alerts.status`.
`Gate.ObjectServera.JournalTableName` *string*	`-objectservera journaltblname` *string*	Name of the journal table that the gateway reads. The default is `alerts.journal`.
`Gate.ObjectServerA.DetailsTableName` *string*	`-objectservera detailstblname` *string*	Name of the details table that the gateway reads. The default is `alerts.details`.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.ObjectServerA.IDUCFlushRate` *integer*	`-objectservera iducflushrate` *integer*	Rate (in seconds) of the granularity of the reader. If you set this property to `0`, the reader gets its updates at the same granular rate as that of the ObjectServer to which it is connected. The default is `0`. **Warning:** If you set this property to a value greater than `0`, the reader issues automatic IDUC flush requests to the ObjectServer with this frequency. This enables the reader to run at a faster granularity than that of the ObjectServer, thus enabling the gateway to capture more detailed event changes in systems where the ObjectServer itself has high granularity settings.
`Gate.ObjectServerB.Buffersize` *integer*	`-objectserverbbufsize` *integer*	Number of entries that the gateway stores in the buffer for this ObjectServer before flushing, if buffering is enabled. This property can be used to fine-tune the efficiency of the gateway. For details, see *Buffer Size* on page 33. The default is `25`. **Note:** The gateway flushes the buffer when the end of a batch of SQL statements has been reached regardless of the buffer size.
`Gate.ObjectServerB.Debug` *boolean*	`-objectserverbdebug` *boolean*	Specifies whether the gateway includes debug messages for this ObjectServer in the debug log. The default is `TRUE`.
`Gate.ObjectServerB.DeleteIfNoDedup` *boolean*	`-objectserverbdelif nodedup` *boolean*	If the gateway forwards deletes, this attribute controls the deletion: `FALSE` - Delete is always applied. `TRUE` - Prevents the deletion from being applied if the event in the target server indicates that the event has occurred again since the delete was issued. The default is `FALSE`. **Note:** In most cases, this property is set to `FALSE`. For details of using an alternative deletion strategy, see *Alternative Deletion Strategy* on page 33.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.ObjectServerB.Description` *string*	`-objectserverb description` *string*	Application description for the connection to ObjectServer B. This description is used in triggers and allows you to determine which component of the gateway attempted to perform an action. The default is "".
`Gate.ObjectServerB.FailbackEnabled` *boolean*	`-objectserverb failbackenabled` *boolean*	Allows you to enable failback for this ObjectServer. For details about failback functionality, see *Failback* on page 31. The default is `FALSE`.
`Gate.ObjectServerB.FailbackTimeout` *integer*	`-objectserverb failbacktimeout`	Time (in seconds) that the gateway allows before checking for the return of the master ObjectServer and failing back. The default is `30`.
`Gate.ObjectServerB.LogOSSql` *boolean*	`-objectserverb logossql` *boolean*	Specifies whether the gateway logs all SQL commands sent to this ObjectServer in debug mode. The default is `FALSE`.
`Gate.ObjectServerB.Password` *string*	`-objectserverb password` *string*	Password associated with the user specified by the `Gate.ObjectServerB.Username` property. This password *must* be encrypted by the `nco_g_crypt` utility. For details about this utility, see the Netcool/OMNIbus Administration Guide. Plain text passwords are not accepted as the gateway assumes that all passwords are encrypted by this tool for security. The default is "".
`Gate.ObjectServerB.ReconnectTimeout` *integer*	`-objectserverbreconn timeout` *integer*	If the gateway loses the connection to this ObjectServer, this property defines the time (in seconds) between each reconnection poll attempt. The default is `30`.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.ObjectServerB.RefreshCacheOnUpdate` *boolean*	`-objectserverbref cacheonupd` *boolean*	Controls the refreshing of the hash table cache for this ObjectServer: `TRUE` - Cache is resynchronized with the target ObjectServer prior to processing each collection of row updates for a table in the current `IDUC` window. `FALSE` - Gateway assumes that its cache is accurate and does not resynchronize it. The default is `FALSE`.
`Gate.ObjectServerB.SAF` *boolean*	`-objectserverbsaf` *boolean*	Instructs the gateway to store all table entries if the destination ObjectServer is unavailable and to forward them when the ObjectServer becomes available again. The default is `FALSE`.
`Gate.ObjectServerB.SAFFile` *string*	`-objectserverbsaffile` *string*	Name of the file that the gateway uses to store table entries while the destination ObjectServer is unavailable. The default is `$OMNIHOME/var/objserv_bi/NCO_GATE_ObjectServerB.store`. **Note:** This file is only used if the `Gate.ObjectServerB.SAF` property is set to `TRUE`.
`Gate.ObjectServerB.Server` *string*	`-objectserverbserver` *string*	Name of this ObjectServer. The default is `NCOMS`.
`Gate.ObjectServerB.Username` *string*	`-objectserverb username` *string*	Username used to authenticate the connection to this ObjectServer. This username is used to establish both the writer's IDUC connection and the subsidiary SQL command connection. The default is `root`.
`Gate.ObjectServerB.TblReplicateDefFile` *string*	`-objectserverbtbl repdeffile` *string*	Path to the table replication definition file. The default is `$OMNIHOME/gates/objserv_bi/objserv_bi.objectserverb.tblrep.def`. **Note:** For details about this file, see *Table Replication Definition File* on page 43.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.ObjectServerB.StatusTableName` *string*	`-objectserverb statustblname` *string*	Name of the status table that the gateway reads. The default is `alerts.status`.
`Gate.ObjectServerB.JournalTableName` *string*	`-objectserverb journaltblname` *string*	Name of the journal table that the gateway reads. The default is `alerts.journal`.
`Gate.ObjectServerB.DetailsTableName` *string*	`-objectserverb detailstblname` *string*	Name of the details table that the gateway reads. The default is `alerts.details`.
`Gate.ObjectServerB.IDUCFlushRate` *integer*	`-objectserverb iducflushrate` *integer*	Rate (in seconds) of the granularity of the reader. If you set this property to `0`, the reader gets its updates at the same granular rate as that of the ObjectServer to which it is connected. The default is `0`. **Warning:** If you set this property to a value greater than `0`, the reader issues automatic IDUC flush requests to the ObjectServer with this frequency. This enables the reader to run at a faster granularity than that of the ObjectServer, thus enabling the gateway to capture more detailed event changes in systems where the ObjectServer itself has high granularity settings.

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
Resynchronization Properties		
`Gate.Resync.Enable` *boolean*	`-resyncenable` *boolean*	Allows you to specify that the gateway uses resynchronization. The default is `TRUE`.
`Gate.Resync.Master` *string*	`-resyncmaster` *string*	Allows you to specify which ObjectServer the gateway should always treat as the master during resynchronization. Valid values are `ObjectServerA` and `ObjectServerB`. The default is "". **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*	Allows you 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. Valid values are `ObjectServerA` and `ObjectServerB`. The default is "".

Table 2: Properties Used by the Bidirectional ObjectServer Gateway (Continued)

Property Name	Command Line Options	Description
`Gate.Resync.Type` *string*	`-resynctype` *string*	Type of resynchronization that is required: `Normal` - Gateway determines which of the ObjectServers is the master and which the slave. It then deletes the contents of the slave ObjectServer and populates it with the contents of the master ObjectServer. `Update` - Specifies whether the resynchronization is performed with updates. This option ensures that updates made in the upper tier are not lost during the resynchronization. Use of this parameter is recommended within tiered architectures. The default is `NORMAL`. **Note:** To specify how the gateway determines which ObjectServer is the master and which the slave, use the `Gate.Resync.Master` and `Gate.Resync.Preferred` properties. If you leave both properties blank, the gateway checks the two ObjectServers and treats the one that has been running the longest as the master. If you leave both properties blank and both ObjectServers have been running for the same length of time, the gateway treats ObjectServer A as the master.

Hash Table Cache

The gateway uses a hash table cache to store details of tables that require transferring from one ObjectServer to another. 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. It is also used for any other tables specified using the table replication definition table.

The cache aids performance optimization by providing the gateway with a in-memory summarized view of the contents of the ObjectServers to which it is linked. This means that the gateway does not have to query an ObjectServer to check for the existence of an event, or the `Serial` value or `Tally` value of an event; it can check the cache of the target ObjectServer instead.

You can control the size of the hash table cache by using the `Gate.CacheHashTblSize` property. By default, the size of the hash table cache is `5023` elements (or rows). This can be increased if the status table has a large number of rows (for example, in excess of 20,000).

Note: To maximize efficiency, you should specify a prime number for the `Gate.CacheHashTblSize` property.

Error Handling

The gateway provides configurable error handling. Error handling is provided by the Netcool/OMNIbus Gateway Toolkit (NGTK) library. To specify that the NGTK library logs debug messages, set the `Gate.NGtkDebug` property to `TRUE`.

You can specify which debug messages are included in the debug files by using the `Gate.Mapper.Debug`, `Gate.ObjectServerA.Debug`, and `Gate.ObjectServerB.Debug` properties; these can be set to `TRUE` or `FALSE` as appropriate.

Process Agent Control

The gateway can be run under Process Agent (PA) control. The `Gate.PAAware` property indicates whether the gateway is PA aware. The `Gate.PAAwareName` property indicates which PA is running the gateway.

These properties are maintained automatically by the PA server and provide information only. They should not be changed manually.

Store and Forward

The gateway supports store and forward on any table when the destination ObjectServer goes offline. This feature can be configured to store and forward either all or none of the tables being replicated.

To activate the store and forward function for ObjectServer A, set the `Gate.ObjectServerA.SAF` property to `TRUE`, specify the file to which the alerts are written while the destination ObjectServer is offline using the `Gate.ObjectServerA.SAFFile` property, and restart the gateway. To deactivate the store and forward function, set the `Gate.ObjectServerA.SAF` property to `FALSE` and restart the gateway.

To activate the store and forward function for ObjectServer B, set the `Gate.ObjectServerB.SAF` property to `TRUE`, specify the file to which the alerts are written while the destination ObjectServer is offline using the `Gate.ObjectServerB.SAFFile` property, and restart the gateway. To deactivate the store and forward function, set the `Gate.ObjectServerB.SAF` property to `FALSE` and restart the gateway.

Failback

Failback functionality comes into operation when a gateway loses its connection to the primary ObjectServer; this enables the gateway to connect to a backup ObjectServer. Failback functionality also enables the gateway to reconnect to the primary ObjectServer when it becomes active again. Figure 4 shows the failover-failback relationship between the primary and secondary ObejctServers to which an ObjectServer gateway is connected.

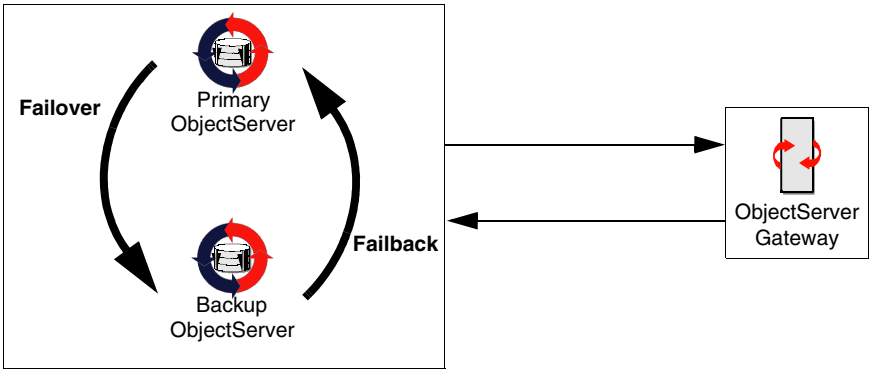


Figure 4: Failback Functionality

Setting Up Failback

To set up failback, set the ObjectServer's `BackupObjectServer` property for the backup ObjectServer to `TRUE`. For details about setting ObjectServer properties, see the Netcool/OMNIbus Administration Guide.

To enable failback, in the gateway properties file, you must set the `Gate.ObjectServerA.Failback` and `Gate.ObjectServerB.Failback` properties to `TRUE`. When the primary ObjectServer fails, the reader and writer fail over to the backup ObjectServer without shutting down. When the reader or writer have detected that it is connected to a backup ObjectServer, it periodically polls for the return of the primary ObjectServer. When the primary ObjectServer has been detected again, the reader or writer automatically fail back to the primary ObjectServer.

To specify the frequency with which the reader and writer parts of the gateway poll the failed ObjectServer, set the `Gate.ObjectServerA.FailbackTimeout` and `Gate.ObjectServerB.FailbackTimeout` properties.

Resynchronization

The gateway supports two resynchronization modes:

- `NORMAL` - Gateway deletes matching data from the destination table and inserts data from the source table.
- `UPDATE` - Gateway inserts rows that are not in the destination table and updates rows in the destination table with the current source values. When events already exist in the destination table, the gateway can be configured to use some destination column values in preference to data from the source table. This filtered column data can then be written back to the source table so that both sides are consistent.

To specify that the gateway uses resynchronization, set the `Gate.Resync.Enable` property to `TRUE`. To specify which type of resynchronization the gateway uses, set the `Gate.Resync.Type` property.

When two ObjectServers are linked by a bidirectional gateway, it regards one as the master and the other as the servant. By default, the gateway treats the ObjectServer that has been running the longest as the master.

You can instruct the gateway to always treat a specific ObjectServer as the master using the `Gate.Resync.Master` property. Alternatively, you can nominate which ObjectServer is the preferred master using the `Gate.Resync.Preferred` property. This indicates which ObjectServer to use as the master when both ObjectServers have been running for the same length of time.

Buffer Size

The buffer size controls the number of entries that the gateway stores in its buffer before flushing them to the ObjectServer. The gateway uses separate buffers for the source and destination ObjectServers and their sizes can be set using the `Gate.ObjectServerA.Buffersize` and `Gate.ObjectServerB.Buffersize` properties. These properties can be adjusted to fine-tune the efficiency of the gateway.

The optimum value for the buffer size depends upon the average event size and the speed of the network. The default value has proved to be efficient for many installations. To determine the most efficient setting for your system, compare the timing figures for resynchronisation operations performed using different settings for this property.

Alternative Deletion Strategy

In most cases, the gateway always applies deletes. However, you may have a system in which gateways are used to collect insert, update, and delete events from a set of ObjectServers and pass them on to an ObjectServer that aggregates them and only passes back deletes.

For example, the aggregation ObjectServer may make available events pooled from a set of ObjectServers. Deletion events would be passed back by the aggregation ObjectServer as events are addressed. In this situation, you may want to prevent a deletion from being applied if the event in the target server indicates that the event has occurred again since the delete was issued; for example, if an attempted resolution to a problem has failed.

To set up this deletion strategy, set the `Gate.ObjectServerA.DeleteIfNoDedup` and `Gate.ObjectServerB.DeleteIfNoDedup` properties to `TRUE`.

Map Definition File

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. If you want to replicate user related system tables (for example, `SecurityUsers`, `SecurityGroups`, `SecurityRoles`, `SecurityRoleGrants`, and `SecurityGroupMembers`), you must also include details of these mappings in this file. The path of the map definition file is determined by the `Gate.Mapfile` property in the properties file. For details, see *Properties File for the Unidirectional Gateway* on page 7.

Syntax

Mappings for use with the ObjectServer writer must follow this syntax:

```
CREATE MAPPING mappingname
(
' dest_fieldname' = ( '@src_fieldname' | simple_expression | attribute ) [ ON INSERT
ONLY ] [ CONVERT TO type ] [ NOT NULL ('@src_fieldname')]
[ , '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 an integer or a set of integers and operators.
- *attribute* is an attribute name as described in *Attribute Names* on page 35.

The optional `ON INSERT ONLY` controls the updating of the field during the life of the alert; when omitted, the field is updated when any change in the state of the alert occurs. When included, the field is created once for the alert, but is never updated.

The optional `CONVERT TO` *type* enables the mapping to define a forced conversion for situations where a source field may not match the type of the destination field. The *type* can be `INTEGER`, `STRING`, or `DATE`.

The optional `NOT NULL` indicates that the mapping is only performed if the source field is not null; that is, has a value set for it.

Note: The ordering of the fields in the mapping must exactly match the ordering of the fields in the ObjectServer table being written to. If this is not the case, values are written to the incorrect fields or the writer generates invalid SQL commands.

Attribute Names

Attribute names enable you to include additional data in mapping definitions. The following types of attributes can be specified:

- Cache value access attributes
- Dynamic attributes

This section describes the attribute names can be used in the map definition file.

Cache Value Access Attributes

Cache value attributes enable the gateway to access values stored in the cross-reference cache. Table 3 describes the cache value attributes that can be used in mapping definitions.

Table 3: Cache Value Access Attributes

Attribute Name	Description
`STATUS.SERIAL`	Cached serial number for the status table row associated with the current journal or details table row.
`STATUS.SERVER_SERIAL`	Cached server serial number for the status table row associated with the current journal or details table row.
`STATUS.SERVER_NAME`	Cached server name for the status table row associated with the current journal or details table row.
`STATUS.IDENTIFIER`	Cached identifier for the status table row 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. Table 4 describes the dynamic attributes that can be used in mapping definitions.

Table 4: Dynamic Attributes

Attribute Name	Description
ACTION_CODE	Single character string that specifies the type of operation performed: I - Insert U - Update D - Delete
ACTION_TIME	Time in UTC that the action occurred.
DELETEDAT	If the action was a delete, this attributes displays the date at which the row was deleted.

Conversion Functions

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

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

Example Mappings

The following example shows the mappings for the ObjectServer tables into which the gateway writes:

```
CREATE MAPPING StatusMap
(
'Identifier'        =                   '@Identifier'ON INSERT ONLY,
'Serial'            =                   '@Serial'ON INSERT ONLY,
'Node'              =                   '@Node'ON INSERT ONLY,
'NodeAlias'         =                   '@NodeAlias'ON INSERT ONLY
                                         NOTNULL '@Node',
'Manager'           =                   '@Manager'ON INSERT ONLY,
'Agent'             =                   '@Agent'ON INSERT ONLY,
'AlertGroup'        =                   '@AlertGroup'ON INSERT ONLY,
'AlertKey'          =                   '@AlertKey'ON INSERT ONLY,
'Severity'          =                   '@Severity',
'Summary'           =                   '@Summary',
'Location'          =                   '@Location'ON INSERT ONLY,
'StateChange'       =                   '@StateChange',
'FirstOccurrence'   =                   '@FirstOccurrence'ON INSERT ONLY,
'LastOccurrence'    =                   '@LastOccurrence',
'Poll'              =                   '@Poll'ON INSERT ONLY,
'Type'              =                   '@Type'ON INSERT ONLY,
'Tally'             =                   '@Tally',
'Class'             =                   '@Class'ON INSERT ONLY,
'OwnerUID'          =                   '@OwnerUID',
'OwnerGID'          =                   '@OwnerGID',
'Acknowledged'      =                   '@Acknowledged',
'Flash'             =                   '@Flash',
'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 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'
# );
```

Startup Command File

The startup command file contains a set of commands that the gateway executes each time it starts.

Commands

You can include the following commands in the startup command file:

- `SET PROPERTY`
- `GET PROPERTY`
- `SHOW PROPS`
- `GET CONFIG`
- `SET LOG LEVEL TO`
- `TRANSFER`
- `FAILOVER_SYNC`

Note: You can also use `nco_sql` to issue these commands manually when the gateway is running. For details about `nco_sql`, see the Netcool/OMNIbus Administration Guide.

SET PROPERTY

Sets the property to the value specified.

Syntax

```
SET PROPERTY '<string>' TO <property_value> ;
```

Example

```
SET PROPERTY 'Gate.Reader.Debug' TO FALSE ;
```

GET PROPERTY

Returns the value of the property specified.

Syntax

```
GET PROPERTY '<string>' ;
```

Example

```
GET PROPERTY 'Gate.Reader.Debug' ;
```

SHOW PROPS

Displays the gateway's current configuration by listing all properties and their values.

Syntax

```
SHOW PROPS ;
```

Example

```
SHOW PROPS ;
go
```

GET CONFIG

Displays the gateway's current configurationby listing all properties and their values.

Note: `GET CONFIG` is equivalent to `SHOW PROPS` and may be removed from later versions of the ObjectServer gateway.

Syntax

```
GET CONFIG ;
```

Example

```
GET CONFIG ;
go
```

SET LOG LEVEL TO

Changes the gateway's log level to the specified setting.

Syntax

```
SET LOG LEVEL TO level;
```

Where *level* can be:

- debug
- information
- warning
- error
- fatal

Example

```
SET LOG LEVEL TO debug;
go
```

TRANSFER

Initiates a data transfer operation between tables in the two ObjectServers. You can transfer a partial data set using a filter condition. If the target table does not match the source table, a map can be specified as part of the transfer operation. This map must be defined within the gateway's map definition file. For details about the map definition file, see *Map Definition File* on page 34.

Syntax

```
TRANSFER FROM sourcetable TO targettable
[VIA FILTER filtertext] [WITH DELETE VIA delfiltertext]
[USING TRANSFER_MAP mapname] ;
```

Examples

```
TRANSFER FROM 'master.names' TO 'resync.names'
VIA FILTER 'Name != \'nobody\''
DELETE ;
```

In this example, all records with the name `nobody` are deleted from the table `resync.names` and replaced with the corresponding records in the table `master.names`.

FAILOVER_SYNC

FAILOVER_SYNC is used to synchronize master table data between primary and backup ObjectServers. The FAILOVER_SYNC command specifies which master tables to transfer during the data transfer activity. For information about ObjectServer failover, see the Netcool/OMNIbus Administration Guide.

Syntax

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

Examples

```
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;
```

Table Replication Definition File

The table replication definition file defines the tables in the source ObjectServer that should be replicated in the target system using the ObjectServer Gateway. For unidirectional gateways, the table definition file is specified using the `Gate.Reader.TblReplicateDefFile` property. For bidirectional gateways, a table definition file can be specified for both ObjectServers using the `Gate.ObjectServerA.TblReplicateDefFile` and `Gate.ObjectServerB.TblReplicateDefFile` properties.

Syntax

```
REPLICATE ALL | (INSERTS, UPDATES, DELETES)
       FROM TABLE sourcetable
       USING MAP mapname
       [FILTER WITH filter_clause]
       [INTO targettable]
       [ORDER BY column_name [ASC|DESC], ... ]
       [WITH NORESYNC]
       [SET UPDTOINS CHECK TO {ENABLED|DISABLED|FORCED}]
       [AFTER IDUC DO command] ;
```

Where:

- *sourcetable* is the table to be replicated in the target ObjectServer.
- *mapname* is the map definition that the defines the table.
- *column_name* is the column by which the rows returned to the gateway from the ObjectServer are ordered.
- *filter_clause* defines the filter the gateway uses to select rows for replicating.
- *propertyname* is the property the gateway uses to filter the table data (only rows that satisfy the filter are replicated).
- *propertyvalue* is argument to be used in the filter.
- *targettable* is the name of the table in which to replicate the data.

The optional `WITH NORESYNC` option allows you to specify tables that should not be resynchronized.

The optional `ORDER BY` option allows you to specify the order in which the rows are returned to the gateway from the ObjectServer. `ASC` specifies that the rows are sorted in ascending order; `DESC` specifies descending order. If you specify neither `ASC` nor `DESC`, the rows are sorted in ascending order.

You can define multiple columns for sorting by specifying a comma-separated list; for example:

```
ORDER BY 'Serial DESC, StateChange ASC'
```

The optional `SET UPDTOINS CHECK TO` option allows you to configure the update-to-insert functionality. `ENABLED` instructs the gateway to perform normal update-to-insert conversion checking; `DISABLED` instructs the gateway to perform no update-to-insert conversion checking; `FORCED` instructs the gateway to convert all updates to inserts regardless of whether the events already exist in the target system.

The optional `AFTER IDUC DO` option allows you to specify a column and associated value that the gateway applies to all rows that have been inserted, updated, or deleted.

Example Table Replication Definition File

The following shows an example 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\'';

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';
```

Contact Information

Corporate

Region	Address	Telephone	Fax	World Wide Web
USA	Micromuse Inc. (HQ) 139 Townsend Street San Francisco CA 94107 USA	1-800-Netcool (638 2665) +1 415 538 9090	+1 415 538 9091	http://www.micromuse.com
EUROPE	Micromuse Ltd. Disraeli House 90 Putney Bridge Road London SW18 1DA United Kingdom	+44 (0) 20 8875 9500	+44 (0) 20 8875 9995	http://www.micromuse.co.uk
ASIA-PACIFIC	Micromuse Ltd. Level 2 26 Colin Street West Perth Perth WA 6005 Australia	+61 (0) 8 9213 3400	+61 (0) 8 9486 1116	http://www.micromuse.com.au

Technical Support

Region	Telephone	Fax
USA	1-800-Netcool (800 638 2665) +1 415 538 9090 (San Francisco)	+1 415 538 9091
EUROPE	+44 (0) 20 8877 0073 (London, UK)	+44 (0) 20 8875 0991
ASIA-PACIFIC	+61 (0) 8 9213 3470 (Perth, Australia)	+61 (0) 8 9486 1116
	E-mail	World Wide Web
GLOBAL	support@micromuse.com	http://support.micromuse.com

License Generation Team

E-Mail	World Wide Web
licensing@micromuse.com	http://support.micromuse.com/helpdesk/licenses