

Netcool®/OMNIbus™

v7

Administration Guide

© 2004 Micromuse Inc., Micromuse Ltd.

All rights reserved. No part of this work may be reproduced in any form or by any person without prior written permission of the copyright owner. This document is proprietary and confidential to Micromuse, and is subject to a confidentiality agreement, as well as applicable common and statutory law.

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 TeX 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, Domino, Lotus, Lotus Notes, 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.

InstallShield is a registered trademark of InstallShield Software Corporation.

Microsoft, Windows 95/98/Me/NT/2000/XP are either registered trademarks or trademarks of Microsoft Corporation.

Microsoft Active Directory, Microsoft Internet Information Server/Services (IIS), Microsoft Exchange Server, Microsoft SQL Server, Microsoft perfmom and Microsoft Cluster Service are registered trademarks of Microsoft Corporation.

BEA and WebLogic are registered trademarks of BEA Systems Inc.

FireWall-1 is a registered trademark of Check Point Software Technologies Ltd.

Netscape and Netscape Navigator are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Netscape's logos and Netscape product and service names are also trademarks of Netscape Communications Corporation, which may be registered in other countries.

Micromuse acknowledges the use of Xpm tool kit components.

SentinelLM is a trademark of Rainbow Technologies Inc.

GLOBETrotter and FLEXIm are registered trademarks of Globetrotter Software Inc.

Red Hat, the Red Hat "Shadow Man" logo, RPM, Maximum RPM, the RPM logo, Linux Library, PowerTools, Linux Undercover, RHmember, RHmember More, Rough Cuts, Rawhide and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat Inc. in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Nokia is a registered trademark of Nokia Corporation.

WAP Forum™ and all trademarks, service marks and logos based on these designations (Trademarks) are marks of Wireless Application Protocol Forum Ltd.

Micromuse acknowledges the use of InstallAnywhere software in Netcool/WAP Service Monitors. Copyright © Zero G Software Inc.

Orbix is a registered trademark of IONA Technologies PLC. Orbix 2000 is a trademark of IONA Technologies PLC.

Micromuse acknowledges the use of Graph Layout Toolkit in Netcool/ Precision for IP Networks. Copyright © 1992 - 2001, Tom Sawyer Software, Berkeley, California. All rights reserved.

Portions of Netcool/Precision for IP Networks are © TIBCO Software, Inc. 1994-2003. All rights reserved. TIB and TIB/Rendezvous are trademarks of TIBCO Software, Inc.

Portions of Netcool/Precision for IP Networks are Copyright © 1996-2003, Daniel Stenberg, <daniel@haxx.se>.

SAP, R/2 and R/3 are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle is a registered trademark of Oracle Corporation.

Micromuse acknowledges the use of Digital X11 in Netcool/Precision for IP Networks. Copyright 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts. All Rights Reserved. DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Netcool/Service Monitors include software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

Micromuse acknowledges the use of software developed by ObjectPlanet. ©2003 ObjectPlanet, Inc, Ovre Slottsgate, 0157 Oslo, Norway.

Micromuse acknowledges the use of Expat in Netcool/ASM. Copyright 1998, 1999, 2000 Thai Open Source Software Center Ltd and Clark Cooper. Copyright 2001, 2002 Expat maintainers. THE EXPAT SOFTWARE IS PROVIDED HEREUNDER "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 AUTHORS OR COPYRIGHT HOLDERS OF THE EXPAT SOFTWARE 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 EXPAT SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Expat explicitly grants its permission to any person obtaining a copy of any Expat software and

associated documentation files (the "Expat Software") to deal in the Expat Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Expat Software. Expat's permission is subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Expat Software. Except as set forth hereunder, all software provided by Micromuse hereunder is subject to the applicable license agreement.

All other trademarks, registered trademarks and logos are the property of their respective owners.

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

www.micromuse.com

Document Version Number: 1.1

Contents

- Preface 1
- Audience..... 2
- About the Netcool/OMNIbus v7 Administration Guide 3
- Associated Publications 4
 - Netcool@/OMNIbus™ Installation and Deployment Guide 4
 - Netcool@/OMNIbus™ User Guide 4
 - Netcool@/OMNIbus™ Administration Guide..... 4
 - Netcool@/OMNIbus™ Probe and Gateway Guide 4
 - Online Help 4
- Typographical Notation 5
 - Note, Tip, and Warning Information 6
 - Syntax and Example Subheadings 7
- Operating System Considerations 8

- Chapter 1: Configuring the ObjectServer and Proxy Server..... 9**
- ObjectServer Overview 10
- Creating the ObjectServer 11
 - Database Initialization Files 11
 - nco_dbinit Properties and Command Line Options 12
- Starting and Stopping the ObjectServer 16
 - Starting and Stopping an ObjectServer Using Process Control (UNIX)..... 16
 - Starting and Stopping an ObjectServer Using Services (Windows)..... 16
 - Starting an ObjectServer Manually 17
 - Configuring a Running ObjectServer..... 17
 - Stopping an ObjectServer Manually..... 18

ObjectServer Properties and Command Line Options	19
Specifying ObjectServer Command Line Options	19
Specifying ObjectServer Properties	19
How Properties and Command Line Options Are Processed at Startup	19
Running the ObjectServer in Secure Mode	30
Client Tool Updates Using IDUC	32
Specifying the IDUC Update Interval	32
Specifying the IDUC Port	32
Starting the Proxy Server	34
Connecting to the Proxy Server	34
Proxy Server Properties and Command Line Options	35
Running the Proxy Server in Secure Mode	38
Checkpointing	39
Checkpoint File Storage	39
Data Recovery During ObjectServer Startup	40
Checkpoint Verification Utility	40
Chapter 2: Netcool/OMNIBus Administrator	43
Introduction to Netcool/OMNIBus Administrator	44
Prerequisites	44
Starting Netcool/OMNIBus Administrator and Connecting to an ObjectServer	45
On UNIX	45
On Windows	45
Netcool/OMNIBus Administrator Properties and Command Line Options	46
Property and Command Line Processing	48
Logging out of Netcool/OMNIBus Administrator	49

Netcool/OMNIbus Administrator Overview.....	50
Secure Sockets Layer Connections.....	50
Selecting ObjectServer Objects.....	51
Using Netcool/OMNIbus Administrator Windows.....	52
Configuring Editor Syntax Coloring.....	54
Selecting a Web Browser for Displaying Online Help.....	54
Managing Roles, Groups, and Users.....	55
Configuring Roles.....	55
Configuring Groups.....	58
Configuring Users.....	63
Configuring ObjectServer Objects.....	68
ObjectServer Object Naming Conventions.....	68
Configuring Restriction Filters.....	69
Configuring Event List Menus.....	71
Configuring Tools.....	75
Configuring Prompts.....	79
Configuring Triggers.....	83
Configuring Procedures.....	91
Configuring Conversions.....	96
Configuring Event List Alert Severity Colors (Windows Event Lists Only).....	98
Configuring Column Visuals.....	100
Configuring Classes.....	102
Viewing and Changing ObjectServer Properties.....	104
Configuring Databases.....	106
Configuring User Defined Signals.....	111
Configuring ObjectServer Files.....	115
Accessing an ObjectServer Using the SQL Interactive Interface (iSQL).....	119
Opening iSQL.....	120

Chapter 3: ObjectServer SQL	123
Alert Processing in the ObjectServer	124
Introduction to Deduplication	124
Additional Alert Processing Capabilities	125
Introduction to ObjectServer SQL	126
ObjectServer Naming Conventions	126
Using the SQL Interactive Interface	128
Starting the SQL Interactive Interface	128
Executing SQL Commands in the SQL Interactive Interface	129
Exiting the SQL Interactive Interface	131
Using the Secure SQL Interactive Interface	132
Encrypting Passwords in UNIX nco_sql Scripts	132
Storage Structures and Data Definition Language SQL Commands	133
Databases	133
Tables	135
Views	140
Restriction Filters	142
Files	143
Reserved Words	147
SQL Building Blocks: Operators, Functions, Expressions, and Conditions	150
Operators	150
Functions	157
Expressions	161
Conditions	162

Data Manipulation Language SQL Commands	164
Adding Table Data: The INSERT Command	164
Modifying Table Data: The UPDATE Command	165
Removing Table Data: The DELETE Command	166
Viewing Table Data: The SELECT Command	167
Logging to Files: The WRITE INTO Command	171
Viewing Table Column Information: The DESCRIBE Command	172
Adding and Modifying Service Status Data: The SVC Command	172
System and Session SQL Commands	174
Managing the ObjectServer: The ALTER SYSTEM Command	174
Changing the Default Database: The SET and USE DATABASE Commands	175
Verifying SQL Syntax: The CHECK STATEMENT Command	176
Security SQL Commands	177
Creating, Modifying, and Deleting Users and Groups	177
Creating, Modifying, and Deleting Roles	181
Assigning System and Object Permissions to Roles	183
Granting Roles to Groups	187
Procedures	189
Creating SQL Procedures	189
Creating External Procedures	198
Executing Procedures	199
Dropping Procedures	200
Automation: Triggers and Trigger Groups	201
Types of Triggers	201
Trigger Groups	202
Syntax Elements Common to All Types of Triggers	203
Database Triggers	207
Temporal Triggers	210
Signals and Signal Triggers	211

Automation Examples	220
Trigger to Deduplicate the Status Table	220
Trigger to Deduplicate the Details Table	220
Trigger to Clean the Details Table	221
Trigger to Set the Alerts Table StateChange Column	221
Trigger to Delete Clears	221
Trigger to E-mail on Critical Alerts	222
Chapter 4: Process Control	223
Introduction	224
Execution of External Procedures in Automations	224
Configuring and Managing Netcool/OMNIBus Components on UNIX	224
Process Agents on UNIX	225
Process Agents on Windows	225
Process Control Components	226
Process Control Agents	226
Processes	226
Services	227
Process Control Utilities	227
Creating a Process Control System Configuration	228
Before You Configure Process Control	228
Creating User Groups	228
Configuring Process Control Agents	228
Defining Processes, Services, and Hosts	231
Defining Processes	231
Defining Services	233
Defining Secure Hosts	235
Defining Routing Hosts, User Names, and Passwords	236
Example Process Agent Configuration File	238
Process Control Agent Daemon Command Line Options	240

Process Control Management	244
Displaying Service Status - nco_pa_status	244
Starting a Service or Process - nco_pa_start	245
Stopping a Service or Process - nco_pa_stop	246
Shutting Down a Process Control Agent - nco_pa_shutdown	247
Adding a New Service or Process to a Running PAD - nco_pa_addentry	248
Appendix A: ObjectServer Tables	251
Alerts Tables	252
alerts.status Table	252
alerts.details Table	258
alerts.journal Table	259
Service Tables	261
service.status Table	261

System Catalog Tables	262
catalog.databases Table	262
catalog.tables Table	262
catalog.base_tables Table	263
catalog.views Table	264
catalog.files Table	264
catalog.restrictions Table	265
catalog.columns Table	265
catalog.primitive_signals Table	266
catalog.primitive_signal_parameters Table	266
catalog.trigger_groups Table	267
catalog.triggers Table	267
catalog.database_triggers Table	268
catalog.signal_triggers Table	269
catalog.temporal_triggers Table	269
catalog.procedures Table	269
catalog.sql_procedures Table	270
catalog.external_procedures Table	270
catalog.procedure_parameters Table	271
catalog.connections Table	271
catalog.properties Table	272
catalog.security_permissions Table	273
catalog.profiles Table	273
Statistics Tables	274
catalog.profiles Table	274
master.stats Table	275
catalog.trigger_stats Table	276

Client Tool Support Tables	278
alerts.objclass Table	278
alerts.objmenus Table	278
alerts.objmenuitems Table	278
alerts.resolutions Table	279
alerts.conversions Table	280
alerts.col_visuals Table	280
alerts.colors Table	281
Desktop Tools Tables	283
tools.actions Table	283
tools.action_access Table	285
tools.menus Table	285
tools.menu_items Table	285
tools.prompt_defs Table	286
tools.menu_defs Table	287
Desktop ObjectServer Tables	288
master.national Table	288
master.servergroups Table	288
Security Tables for Backward Compatibility	290
Appendix B: Desktop Reference	291
Event List Command Line Options	292
UNIX Command Line Options	292
Windows Command Line Options	294
Using the Transient Event List (nco_elct)	296
UNIX Command Line Options	296
Windows Command Line Options	298
Using nco_elct Example - Command Line or Script	299
Using nco_elct Example - In a Tool	300

SQL Commands and Variable Expressions in Tools, Automations, and Transient Event Lists 302

Changing the Date Format in the Event List on UNIX. 305

Contact Information. 317

Preface

This guide describes how to configure and manage Netcool/OMNIbus. The following chapters and appendices describe each functional area, and task-oriented examples are provided to assist administrators.

This preface contains the following sections:

- *Audience* on page 2
- *About the Netcool/OMNIbus v7 Administration Guide* on page 3
- *Associated Publications* on page 4
- *Typographical Notation* on page 5
- *Operating System Considerations* on page 8

Audience

This guide is intended for administrators, and provides detailed cross-platform information about administrative tools, functions, and capabilities. In addition, it is designed to be used as a reference guide to assist you in designing and configuring your environment.

Much of the information about Netcool/OMNIbus Administrator contained in this document is also provided on-line within the help system.

It is assumed that you understand how Netcool/OMNIbus works. For more information on Netcool/OMNIbus, refer to the publications described in *Associated Publications* on page 4.

About the Netcool/OMNIbus v7 Administration Guide

This book is organized as follows:

- Chapter 1: *Configuring the ObjectServer and Proxy Server* on page 9 describes how to create, start, and stop the ObjectServer and proxy server, and how to configure the ObjectServer and proxy server using properties and command line options.
- Chapter 2: *Netcool/OMNIbus Administrator* on page 43 contains information about using Netcool/OMNIbus Administrator to configure and manage ObjectServers.
- Chapter 3: *ObjectServer SQL* on page 123 describes how alerts are stored and managed in the ObjectServer. It also describes the data structures of the ObjectServer and the syntax of ObjectServer SQL.
- Chapter 4: *Process Control* on page 223 describes the components, configuration, and management utilities associated with the Netcool/OMNIbus process control system.
- Appendix A: *ObjectServer Tables* on page 251 contains ObjectServer database table information.
- Appendix B: *Desktop Reference* on page 291 contains reference information for the Netcool/OMNIbus desktop.

Associated Publications

To efficiently administer Netcool/OMNIBus, you must possess an understanding of the Netcool/OMNIBus technology. This section provides a description of the documentation that accompanies Netcool/OMNIBus.

Netcool®/OMNIBus™ Installation and Deployment Guide

This book is intended for Netcool administrators who need to install and deploy Netcool/OMNIBus. It includes installation, upgrade, and licensing procedures. In addition, it contains information about configuring security and component communications. It also includes examples of Netcool/OMNIBus architectures and how to implement them.

Netcool®/OMNIBus™ User Guide

This book is intended for anyone who needs to use Netcool/OMNIBus desktop tools on UNIX or Windows platforms. It provides an overview of Netcool/OMNIBus components, as well as a description of the operator tasks related to event management using the desktop tools.

Netcool®/OMNIBus™ Administration Guide

This book is intended for system administrators who need to manage Netcool/OMNIBus. It describes how to perform administrative tasks using the Netcool/OMNIBus Administrator GUI, command line tools, and process control. It also contains descriptions and examples of ObjectServer SQL syntax and automations.

Netcool®/OMNIBus™ Probe and Gateway Guide

This guide contains introductory and reference information about probes and gateways, including probe rules file syntax and gateway commands. For more information about specific probes and gateways, refer to the documentation available for each probe and gateway on the Micromuse Support Site.

Online Help

Netcool/OMNIBus GUIs contain context-sensitive online help with index and search capabilities.

Typographical Notation

Table 1 shows the typographical notation and conventions used to describe commands, SQL syntax, and graphical user interface (GUI) features. This notation is used throughout this book and other Netcool® publications.

Table 1: Typographical Notation and Conventions (1 of 2)

Example	Description
Monospace	<p>The following are described in a monospace font:</p> <ul style="list-style-type: none"> • Commands and command line options • Screen representations • Source code • Object names • Program names • SQL syntax elements • File, path, and directory names <p>Italicized monospace text indicates a variable that the user must populate. For example, <code>-password password</code>.</p>
Bold	<p>The following application characteristics are described in a bold font style:</p> <ul style="list-style-type: none"> • Buttons • Frames • Text fields • Menu entries <p>A bold arrow symbol indicates a menu entry selection. For example, File→Save.</p>
<i>Italic</i>	<p>The following are described in an italic font style:</p> <ul style="list-style-type: none"> • An application window name; for example, the <i>Login</i> window • Information that the user must enter • The introduction of a new term or definition • Emphasized text

Table 1: Typographical Notation and Conventions (2 of 2)

Example	Description
[1]	Code or command examples are occasionally prefixed with a line number in square brackets. For example: [1] First command... [2] Second command... [3] Third command...
{ a b }	In SQL syntax notation, curly brackets enclose two or more required alternative choices, separated by vertical bars.
[]	In SQL syntax notation, square brackets indicate an optional element or clause. Multiple elements or clauses are separated by vertical bars.
	In SQL syntax notation, vertical bars separate two or more alternative syntax elements.
...	In SQL syntax notation, ellipses indicate that the preceding element can be repeated. The repetition is unlimited unless otherwise indicated.
, ...	In SQL syntax notation, ellipses preceded by a comma indicate that the preceding element can be repeated, with each repeated element separated from the last by a comma. The repetition is unlimited unless otherwise indicated.
<u>a</u>	In SQL syntax notation, an underlined element indicates a default option.
()	In SQL syntax notation, parentheses appearing within the statement syntax are part of the syntax and should be typed as shown unless otherwise indicated.

Many Netcool commands have one or more command line options that can be specified following a hyphen (-).

Command line options can be `string`, `integer`, or `BOOLEAN` types:

- A `string` can contain alphanumeric characters. If the string has spaces in it, enclose it in quotation (") marks.
- An `integer` must contain a positive whole number or zero (0).
- A `BOOLEAN` must be set to `TRUE` or `FALSE`.

SQL keywords are not case-sensitive, and may appear in uppercase, lowercase, or mixed case. Names of ObjectServer objects and identifiers are case-sensitive.

Note, Tip, and Warning Information

The following types of information boxes are used in the documentation:



Note: Note is used for extra information about the feature or operation that is being described. Essentially, this is for extra data that is important but not vital to the user.



Tip: Tip is used for additional information that might be useful for the user. For example, when describing an installation process, there might be a shortcut that could be used instead of following the standard installation instructions.



Warning: Warning is used for highlighting vital instructions, cautions, or critical information. Pay close attention to warnings, as they contain information that is vital to the successful use of our products.

Syntax and Example Subheadings

The following types of constrained subheading are used in the documentation:



Syntax

Syntax subheadings contain examples of ObjectServer SQL syntax commands and their usage. For example:

```
CREATE DATABASE database_name;
```



Example

Example subheadings describe typical or generic scenarios, or samples of code. For example:

```
[1] <body>
[2]   
[6] </body>
```

Operating System Considerations

All command line formats and examples are for the standard UNIX shell. UNIX is case-sensitive. You must type commands in the case shown in the book.

Unless otherwise specified, command files are located in the `$OMNIHOME/bin` directory, where `$OMNIHOME` is the UNIX environment variable that contains the path to the Netcool/OMNIBus home directory.

On Microsoft Windows platforms, replace `$OMNIHOME` with `%OMNIHOME%` and the forward slash (/) with a backward slash (\).

Chapter 1: Configuring the ObjectServer and Proxy Server

This chapter describes how to create, start, and stop the ObjectServer and proxy server, and how to configure the ObjectServer and proxy server using properties and command line options. It also describes how to configure the ObjectServer and proxy server to only accept secure connections.

For information about installing Netcool/OMNIBus and configuring deployments, see the Netcool/OMNIBus Installation and Deployment Guide.

For information on how alerts are stored and managed in the ObjectServer and ObjectServer SQL, see Chapter 3: *ObjectServer SQL* on page 123.

This chapter contains the following sections:

- *ObjectServer Overview* on page 10
- *Creating the ObjectServer* on page 11
- *Starting and Stopping the ObjectServer* on page 16
- *ObjectServer Properties and Command Line Options* on page 19
- *Running the ObjectServer in Secure Mode* on page 30
- *Client Tool Updates Using IDUC* on page 32
- *Starting the Proxy Server* on page 34
- *Proxy Server Properties and Command Line Options* on page 35
- *Running the Proxy Server in Secure Mode* on page 38
- *Checkpointing* on page 39

1.1 ObjectServer Overview

The ObjectServer is the database server at the core of Netcool/OMNIbus. Alert information is forwarded to the ObjectServer from external programs such as probes, monitors, and gateways, stored and managed in database tables, and displayed in the event list.

In a standard configuration, alerts are forwarded directly to the ObjectServer. You can use the proxy server to reduce the number of probe connections to an ObjectServer.

You can run the ObjectServer and proxy server in secure mode. In secure mode, the ObjectServer and proxy server authenticate probe, gateway, and proxy server connection requests by requiring a user name and an encrypted password.

The ObjectServer supports persistence of data using disk-based checkpoints and logs. Checkpoints write all data to disk at system-defined intervals to enable data recovery if the server stops unexpectedly. Between checkpoints, additional modifications to the database are logged to disk.

1.2 Creating the ObjectServer

This section describes how to create the ObjectServer.

To create a new ObjectServer, use the `nco_dbinit` utility, which does the following:

- Creates the ObjectServer properties file in the `$OMNIHOME/etc` directory
- Creates the default database tables and data
- Creates default users, groups, and roles

You can create one or more ObjectServers on a host machine.

Table 2: `nco_dbinit` Database Creation Utility

To create an ObjectServer on...	Enter the following command...
UNIX	<code>\$OMNIHOME/bin/nco_dbinit -server <i>servername</i></code>
Windows	<code>%OMNIHOME%\bin\nco_dbinit -server <i>servername</i></code>

In this command, *servername* is the name of the new ObjectServer. Additional command line options for the `nco_dbinit` utility are described in *nco_dbinit Properties and Command Line Options* on page 12.

Database Initialization Files

The `nco_dbinit` utility uses the following SQL files to create the default database tables and data for the new ObjectServer:

- `application.sql` – creates the default tables for the alerts and tools databases.
- `automation.sql` – creates the default trigger groups and triggers.
- `desktop.sql` – specifies default values for the desktop tables, including default colors, conversions, tools, and menus.
- `system.sql` – specifies the security database and tables, and system users, groups, roles, and permissions. You must not edit this file.
- `security.sql` – specifies additional operator and administrator roles.

During database initialization, a properties file is created for the new ObjectServer. This properties file, `$OMNIHOME/etc/servername.props`, contains the settings for configuring the ObjectServer. ObjectServer properties and their default values are described in *ObjectServer Properties and Command Line Options* on page 19.

Additionally, the default ObjectServer `root` and `nobody` users are created, along with default groups and roles to help you manage permissions. You can configure users, groups, and roles using Netcool/OMNIBus Administrator or ObjectServer SQL, as described in this guide. For an overview of user access security, communication security, and default groups and roles, see the Netcool/OMNIBus Installation Guide.



Tip: The `root` and `nobody` users are created with an empty string as a password by default. You can reset the passwords using Netcool/OMNIBus Administrator or the `ALTER USER` ObjectServer SQL command.

nco_dbinit Properties and Command Line Options

The `nco_dbinit` utility reads the properties file when it starts. If a property is not specified in this file, the default value is used unless a command line option is used to override it. The default location of the properties file is `$OMNIHOME/etc/nco_dbinit.props`.

The properties and command line options supported by `nco_dbinit` are described in Table 3.

Table 3: nco_dbinit Properties and Command Line Options (1 of 4)

Property	Command Line Option	Description
<code>ApplicationFile</code> <i>string</i>	<code>-applicationfile</code> <i>string</i>	The application SQL definition file, which creates the default tables for the <code>alerts</code> and <code>tools</code> databases. The default file is <code>\$OMNIHOME/etc/application.sql</code> .
<code>AutomationFile</code> <i>string</i>	<code>-automationfile</code> <i>string</i>	The automation SQL definition file, which creates the default triggers and trigger groups. The default file is <code>\$OMNIHOME/etc/automation.sql</code> .
<code>CopyPropsFile</code> TRUE FALSE	<code>-nopropsfile</code>	Determines whether a new properties file is created for the target ObjectServer. If the <code>-nopropsfile</code> command line option is specified or the <code>CopyPropsFile</code> property is set to <code>FALSE</code> , the default properties file is not copied for the target ObjectServer. The default is <code>TRUE</code> ; that is, the default properties file is copied and renamed for the new ObjectServer.
<code>DesktopFile</code> <i>string</i>	<code>-desktopfile</code> <i>string</i>	The desktop SQL definition file, which inserts default values into the desktop tables, including default colors, conversions, tools, and menus. The default file is <code>\$OMNIHOME/etc/desktop.sql</code> .

Table 3: nco_dbinit Properties and Command Line Options (2 of 4)

Property	Command Line Option	Description
DesktopServer TRUE FALSE	-desktopserver	Indicates that the ObjectServer should be created as a desktop ObjectServer. For information on desktop ObjectServer architectures, see the Netcool/OMNIbus Installation and Deployment Guide.
DesktopServerFile <i>string</i>	-desktopserverfile <i>string</i>	Configures the ObjectServer as a desktop ObjectServer using this SQL definition file, which creates the master.national table for the desktop server and the corresponding MasterSerial column in the alerts.status table. The default file is \$OMNIHOME/etc/desktopserver.sql. For information on desktop ObjectServer architectures, see the Netcool/OMNIbus Installation and Deployment Guide.
Force TRUE FALSE	-force	Forces existing database files to be overwritten. Warning: This option overwrites any existing data. All modifications will be lost.
N/A	-help	Displays help on the command line options and exits.
Memstore.DataDirectory <i>string</i>	-memstoredata <i>string</i>	Specifies the path for the ObjectServer to use for database files. The default is \$OMNIHOME/db.
MessageLevel <i>string</i>	-messagelevel <i>string</i>	Specifies the message logging level. Possible values are: fatal, error, warn, info, and debug. The default level is info. Messages that are logged at each level are listed below: fatal - fatal only. error - fatal and error. warn - fatal, error, and warn. info - fatal, error, warn, and info. debug - fatal, error, warn, info, and debug.

Table 3: nco_dbinit Properties and Command Line Options (3 of 4)

Property	Command Line Option	Description
MessageLog <i>string</i>	-messagelog <i>string</i>	Specifies where messages are logged. The default is <code>stderr</code> .
N/A	-propsfile <i>string</i>	Specifies the path to the <code>nco_dbinit</code> properties file. The default is <code>\$OMNIHOME/etc/nco_dbinit.props</code> .
ObjectServerPropsFile <i>string</i>	-objservpropsfile <i>string</i>	Specifies the path to the source ObjectServer properties file. The default is <code>\$OMNIHOME/install/NCOMS.props</code> .
Props.CheckNames TRUE FALSE	N/A	When <code>TRUE</code> , <code>nco_dbinit</code> does not run if any specified property is invalid. The default is <code>TRUE</code> .
RestrictPasswords TRUE FALSE	-restrictpasswords TRUE FALSE	When <code>TRUE</code> , passwords must conform to the following restrictions: <ul style="list-style-type: none"> The password must consist of at least eight characters. The password must contain at least one numeric character. The password must contain at least one alphabetic character. The default is <code>FALSE</code> .
Sec.AuditLevel <i>string</i>	-secauditlevel <i>string</i>	Specifies the security audit level. The default is <code>warn</code> .
Sec.AuditLog <i>string</i>	-secauditlog <i>string</i>	Specifies the location to log the security audit trail. The default is <code>stdout</code> .
SecurityFile <i>string</i>	-securityfile <i>string</i>	Specifies the path to the security definition file, which defines the operator and administrator roles. The default is <code>\$OMNIHOME/etc/security.sql</code> .
Server <i>string</i>	-server <i>string</i>	Specifies the name of ObjectServer to initialize. The default is <code>NCOMS</code> .

Table 3: nco_dbinit Properties and Command Line Options (4 of 4)

Property	Command Line Option	Description
SystemFile <i>string</i>	-systemfile <i>string</i>	Specifies security database and tables, and system users, groups, roles, and permissions. The default is \$OMNIHOME/etc/system.sql. Warning: Do not modify this file.
N/A	-version	Displays version information for nco_dbinit and exits.

1.3 Starting and Stopping the ObjectServer

This section describes how to start and stop an ObjectServer. You must have an ObjectServer running before you can use the components of Netcool/OMNIBus. You can start and stop an ObjectServer:

- Automatically, using process control (on UNIX) and services (on Windows)
- Manually, from the command line

Starting and Stopping an ObjectServer Using Process Control (UNIX)

If started by the process agent, the ObjectServer automatically restarts if it fails. By starting the process agent when system boots, you can make the ObjectServer start automatically, as described in *Automatically Starting Process Control Agents on Reboot* on page 229.

You can start and stop the ObjectServer from a remote machine. The name you specify in the `-name` option is compared to the process agent names configured in the Server Editor. The host machine and port are identified and the command is sent to the correct process agent.

An ObjectServer can be started and stopped, as a process, using the process control agent. The ObjectServer must be defined as a process or part of a service.

To start an ObjectServer as a process, enter the following command:

```
nco_pa_start -process ObjectServer
```

To stop an ObjectServer as a process, enter the following command:

```
nco_pa_stop -process ObjectServer
```

In these example, the ObjectServer has been defined as a process called `ObjectServer`. For detailed instructions about using process control, see Chapter 4: *Process Control* on page 223.

Starting and Stopping an ObjectServer Using Services (Windows)

On Windows systems, ObjectServer restart capability is provided by services. When set to automatic, the ObjectServer starts when the machine starts up.

To run the ObjectServer as a service, do the following:

1. Click **Start**→**Netcool OMNIBus**→**Stop Running Netcool Services**.
2. Open the Windows Control Panel.
3. Double-click **Administrative Tools** and double-click the **Services** icon.
4. Double-click **NCO Object Server**. The properties window for this service opens.

5. In the **Start Parameters** field, enter the command line options for the local ObjectServer. For example, enter: `-name NCOMS2`
6. Ensure that the **Startup Type** is set to automatic.
7. Click the **Start** button to start the ObjectServer as a Windows service.

Once the service starts, click **OK** to close the properties window.

Starting an ObjectServer Manually

To start the ObjectServer manually from the command line:

Table 4: Starting the ObjectServer from the Command Line

On...	Enter the following command...
UNIX	<code>\$OMNIHOME/bin/nco_objserv [-name <i>servername</i>]</code>
Windows	<code>%OMNIHOME%\bin\nco_objserv [-name <i>servername</i>]</code>

In this command, *servername* is the ObjectServer name. If you do not specify the `-name` command line option, `nco_objserv` attempts to start the NCOMS ObjectServer. For additional command line options, see *Specifying ObjectServer Command Line Options* on page 19.



Note: An ObjectServer started from the command line is not under process control. The ObjectServer must be restarted manually if it is shut down.

On startup, the ObjectServer attempts to open the properties file *servername.props*, where *servername* is the name of the specified ObjectServer. The properties file is located in the `$OMNIHOME/etc` directory.

For information about the properties file, see *Specifying ObjectServer Properties* on page 19.

Configuring a Running ObjectServer

When the ObjectServer is running, the following methods are available to make changes to the configuration:

- Alter system commands, as described in *System and Session SQL Commands* on page 174
- Netcool/OMNIbus Administrator, as described in Chapter 2: *Netcool/OMNIbus Administrator* on page 43

Stopping an ObjectServer Manually

To stop an ObjectServer that has been started manually, use the SQL interactive interface.

1. To connect to an ObjectServer using the SQL interactive interface, do one of the following:

Table 5: Connecting to the ObjectServer

On...	Enter the following command...
UNIX	<code>\$OMNIHOME/bin/nco_sql [-server <i>servername</i>] [-user <i>username</i>]</code>
Windows	<code>%OMNIHOME%\bin\redist\isql.exe -S <i>servername</i> -U <i>username</i></code>

In these commands, *servername* is the name of a local or remote ObjectServer and *username* is a valid user name. If you do not specify the `-server` command line option, the SQL interactive interface connects to the NCOMS ObjectServer. If you do not specify a user name, the default is the user running the command.



Tip: On Windows you must specify the ObjectServer name and user name.

2. Provide the requested password.
3. When the SQL prompt appears, enter the following:

```
1> alter system shutdown;
2> go
```



Note: You must have the appropriate permissions to stop the ObjectServer.

For more information about how to use the SQL interactive interface, including other `nco_sql` command line options, see *Using the SQL Interactive Interface* on page 128.

The `alter system shutdown` command is described in *Managing the ObjectServer: The ALTER SYSTEM Command* on page 174.

If an ObjectServer is started under process control, the process agent restarts it automatically after a manual shutdown. In this case, you must shut down the ObjectServer using process control.

1.4 ObjectServer Properties and Command Line Options

The ObjectServer reads the properties file when it starts. If a property is not specified in this file, the default value is used unless a command line option is used to override it. The default location of the properties file is `$OMNIHOME/etc/servername.props`.

Specifying ObjectServer Command Line Options

Command line options for the ObjectServer use the following format:

```
nco_objserv [ -option [ value ] ... ]
```

In this command, `-option` is the command line option and `value` is the value you are setting the option to. Not every option requires you to specify a value.

On UNIX, you can add command line options to `nco_objserv` commands in the process agent configuration file. See *Defining Processes, Services, and Hosts* on page 231 for additional information.

Specifying ObjectServer Properties

You can change the settings for ObjectServer properties in the SQL interactive interface using the `alter system set` SQL command or by editing the properties file. The `alter system set` command is described in *System and Session SQL Commands* on page 174.

You can query the `catalog.properties` table to view information about ObjectServer properties. For example, to retrieve a list of properties that cannot be modified, enter the query:

```
select PropName from catalog.properties where IsModifyable=FALSE;
```

To retrieve a list of properties that are not modified until the ObjectServer is restarted, enter the query:

```
select PropName from catalog.properties where IsImmediate=FALSE;
```

How Properties and Command Line Options Are Processed at Startup

There are default values for each property. In an unedited properties file, all properties are set to the default values, and are commented out with a hash symbol (#) at the beginning of the line.

If you change a setting in the properties file and remove the hash symbol, this overrides the default.

If you change a setting on the command line, this overrides both the default value and the setting in the properties file. To avoid errors, add as many properties as possible to the properties file rather than using the command line options.

The ObjectServer properties and command line options are described in Table 6.

Table 6: ObjectServer Properties and Command Line Options (1 of 10)

Property	Command Line Option	Description
AlertSecurityModel <i>integer</i>	<code>-alertsecuritymodel <i>integer</i></code>	<p>This property determines whether group row level security is enforced in the event list. By default, group row level security is disabled (0). In this case:</p> <ul style="list-style-type: none"> A member of the <code>Normal</code> group can modify a row that is assigned to themselves or the <code>nobody</code> user. A member of the <code>Administrator</code> group can modify a row that is assigned to themselves, the <code>nobody</code> user, or a member of the <code>Normal</code> group. <p>If the <code>AlertSecurityModel</code> property is enabled (1), only users in the group that owns the row can modify the row. In this case, a member of the <code>Normal</code> or <code>Administrator</code> group can modify a row that is assigned to a group of which they are a member.</p> <p>A member of the <code>System</code> group can always modify any row.</p> <p>For more information on system and default groups, see the Netcool/OMNIBus Installation and Deployment Guide.</p>
AllowConnections TRUE FALSE	<code>-disallowconnections</code>	Specifies whether non-root users can connect to the ObjectServer. If <code>FALSE</code> , no new connections to the ObjectServer are allowed. The default is <code>TRUE</code> .
AllowISQL TRUE FALSE	<code>-disallowisql</code>	Specifies whether connections to the ObjectServer are allowed using the SQL interactive interface. If <code>FALSE</code> , no user can connect using <code>nco_sql</code> . The default is <code>TRUE</code> . If <code>TRUE</code> , this can be enabled for each user using Netcool/OMNIBus Administrator.
AllowISQLwrite TRUE FALSE	<code>-disallowisqlwrite</code>	Specifies whether modifications to the ObjectServer are allowed using the SQL interactive interface. If <code>FALSE</code> , no user can modify the ObjectServer using <code>nco_sql</code> . The default is <code>TRUE</code> . If <code>TRUE</code> , this can be enabled for each user using Netcool/OMNIBus Administrator.

Table 6: ObjectServer Properties and Command Line Options (2 of 10)

Property	Command Line Option	Description
AllowTimedRefresh TRUE FALSE	-allowtimedrefresh TRUE FALSE	This property determines whether the user can enable timed refresh in the Refresh tab of the <i>Event List Preferences</i> window. If TRUE, the event list preferences can be set to allow alert information to be updated at a specified interval rather than waiting for notification of updates from the ObjectServer. The default is FALSE. If FALSE, the timed refresh check box is grayed out in the Refresh tab of the <i>Event List Preferences</i> window and timed refresh is disabled.
Auto.Debug TRUE FALSE	-autodebug TRUE FALSE	If TRUE, automation debugging is enabled. The default is FALSE.
Auto.Enabled TRUE FALSE	-autoenabled TRUE FALSE	If TRUE, automations are enabled. The default is TRUE.
Auto.StatsInterval <i>integer</i>	-autostatsinterval <i>integer</i>	Specifies the interval in seconds at which the automation system collects statistics. The default is 60. Statistics are gathered unless the -autoenabled command line option is set to FALSE, which disables all automations.
Auto.StatsLogfile <i>string</i>	-autostatslogfile <i>string</i>	Specifies the log file that the automation system writes statistics to. By default, the trigger statistics are logged to the file \$OMNIHOME/log/ <i>servername_trigger_stats.logn</i> .

Table 6: ObjectServer Properties and Command Line Options (3 of 10)

Property	Command Line Option	Description
BackupObjectServer TRUE FALSE	-backupserver	Provides failback capability with desktop clients, probes, the proxy server, and the ObjectServer Gateway. The default is FALSE; the desktop clients, probes, the proxy server, and gateways are assumed to be connected to a primary ObjectServer. When TRUE, the desktop clients, probes, the proxy server, and gateways are made aware that they are connected to the backup ObjectServer in a failover pair. If this is the case, the desktop clients, probes, the proxy server, and gateways will automatically check for the recovery of the primary ObjectServer in the failover pair and switch back (fail back) when it has restarted.
Connections <i>integer</i>	-connections <i>integer</i>	Sets the maximum number of available connections for desktop clients, probes, and gateways. The default value is 30. Up to two connections may be used by the system.
DeleteLogFile <i>string</i>	-DELETES <i>string</i>	The path and name of the delete log file, where all delete commands are recorded if delete logging is enabled. By default, deletes are logged to the file <code>\$OMNIHOME/log/servername_deletes_file.logn</code> .
DeleteLogging TRUE FALSE	-DELETE_LOGGING TRUE FALSE	When TRUE, delete logging is enabled. The default is FALSE.
DeleteLogLevel <i>integer</i>	-DELETES_LEVEL <i>integer</i>	The log level determines how much information is sent to the delete log file. Possible settings are: <0—No logging. 0—Client type (application ID; for example, <code>ctisql</code> for <code>nco_sql</code>) and SQL executed. This is the default log level. 1—Time, user ID, client type, and SQL executed.

Table 6: ObjectServer Properties and Command Line Options (4 of 10)

Property	Command Line Option	Description
DeleteLogSize <i>integer</i>	-DELETES_SIZE <i>integer</i>	<p>The maximum size of the delete log file. When the log file <i>servername_deletes_file.log1</i> reaches the specified size, it is renamed <i>servername_deletes_file.log2</i> and a new log file, <i>servername_deletes_file.log1</i>, is created. When the new file reaches its maximum size, the older file is deleted and the process repeats.</p> <p>The output from a single delete command is never split between log files. Therefore log files may be larger than the specified size.</p> <p>The default log file size is 1024 KBytes.</p>
DTMaxTopRows <i>integer</i>	-dtmaxtoprows <i>integer</i>	<p>The maximum number of rows that an administrator can specify when using the View Builder to restrict the number of rows an event list user can view. The default is 100.</p>

Table 6: ObjectServer Properties and Command Line Options (5 of 10)

Property	Command Line Option	Description
GWDeduplication <i>integer</i>	-gwdeduplication <i>integer</i>	<p>This property controls the behavior when there is an attempt to reinsert an existing event in the ObjectServer. Possible values are:</p> <p>0—When set to this value (the default):</p> <ul style="list-style-type: none"> • If the client is not a gateway, the Tally field is incremented. • The LastOccurrence, Summary, and AlertKey fields are updated with the new values and the StateChange and InternalLast fields are updated with the current time. • If the new Severity is greater than 0 (clear) and the old Severity was 0, the alert is also updated with the new Severity value. <p>1—If the client is a gateway, the new event replaces old event.</p> <p>2—If the client is a gateway, the new event is ignored.</p> <p>3—When set to this value:</p> <ul style="list-style-type: none"> • For all clients, the Tally field is incremented. • The LastOccurrence, Summary, and AlertKey fields are updated with the new values and the StateChange and InternalLast fields are updated with the current time. • If the new Severity is greater than 0 (clear) and the old Severity was 0, the alert is also updated with the new Severity value.
Granularity <i>integer</i>	-granularity <i>integer</i>	<p>Controls the update interval, in seconds, of IDUC broadcasts to desktops and gateways. Reducing this value increases the update rate to the clients. The default interval is 60 seconds.</p> <p>For more information, see <i>Client Tool Updates Using IDUC</i> on page 32.</p>

Table 6: ObjectServer Properties and Command Line Options (6 of 10)

Property	Command Line Option	Description
N/A	<code>-help</code>	Displays help on the command line options and exits.
<code>Iduc.ListeningHostname</code> <i>string</i>	<code>-listeninghostname</code> <i>string</i>	<p>Sets a host name for clients to use to locate the ObjectServer. On systems where DNS is used, the name returned by a host machine internally may not be the name by which it is referred to on the network.</p> <p>For example, a machine named <code>sfo</code> may actually be identified on the network as <code>sfo.bigcorp.org</code>. To allow clients to connect to the ObjectServer on <code>sfo</code>, set this property to <code>sfo.bigcorp.org</code>.</p> <p>The default is the name of the local machine, as reported by the <code>hostname</code> command.</p>
<code>Iduc.ListeningPort</code> <i>integer</i>	<code>-listeningport</code> <i>integer</i>	<p>Sets the port for the IDUC communication connection. This is the port on which the ObjectServer sends updates to each event list and gateway. If not specified, the IDUC port is selected by the ObjectServer at random from the unused port numbers available. The default is 0, that is, take the next available port.</p> <p>You can also specify the port in the <code>/etc/services</code> file on the host machine.</p> <p>For more information, see <i>Client Tool Updates Using IDUC</i> on page 32.</p>
<code>Ipc.SSLCertificate</code> <i>string</i>	<code>-sslcertificate</code> <i>string</i>	<p>Specifies the path to the SSL certificate. The default is <code>\$OMNIHOME/etc/servername.crt</code>.</p> <p>For more information on setting up a Netcool system using SSL communications, see the Netcool/OMNIBus Installation and Deployment Guide.</p>
<code>Ipc.SSLEnable</code> TRUE FALSE	<code>-sslenable</code>	<p>Enables SSL communications.</p> <p>For more information on setting up a Netcool system using SSL communications, see the Netcool/OMNIBus Installation and Deployment Guide.</p>

Table 6: ObjectServer Properties and Command Line Options (7 of 10)

Property	Command Line Option	Description
<code>Ipc.SSLPrivateKeyPassword</code> <i>string</i>	<code>-sslprivatekeypassword</code> <i>string</i>	Specifies the SSL private key password. The default is ''. For more information on setting up a Netcool system using SSL communications, see the Netcool/OMNibus Installation and Deployment Guide.
<code>MaxLogFileSize</code> <i>integer</i>	<code>-maxlogfilesize</code> <i>integer</i>	Specifies the maximum size the log file can grow to, in KBytes. The default is 1024 KBytes. Once it reaches the size specified, the <code>servername.log</code> file is renamed <code>servername.log_OLD</code> and a new log file is started. When the new file reaches the maximum size, it is renamed and the process starts again.
<code>MessageLevel</code> <i>string</i>	<code>-messagelevel</code> <i>string</i>	Specifies the message logging level. Possible values are: <code>debug</code> , <code>info</code> , <code>warn</code> , <code>error</code> , and <code>fatal</code> . The default level is <code>warn</code> . Messages that are logged at each level are listed below: <code>fatal</code> - fatal only. <code>error</code> - fatal and error. <code>warn</code> - fatal, error, and warn. <code>info</code> - fatal, error, warn, and info. <code>debug</code> - fatal, error, warn, info, and debug.
<code>MessageLog</code> <i>string</i>	<code>-messagelog</code> <i>string</i>	Specifies the path to which messages are logged. The default is <code>\$OMNIHOME/log/NCOMS.log</code> . On Windows, if the system cannot write to the specified log file (for example, as the result of a fatal error) it writes the error to a file named: <code>%OMNIHOME%\log\nco_objserv_nnnn.err</code> In this file name, <code>nnnn</code> is the process ID (PID) of the ObjectServer process. The ObjectServer must be running as a service; otherwise, it cannot write errors to a file.

Table 6: ObjectServer Properties and Command Line Options (8 of 10)

Property	Command Line Option	Description
N/A	<code>-name string</code>	Sets the ObjectServer name, which must be unique. This is the name that is configured in the Server Editor. The default is <code>NCOMS</code> .
<code>PA.Name string</code>	<code>-pa string</code>	<p>Sets the process control agent name. When an external procedure is run from an automation, the ObjectServer can start an external process. To start the process, the ObjectServer contacts a process control agent. The default name for the process control agent is <code>NCO_PA</code>.</p> <p>This option is only supported on UNIX platforms.</p> <p>For more information on process control, see Chapter 4: <i>Process Control</i> on page 223.</p>
<code>PA.Password string</code>	<code>-papassword string</code>	<p>Specifies the password for the user connecting to a process control agent to run external procedures in automations. The default is <code>' '</code>.</p> <p>This option is only supported on UNIX platforms.</p>
<code>PA.Username string</code>	<code>-pausername string</code>	<p>Specifies the user name for connecting to a process control agent to run external procedures in automations. A value must be specified when the process control agent is running in secure mode. The default is <code>root</code>.</p> <p>This option is only supported on UNIX platforms.</p>
<code>ProfileStatsInterval integer</code>	<code>-profilestatsinterval integer</code>	Specifies the interval in seconds at which the profiler writes information to the profile log file. The default is 60 seconds.
<code>Profile TRUE FALSE</code>	<code>-profile</code>	<p>Controls ObjectServer profiling. If <code>TRUE</code>, the amount of time it takes for clients to execute SQL is logged to the <code>catalog.profiles</code> table, described in <i>Statistics Tables</i> on page 274. The default is <code>FALSE</code>.</p> <p>The profile statistics are also logged to the file <code>\$OMNIHOME/log/servername_profiler_report.logn</code> every profile statistics interval.</p>

Table 6: ObjectServer Properties and Command Line Options (9 of 10)

Property	Command Line Option	Description
Props.CheckNames TRUE FALSE	N/A	When TRUE, the ObjectServer does not run if any specified property is invalid. The default is TRUE.
N/A	-propsfile <i>string</i>	Sets the ObjectServer properties file name. The default name is <i>servername.props</i> , where the <i>servername</i> is defined by the <i>-name</i> option.
RestrictionUpdateCheck TRUE FALSE	-norestrictionupdatecheck	When FALSE, users with restriction filters applied can update alerts that will not appear in their view after the update. The default is FALSE.
RestrictPasswords TRUE FALSE	-restrictpasswords TRUE FALSE	When TRUE, passwords must conform to the following restrictions: <ul style="list-style-type: none"> • The password must consist of at least eight characters. • The password must contain at least one numeric character. • The password must contain at least one alphabetic character. The default is FALSE.
RestrictProxySQL TRUE FALSE	-restrictproxysql	When TRUE, connections from a proxy server are restricted in the ObjectServer SQL commands that can be executed. The only modifications to ObjectServer data that can be made are inserts into the <code>alerts.status</code> and <code>alerts.details</code> tables. The default is FALSE. If FALSE, connections from a proxy server can execute any ObjectServer SQL commands.
Sec.AuditLevel <i>string</i>	-secauditlevel <i>string</i>	Specifies the level of security auditing performed. Possible values are <code>debug</code> , <code>info</code> , <code>warn</code> , and <code>error</code> . The default is <code>warn</code> . The <code>debug</code> and <code>info</code> levels generate messages for authentication successes and failures, while <code>warn</code> and <code>error</code> levels generate messages for authentication failures only.

Table 6: ObjectServer Properties and Command Line Options (10 of 10)

Property	Command Line Option	Description
<code>Sec.AuditLog</code> <i>string</i>	<code>-secauditlog</code> <i>string</i>	Specifies the file to which audit information is written. The default is <code>\$(OMNIHOME)/log/servername/audit.log</code> .
<code>Sec.UsePam</code> TRUE FALSE	<code>-secusepam</code>	If TRUE and enabled for the user, external authorization can be used. The default is TRUE. For more information on PAM, see the Netcool/OMNIBus Installation and Deployment Guide.
<code>SecureMode</code> TRUE FALSE	<code>-secure</code>	Sets the security mode of the ObjectServer. If TRUE, the ObjectServer authenticates probe, gateway, and proxy server connection requests with a user name and encrypted password. Other client connection requests are always authenticated with a user name and password. For more information about using secure mode, see <i>Running the ObjectServer in Secure Mode</i> on page 30.
N/A	<code>-version</code>	Displays version information for the ObjectServer and exits.

1.5 Running the ObjectServer in Secure Mode

You can run the ObjectServer in secure mode. When you specify the `-secure` command line option, the ObjectServer authenticates probe, gateway, and proxy server connections by requiring a user name and encrypted password. When a connection request is sent, the ObjectServer issues an authentication message. The probe, gateway, or proxy server must respond with the correct user name and password combination.

If you do not specify the `-secure` option, probe, gateway, and proxy server connection requests are not authenticated.



Tip: Connections from other clients, such as the event list and SQL interactive interface, are always authenticated.

When connecting to a secure ObjectServer, each probe and proxy server that makes a connection must have the `AuthUserName` and `AuthPassword` properties specified in its properties file. Each gateway must have the `AUTH_USER` and `AUTH_PASSWORD` commands in the gateway configuration file. If the user name and password combination is incorrect, the ObjectServer issues an error message and rejects the connection.

You can choose any valid user name for the `AuthUserName` property or `AUTH_USER` gateway command. To generate the encrypted password, use the `nco_g_crypt` utility.

Table 7: Running the `nco_g_crypt` Encryption Utility

On...	Enter the following command...
UNIX	<code>\$OMNIHOME/bin/nco_g_crypt plaintext_password</code>
Windows	<code>%OMNIHOME%\bin\nco_g_crypt plaintext_password</code>

The `nco_g_crypt` utility takes the unencrypted password and displays the encrypted password to be entered for the `AuthPassword` property or `AUTH_PASSWORD` gateway command.

An encrypted password is specified in the same way as an unencrypted password when connecting to the ObjectServer. The ObjectServer automatically detects an encrypted password and performs the necessary decryption to verify the password during authentication.



Warning: Encrypted passwords can be used in the same way as unencrypted passwords to access the ObjectServer. Therefore, you must set permissions on any files containing encrypted passwords appropriately to prevent unauthorized access.

For information on running probes and gateways in secure mode, see the Netcool/OMNIBus Probe and Gateway Guide. For information on running proxy servers in secure mode, see *Running the Proxy Server in Secure Mode* on page 38.

1.6 Client Tool Updates Using IDUC

A large quantity of data passes between the ObjectServer and a desktop client each time an event list is updated. To prevent the ObjectServer from becoming overloaded with requests for event list updates, the ObjectServer sends a prompt to the desktop client each time an update is needed. The desktop then requests the updated data from the ObjectServer.

This prompt is sent to the desktop through a communication link that uses an Insert, Delete, Update, or Control (IDUC) communication protocol. The prompt instructs the desktop to refresh all of the event list displays. The IDUC protocol updates gateways in the same way.

The desktop client connects to the ObjectServer using the port defined by the interfaces file to establish the IDUC communication link. The desktop receives the socket number of the IDUC connection on which it will receive the ObjectServer prompts for the updates.

Specifying the IDUC Update Interval

The update interval is controlled by the ObjectServer `Granularity` property or `-granularity` command line option, which is set to 60 seconds by default. The default value is optimal for most systems. Reducing it improves the response time of the client tools but greatly increases network traffic and ObjectServer load.

Specifying the IDUC Port

By default, when an ObjectServer starts an available port number is chosen for the IDUC connection. You can also specify the IDUC port to use. You *must* specify the IDUC port when accessing an ObjectServer protected by a firewall.

On UNIX, you can define these ports by updating the `/etc/services` file. On Windows, you update the `%SystemRoot%\system32\drivers\etc\services` file.

The `services` file has an entry for each ObjectServer in the following format:

```
nco_servername nnnn/tcp
```

In this entry, `servername` is the name of the ObjectServer and `nnnn` is the port number.

The following are example entries for ObjectServers named NCOMS and Denco:

```
nco_NCOMS 7070/tcp  
nco_DENCO 7071/tcp
```

The port can be set to any unused number outside the range from 1 to 1024, which are generally reserved as system numbers.

When the `/etc/services` file is managed by Network Information Service (NIS), you must make the entry in the NIS services file and then copy the updated configuration to all machines.

You can also use the `-listeningport` option on the `ObjectServer` command line to specify the IDUC port.

1.7 Starting the Proxy Server

For introductory information about the proxy server, see the Netcool/OMNIbus Installation and Deployment Guide.

Use the `nco_proxyserv` command to start the proxy server.

Table 8: Starting the proxy server from the Command Line

On...	Enter the following command...
UNIX	<code>\$OMNIHOME/bin/nco_proxyserv [-name proxyname] [-server servername]</code>
Windows	<code>%OMNIHOME%\bin\nco_proxyserv [-name proxyname] [-server servername]</code>

In these commands, *proxyname* is the name of the proxy server and *servername* is the name of the ObjectServer. If you do not specify the `-name` command line option, the `NCO_PROXY` proxy server is started. If you do not specify the `-server` command line option, the proxy server buffers connections for the `NCOMS` ObjectServer.

For additional properties and command line options, see *Proxy Server Properties and Command Line Options* on page 35.



Note: The recommended method for starting a proxy server is using process control. For more information, see Chapter 4: *Process Control* on page 223.

Connecting to the Proxy Server

To ensure that probes connect to an ObjectServer through the proxy server, supply the proxy server name in the `Server` property in the probe properties file or use the `-server` command line option.

All alerts are then sent to the proxy server.

1.8 Proxy Server Properties and Command Line Options

The proxy server reads its properties file when it starts. If a property is not specified in this file, the default value is used unless a command line option is used to override it. The default location of the properties file is `$OMNIHOME/etc/proxyserver.props`.

Command line options for the proxy server use the following format:

```
nco_proxyserv [ -option [ value ] ... ]
```

In this command, *-option* is the command line option and *value* is the value you are setting the option to. Not every option requires you to specify a value.

If you do not specify a properties file when starting an proxy server, the default file is used. Use the `-propsfile` command line option to specify the full path and file name of an alternate properties file.

Table 9 lists proxy server properties and command line options.

Table 9: Proxy Server Command Line Options (1 of 3)

Property	Command Line Option	Description
<code>AuthPassword string</code>	N/A	The password associated with the user name used to authenticate the proxy server when it connects to an ObjectServer running in secure mode. This password must be encrypted with the <code>nco_g_crypt</code> utility. The default is <code>' '</code> . For more information about using secure mode, see <i>Running the Proxy Server in Secure Mode</i> on page 38.
<code>AuthUserName string</code>	N/A	The user name used to authenticate the proxy server when it connects to an ObjectServer running in secure mode. The default is <code>root</code> . For more information about using secure mode, see <i>Running the Proxy Server in Secure Mode</i> on page 38.
<code>ConnectionRatio integer</code>	<code>-ratio integer</code>	Sets the ratio of incoming connections from probes to outgoing connections to an ObjectServer. The default value of 10 creates a 10:1 ratio of incoming to outgoing connections.
N/A	<code>-help</code>	Displays the supported command line options and exits.

Table 9: Proxy Server Command Line Options (2 of 3)

Property	Command Line Option	Description
SSLCertificate <i>string</i>	N/A	Specifies the path to the SSL certificate. The default is <code>\$OMNIHOME/etc/servername.crt</code> . For more information on setting up a Netcool system using SSL communications, see the Netcool/OMNIBus Installation and Deployment Guide.
SSLEnable TRUE FALSE	N/A	Enables SSL communications. For more information on setting up a Netcool system using SSL communications, see the Netcool/OMNIBus Installation and Deployment Guide.
SSLPrivateKeyPassword <i>string</i>	N/A	Specifies the SSL private key password. The default is <code>' '</code> . For more information on setting up a Netcool system using SSL communications, see the Netcool/OMNIBus Installation and Deployment Guide.
N/A	<code>-logfile <i>string</i></code>	Sets the name of the file to which the proxy server writes messages, including errors. By default, the file is <code>\$OMNIHOME/log/servername.log</code> , where the <i>servername</i> is defined by the <code>-name</code> option.
MaxConnections <i>integer</i>	<code>-max <i>integer</i></code>	Sets the maximum number of available connections for probes. The minimum (and default) value is 30, and the maximum value is 250.
NetworkTimeout <i>integer</i>	<code>-networktimeout <i>integer</i></code>	Specifies a time in seconds after which a login attempt or connection to the ObjectServer will time out, should a network failure occur. After the specified timeout period, the proxy server attempts to reconnect to the ObjectServer. If the connection is unsuccessful after a second timeout period, the proxy server will attempt to connect to a backup ObjectServer, where available. The default is 20 seconds.
N/A	<code>-propsfile <i>string</i></code>	Sets the proxy server properties file name. The default name is <code>\$OMNIHOME/etc/servername.props</code> , where the <i>servername</i> is defined by the <code>-name</code> option.
RemoteServer <i>string</i>	<code>-server <i>string</i></code>	Sets the name of the ObjectServer to which the proxy server connects. The default is <code>NCOMS</code> .

Table 9: Proxy Server Command Line Options (3 of 3)

Property	Command Line Option	Description
SecureMode TRUE FALSE	-secure	Set the security mode of the proxy server. If enabled, the proxy server authenticates probe connection requests with a user name and encrypted password. If disabled (the default), probes can connect to the proxy server without a user name and password. For more information about using secure mode, see <i>Running the Proxy Server in Secure Mode</i> on page 38.
ServerName <i>string</i>	-name <i>string</i>	Sets the proxy server name. This is the name that is configured in the Server Editor. The default is NCO_PROXY.
N/A	-version	Displays version information about the proxy server and exits.

1.9 Running the Proxy Server in Secure Mode

You can run the proxy server in secure mode. When you specify the `SecureMode` property or the `-secure` command line option, the proxy server authenticates probe connections by requiring a user name and encrypted password. When a connection request is sent, the proxy server issues an authentication message. The probe must respond with the correct user name and password.

If you do not specify the `-secure` option, probe connection requests are not authenticated.

When connecting to a secure proxy server, each probe must have an `AuthUserName` property and `AuthPassword` property specified in its properties file. If the user name and password combination is incorrect, the proxy server issues an error message.

You can choose any valid user name for the `AuthUserName` property. To generate the encrypted password, use the `nco_g_crypt` utility.

Table 10: Running the `nco_g_crypt` Encryption Utility

On...	Enter the following command...
UNIX	<code>\$OMNIHOME/bin/nco_g_crypt plaintext_password</code>
Windows	<code>%OMNIHOME%\bin\nco_g_crypt plaintext_password</code>

The `nco_g_crypt` utility takes the unencrypted password and displays the encrypted password to be entered for the `AuthPassword` property.

In addition, if the ObjectServer is running in secure mode, the proxy server must have the `AuthUserName` and `AuthPassword` properties in its property file to connect the ObjectServer. This `AuthPassword` is also generated using the `nco_g_crypt` utility. If the user name and password combination is incorrect, the ObjectServer issues an error message.



Warning: Encrypted passwords can be used in the same way as unencrypted passwords to access the ObjectServer. Therefore, you must set permissions on any files containing encrypted passwords appropriately to prevent unauthorized access.

1.10 Checkpointing

ObjectServer data is stored in memory for high speed access. The ObjectServer supports data persistence using checkpoints and logs to copy the data in memory to disk. This enables you to recover the data after a planned or unexpected shutdown occurs.

The following types of files are maintained on disk:

- Checkpoint files, containing the data for entire tables.
- Replay logs, containing the changes made to tables since the last checkpoint.

Every 60 seconds, a checkpoint occurs, and all persistent data is copied to checkpoint files. Between checkpoints, new and changed data is written to replay log files.

Checkpoints also occur each time there is a planned ObjectServer shutdown.

Checkpoint File Storage

Checkpoint files are generated for each persistent memstore. Memstores are containers that are maintained by the ObjectServer and hold ObjectServer tables and data in memory. The ObjectServer uses the memstores described in Table 11.

Table 11: Memstores

Memstore Name	Storage Type	Description
MASTER_STORE	Persistent	Used to store internal descriptions of ObjectServer data.
TABLE_STORE	Persistent	Used to store information for the desktop, including the <code>alerts.status</code> table.
TEMP_STORE	Transient	Used to store data that does not need to be persistent. System tables that contain configuration information are stored here and recreated on startup.
VIRTUAL_STORE	Virtual	Used mainly to catalog rapidly changing data, for example, data about connected clients.



Tip: Only persistent memstores are checkpointed.

Checkpoint files are named:

```
$OMNIHOME/db/servername/storename.chk0
$OMNIHOME/db/servername/storename.chk1
```

Replay logs are also generated for each persistent memstore. The replay files are named:

```
$OMNIHOME/db/servername/storename.log0  
$OMNIHOME/db/servername/storename.log1
```

The checkpoint process writes alternately to the `.chk0` and `.chk1` files. If one file has become corrupted during an unexpected shutdown, the data in the other checkpoint file and the replay logs is used to rebuild the database tables in memory. As each checkpoint starts, the logging process switches to the alternate log file. The older log file is deleted before the start of the next checkpoint.

When a planned ObjectServer shutdown occurs (because the `alter system shutdown` command is executed), a `.tab` file is created for each persistent memstore, named:

```
storename.tab
```

For example, the `.tab` file for the master store in the NCOMS ObjectServer is named:

```
$OMNIHOME/db/NCOMS/MASTER_STORE.tab
```

The format of these files is specific to the hardware and operating system platform on which they were created.

Data Recovery During ObjectServer Startup

When the ObjectServer is restarted after a planned shutdown, the database is rebuilt using the `.tab` files.

When the ObjectServer is restarted after an unexpected shutdown, the database is rebuilt using the checkpoint (`.chk`) and replay log (`.log`) files.

Checkpoint Verification Utility

The `nco_check_store` utility verifies that existing checkpoint files are valid. It is intended to be used by automations and reports the validity with a return code. The return codes are:

- 0—Success. The checkpoint files are valid.
- 1—Failure. The checkpoint files are not valid and should not be used.

You can also use the `nco_check_store` utility from the command line. When invoked from the command line, you must set the message logging level to `info` to display the progress and results of the test. The command line options are described in Table 12.

Table 12: Checkpoint Verification Utility Command Line Options (1 of 2)

Command Line Option	Description
<code>-help</code>	Displays help on the command line options and exits.

Table 12: Checkpoint Verification Utility Command Line Options (2 of 2)

Command Line Option	Description
<code>-memstoredatadirectory <i>string</i></code>	Specifies the path to the ObjectServer database. The default is <code>\$OMNIHOME/db</code> .
<code>-messagelevel <i>string</i></code>	Specifies the message logging level. Possible values are: <code>debug</code> , <code>info</code> , <code>warn</code> , <code>error</code> , and <code>fatal</code> . The default level is <code>warn</code> . Messages that are logged at each level are listed below: <code>fatal</code> - <code>fatal</code> only. <code>error</code> - <code>fatal</code> and <code>error</code> . <code>warn</code> - <code>fatal</code> , <code>error</code> , and <code>warn</code> . <code>info</code> - <code>fatal</code> , <code>error</code> , <code>warn</code> , and <code>info</code> . <code>debug</code> - <code>fatal</code> , <code>error</code> , <code>warn</code> , <code>info</code> , and <code>debug</code> .
<code>-messagelog <i>string</i></code>	Specifies the path to which messages are logged. The default is <code>stderr</code> .
<code>-server <i>string</i></code>	Specifies the name of ObjectServer database to verify. The default is <code>NCOMS</code> .
<code>-version</code>	Displays version information on the checkpoint verification utility and exits.

Chapter 2: Netcool/OMNIbus Administrator

This chapter contains information about using Netcool/OMNIbus Administrator to configure and manage ObjectServers.

It contains the following sections:

- *Introduction to Netcool/OMNIbus Administrator* on page 44
- *Starting Netcool/OMNIbus Administrator and Connecting to an ObjectServer* on page 45
- *Logging out of Netcool/OMNIbus Administrator* on page 49
- *Netcool/OMNIbus Administrator Overview* on page 50
- *Managing Roles, Groups, and Users* on page 55
- *Configuring ObjectServer Objects* on page 68
- *Accessing an ObjectServer Using the SQL Interactive Interface (iSQL)* on page 119

2.1 Introduction to Netcool/OMNIBus Administrator

Netcool/OMNIBus Administrator provides a simple user interface from which you can manage your ObjectServers. This chapter describes how to start and log in to Netcool/OMNIBus Administrator, and how to use Netcool/OMNIBus Administrator to configure ObjectServers. You can also use Netcool/OMNIBus Administrator's SQL Interactive Interface (iSQL) to directly access and manipulate ObjectServer objects and data.

Prerequisites

Netcool/OMNIBus Administrator requires Java Runtime Environment (JRE) version 1.4.1 to be installed on your system. For more information, see the Netcool/OMNIBus Installation and Deployment Guide.

Additionally, Netcool/OMNIBus Administrator requires a license.

You can enable licensing using the `NETCOOL_LICENSE_FILE` environment variable, described in the licensing chapter of the Netcool/OMNIBus Installation and Deployment Guide. You can override this setting using the `license.file` property. For information on the `license.file` property, see Table 13 *nco_config and nco_config.bat Properties and Command Line Options* on page 46.

2.2 Starting Netcool/OMNIBus Administrator and Connecting to an ObjectServer

This section describes how to start Netcool/OMNIBus Administrator on UNIX and Windows platforms. It also contains information about Netcool/OMNIBus Administrator properties and command line options.

On UNIX

To start Netcool/OMNIBus Administrator and log in to a running ObjectServer on a UNIX platform:

1. Enter the following on the command line:

```
nco_config
```

The *Login Required* window appears. For `nco_config` command line options, see *Netcool/OMNIBus Administrator Properties and Command Line Options* on page 46.

2. Enter your ObjectServer user name and password.
3. Click **Login**. The Netcool/OMNIBus Administrator window appears.

To start using Netcool/OMNIBus Administrator, see *Netcool/OMNIBus Administrator Overview* on page 50.

On Windows

To start Netcool/OMNIBus Administrator and log in to a running ObjectServer on Windows:

1. Do one of the following:
 - Select **Start**→**Programs**→**Netcool OMNIBus**→**Administrator Config**.
 - Enter the following on the command line:

```
nco_config.bat
```

The *Login Required* window appears. For `nco_config.bat` command line options, see *Netcool/OMNIBus Administrator Properties and Command Line Options* on page 46.

2. Enter your ObjectServer user name and password.
3. Click **Login**. The Netcool/OMNIBus Administrator window appears.

To start using Netcool/OMNIBus Administrator, see *Netcool/OMNIBus Administrator Overview* on page 50.

Netcool/OMNIbus Administrator Properties and Command Line Options

This section contains information about Netcool/OMNIbus Administrator properties and command line options.

The default Netcool/OMNIbus Administrator properties file is `$OMNIHOME/etc/nco_config.props`.

You can use the properties file as a template and modify it for different purposes. For example, you may use different properties files for logging into different ObjectServers.

The default properties file is read each time you start Netcool/OMNIbus Administrator; however, you can use the `-propsfile` command line option to specify an alternate properties file.

The properties and command line options for `nco_config` and `nco_config.bat` are described in Table 13.

Table 13: `nco_config` and `nco_config.bat` Properties and Command Line Options (1 of 3)

Property	Command Line Option	Description
<code>audit.file.max.size</code> <i>integer</i>	<code>-auditfilesize</code> <i>integer</i>	The maximum file size (in Bytes) for the Netcool/OMNIbus Administrator audit file. The default is 10000.
<code>audit.file.name</code> <i>string</i>	<code>-auditfile</code> <i>string</i>	The full path to the Netcool/OMNIbus Administrator audit file. The default is <code>OMNIHOME/log/nco_config_audit.log</code> .
N/A	<code>-help</code>	Displays help for this command and then exits.
<code>java.security.policy</code> <i>string</i>	<code>-policyfile</code> <i>string</i>	The full path to the Java security policy file.

Table 13: nco_config and nco_config.bat Properties and Command Line Options (2 of 3)

Property	Command Line Option	Description
<code>license.file</code> <i>string</i>	<code>-license</code> <i>string</i>	<p>The host name of the license server, and the port number on which the license server listens for requests.</p> <p>Separate multiple license hosts and ports with ampersands (&). For example:</p> <p><code>27000@myhost1&27000@myhost2&27000@myhost3</code></p> <p>If you are using license hosts in a quorum configuration, separate the quorum servers using commas. For example:</p> <p><code>27000@myhost1,27000@myhost2,27000@myhost3</code></p> <p>Note: Setting this property overrides the <code>NETCOOL_LICENSE_FILE</code> environment variable. If you specify more than one license server, you must use the notation specified above; the notation for multiple license servers described in the Netcool/OMNIBus Installation and Deployment Guide does not work for Netcool/OMNIBus Administrator.</p> <p>The default is <code>27000@localhost</code>.</p>
<code>log.file.max.size</code> <i>integer</i>	<code>-logfilesize</code> <i>integer</i>	<p>The maximum file size (in Bytes) for the Netcool/OMNIBus Administrator log file.</p> <p>The default is 10000.</p>
<code>log.file.name</code> <i>string</i>	<code>-logfile</code> <i>string</i>	<p>The full path to the Netcool/OMNIBus Administrator log file.</p> <p>The default is <code>OMNIHOME/log/nco_config_system.log</code>.</p>
<code>look.and.feel</code> <i>string</i>	<code>-lnf</code> <i>string</i>	<p>The look-and-feel for the Netcool/OMNIBus Administrator user interface. Valid values are:</p> <p>METAL</p> <p>MOTIF</p> <p>SYSTEM (the default)</p> <p>WINDOWS</p>
<code>nco_jdbc.timeout</code> <i>integer</i>	<code>-jdbctimeout</code> <i>integer</i>	<p>The Java Database Connectivity (JDBC) timeout (in seconds).</p> <p>The default is 600.</p>

Table 13: nco_config and nco_config.bat Properties and Command Line Options (3 of 3)

Property	Command Line Option	Description
<code>omni.home string</code>	<code>-omnihome string</code>	The full path to the Netcool/OMNIBus installation directory. Note: Setting this property overrides the <code>OMNIHOME</code> environment variable. The default is <code>/opt/netcool/omnibus</code> .
N/A	<code>-propsfile string</code>	The full path to the Netcool/OMNIBus Administrator properties file. The default is <code>OMNIHOME/etc/nco_config.props</code> .
<code>server string</code>	<code>-server string</code>	The name of the ObjectServer to which you are connecting. The default is <code>NCOMS</code> .
<code>user.name string</code>	<code>-user string</code>	The Netcool/OMNIBus Administrator login user name. The default is <code>root</code> .
<code>user.password string</code>	<code>-password string</code>	The Netcool/OMNIBus Administrator login password.
N/A	<code>-version</code>	Displays version information and then exits.

Property and Command Line Processing

Each property has a default value. In an unedited properties file, all properties are listed with their default values, commented out with a hash symbol (#) at the beginning of the line. A property and its corresponding value are separated by a colon (:). String values are surrounded by single, straight quotes.

You can edit the property values using a text editor. To override the default, change a setting in the properties file and remove the hash symbol.

If you change a setting on the command line, this overrides both the default value and the setting in the properties file. To simplify the command you type to run `nco_config`, add as many properties as possible to the properties file rather than using the command line options.

2.3 Logging out of Netcool/OMNIbus Administrator

To log out of Netcool/OMNIbus Administrator, select **File**→**Logout**.

2.4 Netcool/OMNibus Administrator Overview

Netcool/OMNibus Administrator is a visual interface for the management of ObjectServer objects. An example Netcool/OMNibus Administrator main window is shown in Figure 1.

The pane in the center of the window is the work area where you view, modify, and manage ObjectServer objects.

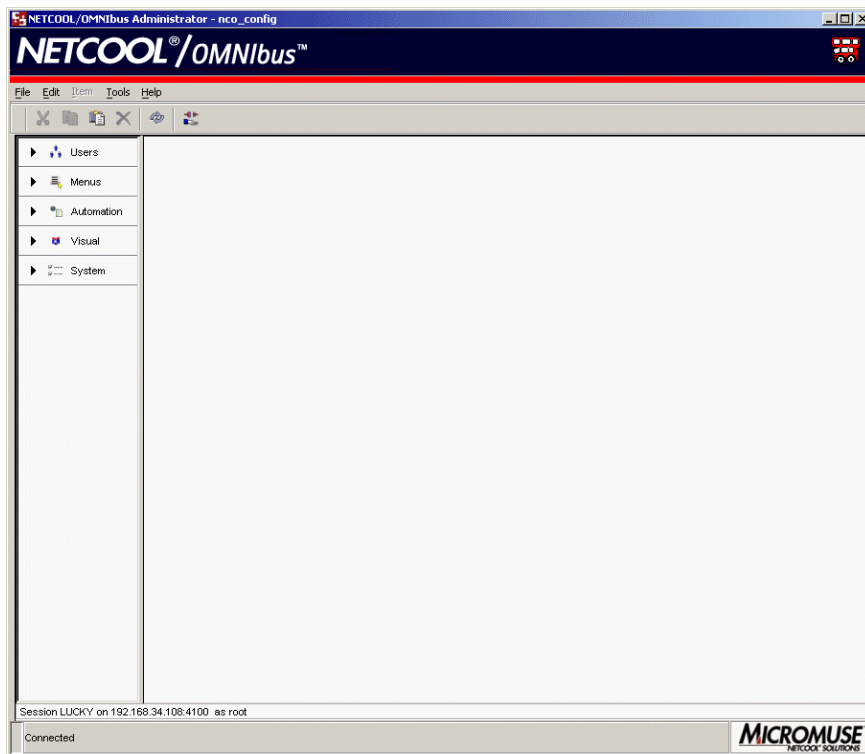


Figure 1: Netcool/OMNibus Administrator Main Window

Secure Sockets Layer Connections

If Netcool/OMNibus Administrator is using an encrypted SSL connection to an ObjectServer, a lock icon will appear in the bottom, left corner of the window, as shown in Figure 2.



Figure 2: Lock Graphic Indicating an SSL Connection to an ObjectServer

For information about using SSL with Netcool/OMNIbus, see the Netcool/OMNIbus Installation and Deployment Guide.

Selecting ObjectServer Objects

Use the drop-down lists on the left side of the Netcool/OMNIbus Administrator window to select the ObjectServer object to configure, as described in Table 14.

Table 14: Selecting ObjectServer Objects in Netcool/OMNIbus Administrator

Drop-down List	Object	See...
Users	Users	page 63
	Groups	page 58
	Roles	page 55
	Restriction filters	page 69
Menus	Menus	page 71
	Tools	page 75
	Prompts	page 79
Automation	Trigger groups	page 84
	Triggers	page 83
	Procedures	page 91
	User defined signals	page 111
Visual	Conversions	page 96
	Colors	page 98
	Column visuals	page 100
	Classes	page 102
System	Properties	page 104
	Databases	page 106
	Log files	page 115
	SQL	page 119

Using Netcool/OMNIBus Administrator Windows

This section contains information about:

- Sorting window columns
- Selecting window columns to display
- Cutting, copying, and pasting

Sorting Window Columns

Many Netcool/OMNIBus Administrator windows display information in columns of data. Click the column title to sort the information by that column. Click multiple times to select either an ascending or descending sort, indicated by the up or down arrow next to the column name.

Figure 3 shows a column sorted in descending order by name.

Name ▾	Enabled
Acknowledged Action	true
Add to Task List	true
Assign Action	true
Change Severity	true
Deacknowledged Action	true

Figure 3: Sorting Window Columns

Selecting Columns to Display

You can select which columns of data to display in some Netcool/OMNIbus Administrator windows. To select the columns of data to display:

1. Right-click the **Rows** status bar at the bottom of the Netcool/OMNIbus Administrator window. The *Column Settings* window is displayed.
2. Click a column name to select or deselect it. A check is displayed next to viewable columns, as shown in Figure 4.

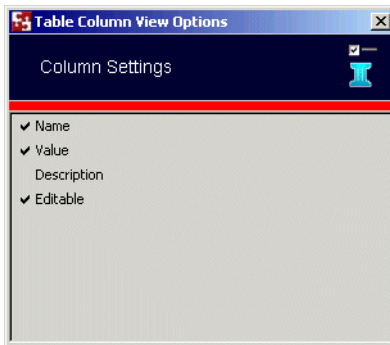


Figure 4: Selecting Columns to Display

Cutting, Copying, and Pasting

You can use the **Edit** menu **Cut**, **Copy**, and **Paste** functions on some types of information. For example, you can copy an existing ObjectServer tool, paste it, and then modify it to create a new tool.

You can copy and paste the following items:

- Menus and menu items
- Restriction filters
- Tools
- Prompts
- Triggers
- Procedures
- Users (in the same ObjectServer only)

Configuring Editor Syntax Coloring

You can enter SQL syntax on some Netcool/OMNIbus Administrator windows (for example, when creating or editing triggers). By default, Netcool/OMNIbus Administrator provides coloring to highlight different keywords and data types. You can change the default syntax coloring scheme.

To change the default syntax coloring scheme:

1. Select **Tools**→**Editor Options**. The *Highlight Details* window is displayed.
2. Use the window items described in Table 15 to change the syntax coloring scheme.

Table 15: Highlight Details Window Items

Window item	Description
Element	Select the data type for which you want to view or change the color.
Color	Displays the currently selected color for this data type. Click the button next to this field to select a new color. You can choose the color using its swatch, HSB, or RGB values.
Use Default	Select this check box to use the default color for the selected data type.

Selecting a Web Browser for Displaying Online Help

To select the web browser for displaying online help:

1. Select **Tools**→**Configure Tools**. The *Choose Tool* window is displayed.
2. Select **Browser**. The *External Program* window is displayed.
3. In the **Executable** field, enter the full path and name of the browser executable, or click the button to the right of the field to select the browser executable.
4. In the **Arguments** field, enter any command line arguments to pass to this executable.

Note: The **Run time environment** drop-down list is not used to select the web browser.

5. Click **OK**. The selected browser will be used to display online help.



2.5 Managing Roles, Groups, and Users

Authorization is the verification of the rights to view and modify information. For example, you may be authorized to create an automation in the ObjectServer for the London NOC, but not for the ObjectServer in the New York NOC.

Each ObjectServer object has actions associated with it. The ObjectServer stores information about the actions each user is and is not authorized to perform on the system and for each object.

Administrators can allow and deny actions on the system and for individual objects by assigning permissions to roles and granting and revoking roles for appropriate groups of users. You can use Netcool/OMNIbus Administrator to grant and revoke permissions for all users of a Netcool/OMNIbus system.

You should set up Netcool/OMNIbus authorization by configuring security objects in the following order:

1. **Roles**, to assign permissions to roles
2. **Groups**, to assign roles to groups
3. **Users**, to add users to groups

Configuring Roles

Roles are collections of permissions that you can assign to a group. Roles determine the types of tasks users in the group can perform. For example, if you assign the `AlertsOperator` role to a group, users in that group can view, update, and delete entries in the `alerts.status` table, and insert entries in the `alerts.journals` table. This collection of permissions allows the members of that group to use the event list.

For detailed information about default roles, see the Netcool/OMNIbus Installation and Deployment Guide.

For information about configuring users, see *Configuring Users* on page 63. For information about configuring groups, see *Configuring Groups* on page 58.

To configure roles, click the **Users** drop-down list; then, click the **Roles** icon. The *Roles* window contains a list of the roles in the selected ObjectServer. An example is shown in Figure 5.

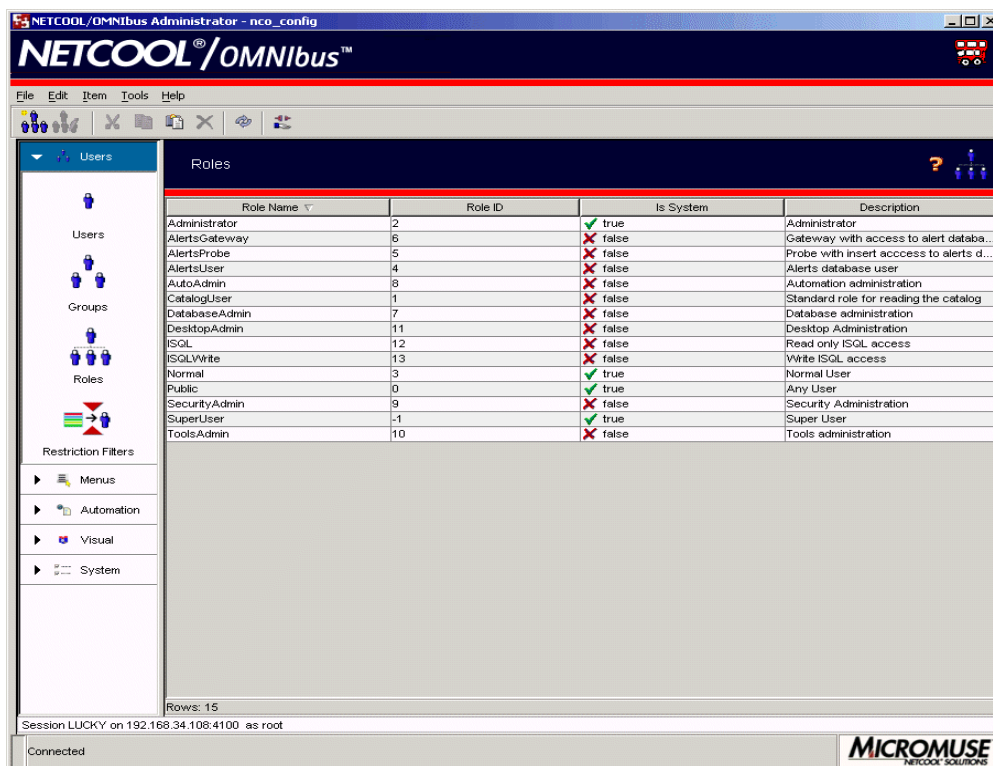


Figure 5: Netcool/OMNIBus Administrator Window - Roles

Creating and Editing Roles

To create or edit a role:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Roles** icon. The *Roles* window is displayed.
3. Click the **Add Role** button, or select the role to edit and click the **Edit Role** button. The *Role Details* window is displayed.

4. Complete or edit the following window items and click **OK**.

Table 16: Role Details Window Items

Window Item	Description
Role Name	Role names are restricted to 64 characters and can include spaces. If you are editing a role, you cannot change the name.
Role ID	Select a unique ID for this role. If you are editing a role, you cannot change the role ID.
Details tab	An optional text description for this role.
Permissions tab	Add and delete role permissions as described next.

Adding and Removing Role Permissions

The *Role Details* window **Permissions** tab displays all permissions currently assigned to a role. You can use this tab to add and remove role permissions.

To add permissions to a role:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Roles** icon. The *Roles* window is displayed.
3. Select the role that you are adding permissions to and click the **Edit Role** button. The *Role Details* window is displayed.
4. Click the **Permissions** tab.
5. Click the **Add Permission** button. The *Permission Objects* window is displayed.
6. Select the object type for which to grant permissions.
7. In the **Objects** list, select the specific object for which to grant permissions.
8. Click **OK**. The *Role Details* window is displayed with the selected permissions.
9. Repeat steps 5-8 to add more permissions.
10. Click **OK** to save the permissions for this role.

To remove permissions from a role:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Roles** icon. The *Roles* window is displayed.
3. Select the role for that you are removing permissions from and click the **Edit Role** button. The *Role Details* window is displayed.

4. Click the **Permissions** tab.
5. To remove a permission from this role, select the permission and click the **Delete permission** button.
6. Repeat step 5 to remove more permissions.
7. Click **OK** to save the permissions for this role.

Deleting Roles

To delete a role:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Roles** icon. The *Roles* window is displayed.
3. Select the role to delete and click the **Delete** button. The role is deleted. If this role has been assigned to a group, the role is removed from the group.

Configuring Groups

Use groups to organize Netcool/OMNIBus users. All members of a group have the permissions assigned to the group's roles.



Tip: The default groups *Normal*, *Administrator*, and *System* provide group row level security in the event list. These groups cannot be deleted or renamed. For detailed information about these and other default groups, see the Netcool/OMNIBus Installation and Deployment Guide.

For information about configuring users, see *Configuring Users* on page 63. For information about configuring roles, see *Configuring Roles* on page 55.

To configure groups, click the **Users** drop-down list; then, click the **Groups** icon. The *Groups* window contains a list of the groups in the selected ObjectServer. An example is shown in Figure 6.

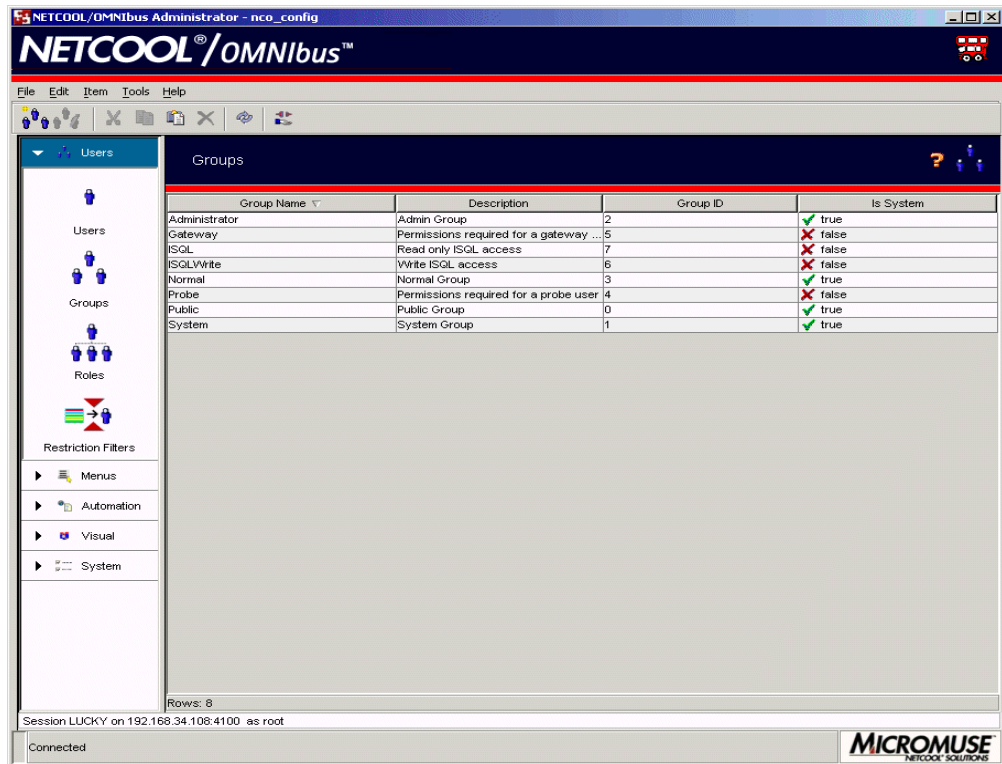


Figure 6: Netcool/OMNIbus Administrator Window - Groups

Creating and Editing Groups

To create or edit a group:

1. On the Netcool/OMNIbus Administrator window, click the **Users** drop-down list.
2. Click the **Groups** icon. The *Groups* window is displayed.
3. Click the **Add Group** button, or select the group to edit and click the **Edit Group** button. The *Group Details* window is displayed.

4. Complete or edit the following window items and click **OK**.

Table 17: Group Details Window Items

Window Item	Description
Group Name	Group names are restricted to 64 characters and can include spaces. If you are editing a group, you cannot change the name.
Group ID	Select a unique ID for this group. If you are editing this group, you cannot change the group ID.
Description	An optional text description for this group.
Roles tab	<p>The Available Roles list displays the roles you can assign to this group. The Applied Roles list displays the roles that are already assigned to this group. Use the arrow buttons to:</p> <ul style="list-style-type: none"> >> Assign all roles to this group. > Assign the role selected in the Available Roles list to the group. < Remove the selected role in the Applied Roles list from this group. << Remove all roles from this group. <p>You can also click the Add new role button to create a role. For information about creating roles, see <i>Creating and Editing Roles</i> on page 56.</p>
Restriction Filters tab	<p>You can optionally select restriction filters to apply to this group. Restriction filters enable you to prevent groups from viewing or modifying certain rows in ObjectServer tables.</p> <p>The All Restriction Filters list displays the available restriction filters. The Assigned Restriction Filters list displays the restriction filters that are assigned to this group.</p> <p>Use the Add Restriction Filter and Remove Restriction Filter buttons to add and remove restriction filters for this group.</p> <p>For information about restriction filters, see <i>Configuring Restriction Filters</i> on page 69.</p>
Users tab	<p>The Non Members list displays the users you can assign to this group. The Members list displays the users that are already assigned to this group. Use the arrow buttons to:</p> <ul style="list-style-type: none"> >> Add all users to this group. > Add the user selected in the Non Members list to the group. < Remove the user selected in the Members list from this group. << Remove all users from this group. <p>For information about configuring users, see <i>Configuring Users</i> on page 63.</p>

Assigning and Removing Group Roles

To assign or remove a role:

1. On the Netcool/OMNIbus Administrator window, click the **Users** drop-down list.
2. Click the **Groups** icon. The *Groups* window is displayed.
3. Select the group and click the **Edit Group** button. The *Group Details* window is displayed.
4. Click the **Roles** tab.

The **Available Roles** column lists the available roles you can assign to this group. The **Applied Roles** column lists the roles that are already assigned to this group.

5. Use the arrow buttons to assign roles to the group, as described in Table 18.

Table 18: Group Details Window Roles Tab Buttons

Button	Description
>>	Assign all roles to this group.
>	Assign the role selected in the Available Roles list to the group.
<	Remove the selected role in the Applied Roles list from this group.
<<	Remove all roles from this group.

6. Click **OK**. The group information is saved.

Assigning and Removing Group Restriction Filters

To assign or remove a restriction filter:

1. On the Netcool/OMNIbus Administrator window, click the **Users** drop-down list.
2. Click the **Groups** icon. The *Groups* window is displayed.
3. Select the group and click the **Edit Group** button. The *Group Details* window is displayed.
4. Click the **Restriction Filters** tab.
5. In the **All Restriction Filters** list, select the restriction filter to assign to this group.
6. Click the **Add Restriction Filter** button. The restriction filter is added to the **Assigned Restriction Filters** list.

Click the **Remove Restriction Filter** button to remove a selected restriction filter from the **Assigned Restriction Filters** list.

For information about creating restriction filters, see *Configuring Restriction Filters* on page 69.

7. Click **OK**. The group information is saved.

Adding and Removing Group Users (from the Groups Window)

To add or remove a group user:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Groups** icon. The *Groups* window is displayed.
3. Select the group and click the **Edit Group** button. The *Group Details* window is displayed.
4. Click the **Users** tab.

The **Non Members** list displays the available groups to which you can add this user. The **Members** list displays the groups to which this group currently belongs.

5. Use the arrow buttons to add the user to groups, as described in Table 19.

Table 19: Group Details Window Users Tab Buttons

Button	Description
>>	Add this user to all available groups.
>	Add this user to the group selected in the Non Members list.
<	Remove this user from the group selected in the Members list.
<<	Remove this user from all groups.

6. Click **OK**. The user group information is saved.

Deleting Groups

To delete a group:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Groups** icon. The *Groups* window is displayed.
3. Select the group to delete.
4. Click the **Delete** button. The group is deleted.

Configuring Users

You can create and modify Netcool/OMNIBus users. You can then assign users to groups for organizational purposes. Group roles control access to ObjectServer objects.

For information about configuring groups, see *Configuring Groups* on page 58. For information about configuring roles, see *Configuring Roles* on page 55.

To configure users, click the **Users** drop-down list; then, click the **Users** icon. The *Users* window contains a list of the users in the selected ObjectServer. An example is shown in Figure 7.

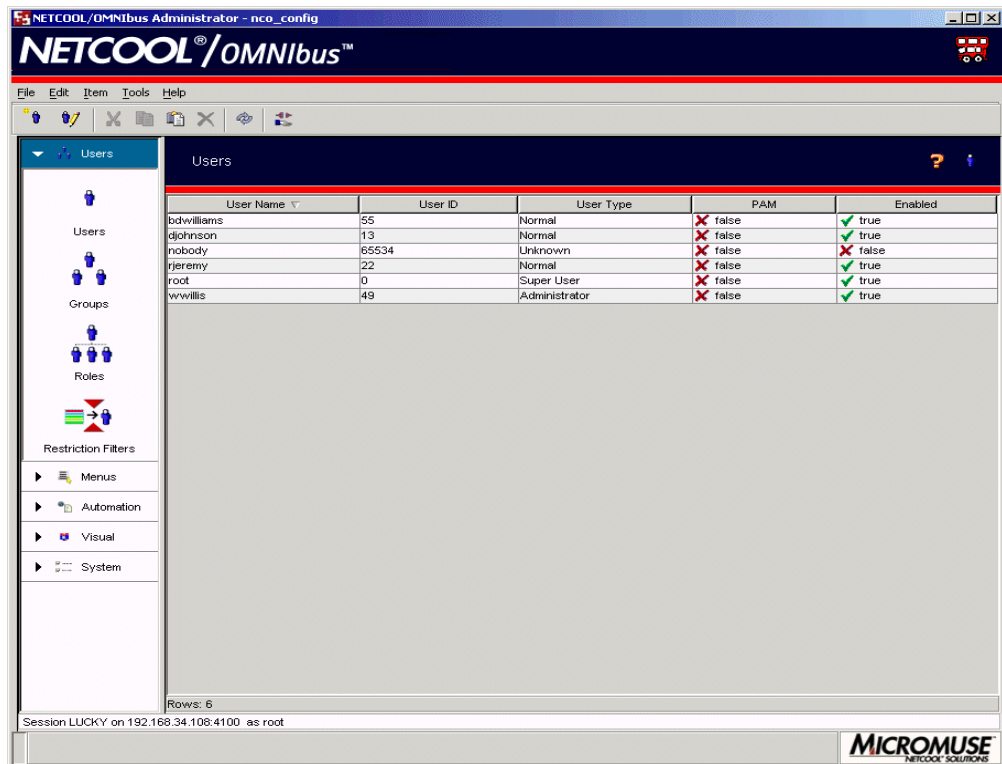


Figure 7: Netcool/OMNIBus Administrator Window - Users

Creating and Editing Users

To create or edit a user:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Users** icon. The *Users* window is displayed.

3. Click the **Add User** button, or select the user to edit and click the **Edit User** button. The *User Details* window is displayed.
4. Complete or edit the following window items and click **OK**.

Table 20: User Details Window Items (1 of 2)

Window Item	Description
Username	User names are restricted to 64 characters and can include spaces. If you are editing a user, you cannot change the name.
User ID	Select a unique ID for this user. This should be set to match the UNIX UID where possible. If you are editing this user, you cannot change the user ID.
Full Name	The user's full name.
Create Conversion	Indicates whether to create a conversion for this user. A conversion enables the user's name to appear in Netcool/OMNibus event lists instead of the user ID.
Groups tab	<p>The Unassigned column lists the available groups to which you can add this user. The Assigned Groups lists the groups to which this user is currently assigned. Use the arrow buttons to:</p> <ul style="list-style-type: none"> >> Add this user to all available groups. > Add this user to the group selected in the Unassigned Groups list. < Remove this user from the group selected in the Assigned Groups list. << Remove this user from all groups. <p>You can also click the Add new group button to create a group. For information about creating groups, see <i>Configuring Groups</i> on page 58.</p> <p>Note: If you do not add the user to a group, the user will have no permissions.</p> <p>For information about default groups, see the Netcool/OMNibus Installation and Deployment Guide.</p>
Restriction Filters tab	<p>You can optionally select restriction filters to apply to this user. Restriction filters enable you to prevent users from viewing or modifying certain rows in ObjectServer tables.</p> <p>The All Restriction Filters list contains the available restriction filters. The Assigned Restriction Filters list contains the restriction filters that are assigned to this user.</p> <p>Use the Add Restriction Filter or Remove Restriction Filter buttons to add or remove restriction filters for this user.</p> <p>For information about restriction filters, see <i>Configuring Restriction Filters</i> on page 69.</p>

Table 20: User Details Window Items (2 of 2)

Window Item	Description
Settings tab	<p>Use the Settings tab to enter and select the following information:</p> <p>Password/Verify</p> <p>Enter an optional password for this user in the text box. The password characters are hidden as you type. If you enter the user's password, retype it in the Verify field.</p> <p>You can click the Change button to reset the user's password. You must then re-enter it.</p> <p>Use PAM</p> <p>Indicates whether the external authentication method specified in the system Pluggable Authentication Modules (PAM) configuration file is used for authentication and password management.</p> <p>If PAM is <i>not</i> used, user names and password are stored in the ObjectServer.</p> <p>PAM is not used for authorization. Authorization is managed in the ObjectServer, as described in <i>Managing Roles, Groups, and Users</i> on page 55. For information about external authenticaton, see the Netcool/OMNIbus Installation and Deployment Guide.</p> <p>User Type</p> <p>A read-only field indicating the type of user. This is set automatically according to the groups to which this user belongs.</p> <p>User Enabled</p> <p>Indicates whether this user is enabled. You can create a user and enable the user at a later time.</p>



Tip: You can copy and paste an existing user to easily create a new user with identical group and restriction filter settings. When you select **Edit**→**Paste**, the *User Details* window appears for you to change the user name. You may also want to add the user's full name and password.

Adding and Removing Group Users (from the Users Window)

To add or remove a group user:

1. On the Netcool/OMNIbus Administrator window, click the **Users** drop-down list.
2. Click the **Users** icon. The *Users* window is displayed.
3. Select the user and click the **Edit User** button. The *User Details* window is displayed.
4. Click the **Groups** tab.

The **Unassigned Groups** list displays the available groups to which you can add this user. The **Assigned Groups** list displays the groups to which this user currently belongs.

- Use the arrow buttons to change the groups the user is a member of, as described in Table 21.

Table 21: User Details Window Groups Tab Buttons

Button	Description
>>	Add this user to all available groups.
>	Add this user to the group selected in the Unassigned Groups list.
<	Remove this user from the group selected in the Assigned Groups list.
<<	Remove this user from all groups.

- Click **OK**. The user group information is saved.

Assigning and Removing User Restriction Filters

To assign and remove a user restriction filter:

- On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
- Click the **Users** icon. The *Users* window is displayed.
- Select the user and click the **Edit User** button. The *User Details* window is displayed.
- Click the **Restriction Filters** tab.
- In the **All Restriction Filters** list, select the restriction filter to assign to this user.
- Click the **Add Restriction Filter** button. The restriction filter is added to the **Assigned Restriction Filters** list.

You can also click the **Remove Restriction Filter** button to remove a selected restriction filter from the **Assigned Restriction Filters** list.

For information about creating restriction filters, see *Configuring Restriction Filters* on page 69.

- Click **OK**. The user information is saved.

Deleting Users

To delete a user:

- On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
- Click the **Users** icon. The *Users* window is displayed.

3. Select the user to delete.
4. Click the **Delete** button. The user is deleted.

2.6 Configuring ObjectServer Objects

You can use Netcool/OMNIbus Administrator to configure the following ObjectServer objects:

- Restriction filters
- Event list menus
- Tools
- Prompts
- Trigger groups
- Triggers
- Procedures
- User defined signals
- Conversions
- Event list alert severity colors
- Column visuals
- Classes
- ObjectServer properties
- Databases
- Tables
- ObjectServer files

This section contains information about configuring each of the above objects.

ObjectServer Object Naming Conventions

When creating any ObjectServer object, you must give it a unique name for that type of object. The names of ObjectServer objects must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (`_`) characters, up to 40 characters in length.



Tip: User, group, and role names can be any text string up to 64 characters in length.

Names of ObjectServer objects are case-sensitive.

Configuring Restriction Filters

Restriction filters enable you to prevent users from viewing or modifying certain rows in ObjectServer tables. You can assign restrictions filters to users and groups. Once the filter has been applied to a user or group, the filter controls the data that can be displayed and modified from Netcool/OMNIBus client applications, and modified in INSERT, UPDATE, and DELETE statements. You can only assign one restriction filter per table to a user or group.

For example, you can create a restriction filter for the `alerts.status` table that prevents operators based in London from viewing alerts that originated from the New York operations center.

To configure restriction filters, click the **Users** drop-down list; then, click the **Restriction Filters** icon. The *Restriction Filters* window contains a list of the restriction filters in the selected ObjectServer. An example is shown in Figure 8.

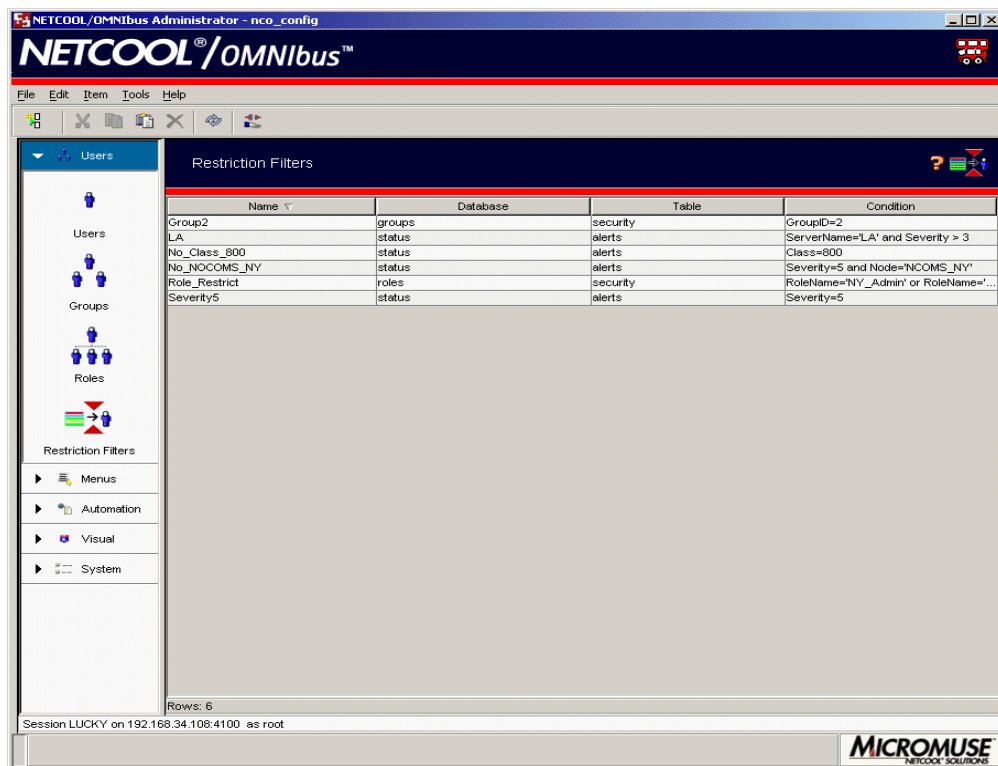


Figure 8: Netcool/OMNIBus Administrator Window - Restriction Filters

Creating Restriction Filters

To create a restriction filter:




1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Restriction Filters** icon. The *Restriction Filters* window is displayed, which lists all restriction filters in the selected ObjectServer.
3. Click the **Add Restriction Filter** button. The *Restriction Filter Details* window is displayed.
4. Complete the following window items and click **OK**.

Table 22: Restriction Filter Details Window Items

Window Item	Description
Filter Name	Enter a unique name for this restriction filter. Follow the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68.
Database	Select the ObjectServer database for which you are creating the restriction filter.
Table	Select the ObjectServer database table for which you are creating the restriction filter.
Condition	Enter the SQL condition (<i>where</i> statement) for the restriction filter. For example: <code>Tally > 100 AND Severity >=4</code> prevents a group with an ID equal to 10 from seeing alerts in the event list if they occur more than 100 times and have a severity greater than or equal to 4.
Helper Buttons	You can use the buttons next to the text area, which are described in Table 23, to help you create the filter condition.

You can use the helper buttons described in Table 23 to create the filter condition.

Table 23: Restriction Filter Helper Buttons

Button	Description
	Click this button to select a column name to add to the command. The column name is substituted for the corresponding event list row value when the tool is executed.
	Click this button to open a list of available conversions. Double-click a conversion to add it to the restriction filter. For information about conversions, see <i>Configuring Conversions</i> on page 96.
	Click this button to check the validity of the restriction filter.

Deleting Restriction Filters

To delete a restriction filter:

1. On the Netcool/OMNIBus Administrator window, click the **Users** drop-down list.
2. Click the **Restriction Filters** icon. The *Restriction Filters* window is displayed, which lists all restriction filters in the selected ObjectServer.
3. Select the restriction filter to delete.
4. Click the **Delete** button. The restriction filter is deleted.



Note: You can only delete restriction filters that are not in use.

Configuring Event List Menus

You can use Netcool/OMNIBus Administrator to customize Netcool/OMNIBus desktop menus. You can:

- Add, rename, and remove menu items, including sub-menus and separators
- Add tools to menus, which can be used with alerts that have an associated class
- Change the order of menu items
- Test menus

To configure Netcool/OMNIBus desktop menus, click the **Menus** drop-down list; then, click the **Menus** icon. The *Menus* window contains a list of the menus in the selected ObjectServer. An example is shown in Figure 9.

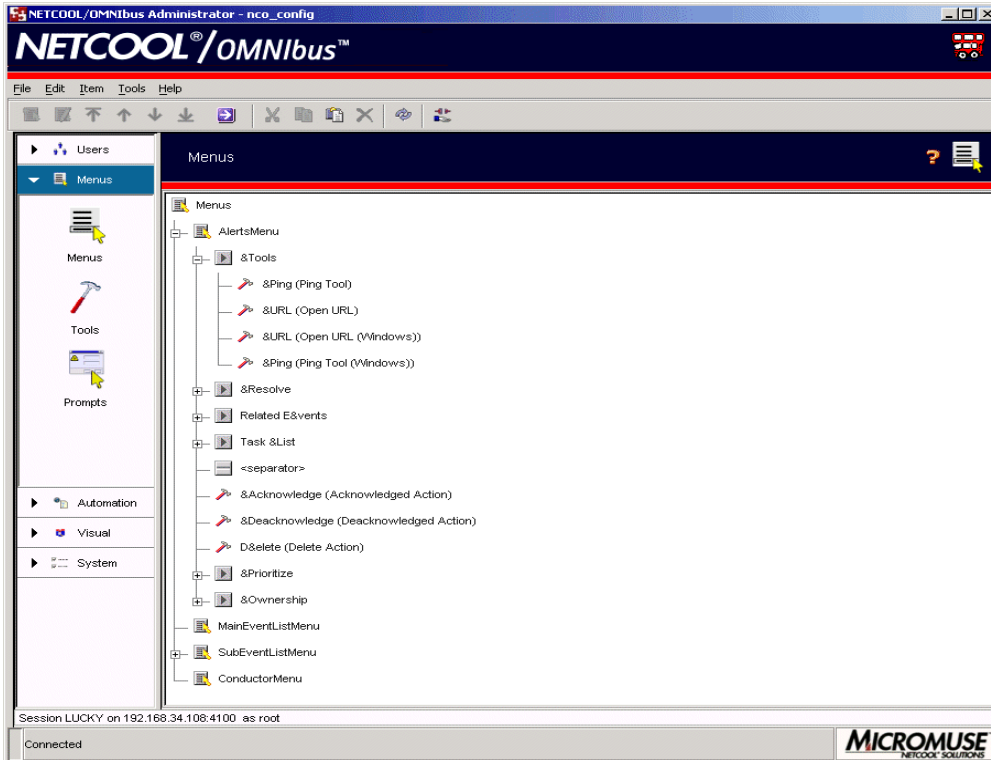


Figure 9: Netcool/OMNIBus Administrator Window - Menus

Click the symbol next to a menu or double-click the menu to see existing menu items and tools. The order in which the menu items appear on the *Menus* window is the order in which they appear in the Netcool/OMNIBus desktop.

Table 24 shows the menus you can configure. You can add items to these menus, or remove items from them, but you cannot delete the menus.

Table 24: Configurable Event List Menus (1 of 2)

Menu Name in the Netcool/OMNIBus Administrator Window	Menu Name in Netcool/OMNIBus
AlertsMenu	Alerts menu and pop-up menu on the event list, when an alert is selected.

Table 24: Configurable Event List Menus (2 of 2)

Menu Name in the Netcool/OMNIBus Administrator Window	Menu Name in Netcool/OMNIBus
MainEventListMenu	Tools menu for the event list monitor box window.
SubEventListMenu	Tools menu for any event list window.
ConductorMenu	Tools menu on the Conductor.

Adding Tools, Sub-menus, and Separators to a Menu

To add a tool, sub-menu, or separator to a desktop menu:

- On the Netcool/OMNIBus Administrator window, click the **Menus** drop-down list.
- Click the **Menus** icon. The *Menus* window is displayed, which lists the menus in the selected ObjectServer.
- Select the menu to which you are adding the menu item.
- Click the **Add Item** button. The *Menu Item Details* window is displayed.
- Click the **Menu Item Type** drop-down arrow and select one of the following:
 - **Tool**
 - **Separator**
 - **Sub Menu**
- If you are adding a tool, click the **Tool** drop-down arrow to select a tool.
You can also click the **Add Tool** button to create a new tool, or click the **Edit Tool** button to edit the selected tool. For information about creating and editing tools, see *Creating and Editing Tools* on page 76.
- If you are adding a tool or a sub-menu, in the **Title** field, enter the name as it will appear in the menu.
- Click **OK**. The new menu item appears on the *Menus* window. The menu item will appear in the event list the next time the desktop is started or the event list is resynced with the ObjectServer.

Editing Menu Items

You can edit tools and sub-menus on event list menus:

- On the Netcool/OMNIBus Administrator window, click the **Menus** drop-down list.
- Click the **Menus** icon. The *Menus* window is displayed, which lists the menus in the selected ObjectServer.

3. Select the menu item to edit.
4. Click the **Edit Item** button. The *Menu Item Details* window is displayed.
5. If you are editing a tool, click the **Tool** drop-down arrow to select a different tool.

You can also click the **Add Tool** button to create a new tool, or click the **Edit Tool** button to edit the selected tool. For information about creating and editing tools, see *Creating and Editing Tools* on page 76.





6. In the **Title** field, enter the name as it will appear in the menu.
7. Click **OK**. The menu item appears on the *Menus* window. The menu item will appear in Netcool/OMNIbus the next time the desktop is started or the event list is resynced with the ObjectServer.

Changing the Order of Menu Items

To change the order of desktop menu items:

1. On the Netcool/OMNIbus Administrator window, click the **Menus** drop-down list.
2. Click the **Menus** icon. The *Menus* window is displayed, which lists the menus in the selected ObjectServer.
3. Select the menu item to reorder.
4. Use the buttons in Table 25 to adjust the position of the menu item.

Table 25: Changing the Position of Desktop Menu Items

Button	Description
	Move item to the top of the menu.
	Move item up one position.
	Move item down one position.
	Move item to the bottom of the menu.

The menu item will be displayed in the selected position the next time the desktop is started or the event list is resynced with the ObjectServer.

Removing a Menu Item

To remove a menu item from the desktop:

1. On the Netcool/OMNIbus Administrator window, click the **Menus** drop-down list.
2. Click the **Menus** icon. The *Menus* window is displayed, which lists the menus in the selected ObjectServer.
3. Select the menu item to remove.
4. Click the **Delete** button. The menu item will be removed from the desktop the next time the desktop is started or the event list is resynced with the ObjectServer.

Testing Menus

You can test menus to see how they will appear in the desktop. To test menus:

1. On the Netcool/OMNIbus Administrator window, click the **Menus** drop-down list.
2. Click the **Menus** icon. The *Menus* window is displayed, which lists the menus in the selected ObjectServer.
3. Click the **Show Menu Structure** (→) button. The menus and menu items appear as they will in the desktop.



Note: The menu names as they appear in Netcool/OMNIbus Administrator differ from their names on the desktop. The names are listed in Table 24 on page 72.

Configuring Tools

Tools help operators manage alerts in event lists. For example, you can create tools that enable operators to:

- Execute external commands (for example, launch a local application, batch file, or script)
- Execute SQL commands on the ObjectServer

A tool can include a prompt window or a pop-up menu for the user to enter or select information. You can add tools to event list menus and associate them with alert classes. When you create a tool, it is added to the ObjectServer `tools` database.

For information about adding tools to event list menus, see *Adding Tools, Sub-menus, and Separators to a Menu* on page 73.



Note: You can use the `nco_elct` utility within a tool to open a customized, transient event list. For example, you can open an event list and apply a filter to view all `critical` alerts from a particular ObjectServer. For information about `nco_elct`, see Appendix B: *Desktop Reference* on page 291.

To configure tools, click the **Menus** drop-down list; then, click the **Tools** icon. The *Tools* window contains a list of the tools in the selected ObjectServer. An example is shown in Figure 10.

The screenshot shows the Netcool/OMNibus Administrator interface. The 'Tools' window is open, displaying a table of tools with columns for Name, Enabled, SQL, Executable, and Forced Journal. The status of each tool is indicated by green checkmarks for 'true' and red 'X' marks for 'false'.

Name	Enabled	SQL	Executable	Forced Journal
Acknowledged Action	✓ true	✓ true	✗ false	✗ false
Add to Task List	✓ true	✓ true	✗ false	✗ false
Assign Action	✓ true	✓ true	✗ false	✗ false
Change Severity	✓ true	✓ true	✗ false	✗ false
Deacknowledged Action	✓ true	✓ true	✗ false	✗ false
Delete Action	✓ true	✓ true	✗ false	✗ false
Group Action	✓ true	✓ true	✗ false	✗ false
Open URL	✓ true	✗ false	✓ true	✗ false
Open URL (Windows)	✓ true	✗ false	✓ true	✗ false
Ping Tool	✓ true	✗ false	✓ true	✗ false
Ping Tool (Windows)	✓ true	✗ false	✓ true	✗ false
Prompted Ping Tool	✓ true	✗ false	✓ true	✗ false
Prompted Ping Tool (Wind...	✓ true	✗ false	✓ true	✗ false
Prompted Telnet Tool	✓ true	✗ false	✓ true	✗ false
Prompted Telnet Tool (Wind...	✓ true	✗ false	✓ true	✗ false
Remove from Task List	✓ true	✓ true	✗ false	✗ false
Sample Tool	✓ true	✗ false	✓ true	✗ false
Sample Tool (Windows)	✓ true	✗ false	✓ true	✗ false
Show Related FE Node	✓ true	✗ false	✓ true	✗ false
Show Related FE Node (VM...	✓ true	✗ false	✓ true	✗ false
Show Related FE Object	✓ true	✗ false	✓ true	✗ false
Show Related FE Object (W...	✓ true	✗ false	✓ true	✗ false
Show Related NE Node	✓ true	✗ false	✓ true	✗ false
Show Related NE Node (VM...	✓ true	✗ false	✓ true	✗ false
Show Related NE Object	✓ true	✗ false	✓ true	✗ false
Show Related NE Object (W...	✓ true	✗ false	✓ true	✗ false
Suppress/Escalate	✓ true	✓ true	✗ false	✗ false
Takeownership Action	✓ true	✓ true	✗ false	✗ false
Telnet Tool	✓ true	✗ false	✓ true	✗ false
Telnet Tool (Windows)	✓ true	✗ false	✓ true	✗ false

Rows: 30
Session LUCKY on 192.168.34.108:4100 as root
Connected

Figure 10: Netcool/OMNibus Administrator Window - Tools

Creating and Editing Tools

To create or edit a tool:

1. On the Netcool/OMNibus Administrator window, click the **Menus** drop-down list.
2. Click the **Tools** icon. The *Tools* window is displayed, which lists the tools in the selected ObjectServer.

3. Click the **Add Tool** button, or select the tool to edit and click the **Edit Tool** button. The *Tool Details* window is displayed.
4. In the **Name** field, enter a unique name for this tool. If you are editing a tool, you cannot change the name.
5. Select the **Enabled** check box to enable operators to use this tool. You can create a tool and enable it at a later time.
6. Complete or edit the remaining window tabs. These are described in the following sections. You can also use the helper buttons and syntax described in *SQL Commands and Variable Expressions in Tools, Automations, and Transient Event Lists* on page 302.



Note: Although you may not be using this tool to run an executable, you must always use the **Executable** tab to indicate the platforms on which the tool can be used. The tool is only available on platforms that are selected.

7. Click **OK**. The tool is saved and displayed in the *Tools* window.

Completing the SQL Tab

Complete or edit the following window items:

Table 26: Tool Details Window Items - SQL Tab

Window Item	Description
Enabled	Select this check box to enable the SQL command for this tool.
Execute for each selected row	Select this check box to execute the SQL command once for each row in an event list row selection.
SQL Commands	Type the SQL command to execute when the tool is used.

Completing the Executable Tab

Complete or edit the following window items:

Table 27: Tool Details Window Items - Executable Tab (1 of 2)

Window Item	Description
Enabled	Select this check box to enable the executable commands for this tool.
Execute for each selected row	Select this check box to execute the executable commands once for each row in an event list row selection.

Table 27: Tool Details Window Items - Executable Tab (2 of 2)

Window Item	Description
Redirect output	Select this check box to control how the output is directed when the commands are run. If you do not select this check box, the output is discarded. When selected, output is echoed through a read-only window.
Redirect errors	Select this check box to control where error messages are sent when the commands are run. If you do not select this check box, error messages are discarded. When selected, errors are echoed through a read-only window.
Platforms	Select the operating system platforms on which the tool will be available. Note: Although you may not be using this tool to run an executable, you must always indicate the platforms on which the tool can be used. The tool is only available on platforms that are selected.
Executable Commands	Enter the commands to execute when this tools is run.

Completing the Journal Tab

Complete or edit the following window items:

Table 28: Tool Details Window Items - Journal Tab

Window Item	Description
Force Journal Entry	Select this check box to force the user to enter journal text before executing the tool. This text is appended to the journal of the selected alert or alerts when the tool is used.
Execute for each selected row	Select this check box to enter journal information once for each row in an event list row selection.
Journal Entry	Enter the text for the journal entry.

Completing the Access Tab

Complete or edit the following window items:

Table 29: Tool Details Window Items - Access Tab

Window Item	Description
Class Access	Select the alert classes for which this tool can be used. For information about configuring classes, see <i>Configuring Classes</i> on page 102.
Group Access	Select the user groups that can use this tool. For information about configuring groups, see <i>Configuring Groups</i> on page 58.

Completing the Description Tab

Enter or edit an optional text description for this tool. This can be useful to anyone who is trying to understand how the tool works.

Deleting Tools

To delete a tool:

1. On the Netcool/OMNIBus Administrator window, click the **Menus** drop-down list.
2. Click the **Tools** icon. The *Tools* window is displayed, which lists the tools in the selected ObjectServer.
3. Select the tool to delete.
4. Click the **Delete** button. The tool is deleted.

Configuring Prompts

A Netcool/OMNIBus event list tool can include a prompt window or a pop-up menu for the user to enter information. When creating or editing a tool, you can include a prompt in an SQL statement, external command, or journal entry. For information about creating event list tools, see *Configuring Tools* on page 75.

To configure prompts, click the **Menu** drop-down list; then, click the **Prompts** icon. The *Prompts* window contains a list of the prompts in the selected ObjectServer. An example is shown in Figure 11.

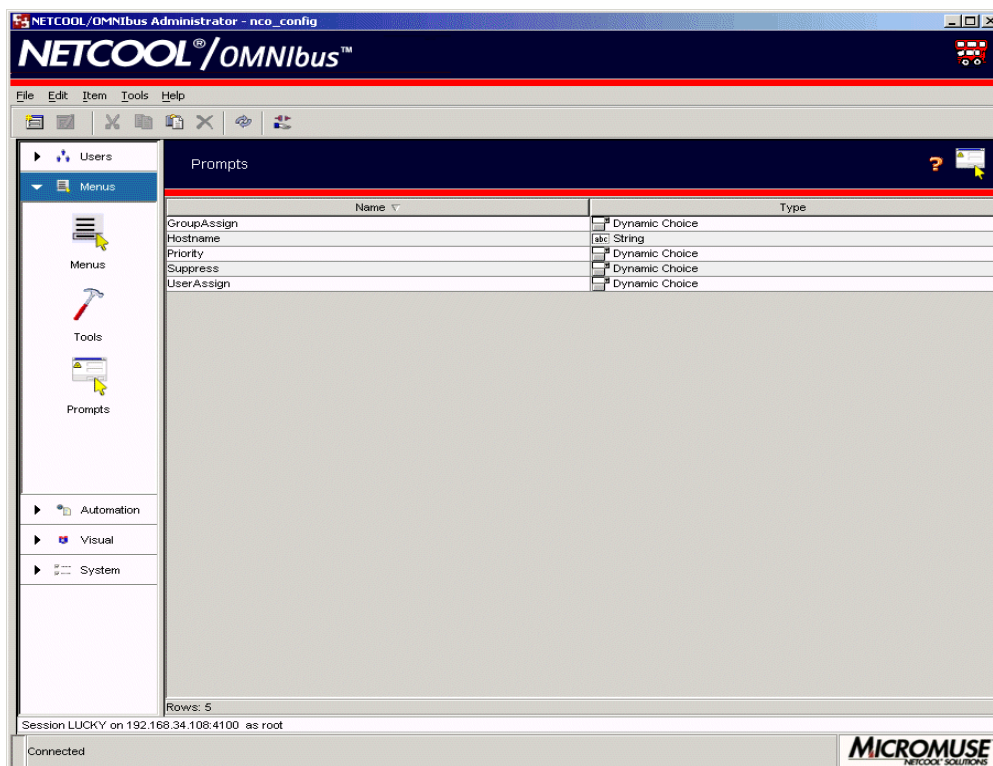


Figure 11: Netcool/OMNibus Administrator Window - Prompts

Creating and Editing Prompts

To create or edit a prompt:

1. On the Netcool/OMNibus Administrator window, click the **Menu** drop-down list.
2. Click the **Prompts** icon. The *Prompts* window is displayed, which lists the name and type of each prompt in the selected ObjectServer.
3. Click the **Add Prompt** button, or select the prompt to edit and click the **Edit Prompt** button. The *Prompt Details* window is displayed.

4. Complete or edit the following window items and click **OK**. The window items change depending on the selected prompt type.

Table 30: Prompt Details Window Items

Window Item	Description
Name	Enter a unique name for the prompt. If you are editing a prompt, you cannot change the name.
Prompt	Enter the prompt text to appear when the tool requests information from the user.
Type	<p>The available prompt types are:</p> <ul style="list-style-type: none"> • String • Integer • Float • Time • Fixed Choice • Lookup • Password • Dynamic Choice <p>The remaining fields depend on the selected prompt type. Information about completing each prompt type is included in the following sections.</p>

The following sections contain information about completing the *Prompt Details* window for each prompt type.

Completing a String Prompt

The String prompt creates a prompt window that accepts one or more characters. To complete the prompt details, enter a default value for the prompt to display.

Completing an Integer Prompt

The Integer prompt creates a prompt window that accepts an integer value. To complete the prompt details enter a default value for the prompt to display.

Completing a Float Prompt

The Float prompt creates a prompt window that accepts a floating point number, which can contain a decimal point.

To complete the prompt details, enter a default value for the prompt to display.

Completing a Time Prompt

The Time prompt creates a prompt window that accepts a time. The default display is the current time.

Completing a Fixed Choice Prompt

A Fixed Choice prompt creates a pop-up menu. The menu is populated by the values that you enter into the **Options** list.

To create a new option, click the **Add Option** button. You can also edit and delete the existing options.

Completing a Lookup Prompt

A Lookup prompt creates a pop-up menu or drop-down list. The menu or drop-down list is populated by the values in a file.

To complete the prompt details do one of the following:

- In the **File** field, enter the path and name of the file.
- Click the **Browse** button to open a standard file selection window. Select the file.

Completing a Password Prompt

The Password prompt creates a prompt window that accepts one or more characters. The password characters appear as asterisks when the user completes the prompt.

Completing a Dynamic Choice Prompt

A single dynamic choice prompt creates a pop-up menu or drop-down list. The menu or drop-down list is populated by the results of a database query. Table 31 describes the dynamic choice prompt window items.

Table 31: Dynamic Choice Prompt Items (1 of 2)

Window item	Description
Database	Select a database from the drop-down list.
Table	Select a table in the selected database from the drop-down list.
Show	Defines the column used to populate the prompt menu. Select a column name from the drop-down list.
Assign	Defines the column used to return a value to the SQL command, external command, or journal entry that contains the prompt. Select a column name from the drop-down list.
Where	Enter an SQL search condition.

Table 31: Dynamic Choice Prompt Items (2 of 2)

Window item	Description
Order By	<p>Defines the column used to order the items in the prompt menu. Select a column name from the drop-down list.</p> <p>Use the Ascending and Descending options to select the sort direction.</p>

Deleting a Prompt

To delete a prompt:

1. On the Netcool/OMNIbus Administrator window, click the **Menus** drop-down list.
2. Click the **Prompts** icon. The *Prompts* window is displayed, which lists the name and type of each prompt in the selected ObjectServer.
3. Select the prompt to delete.
4. Click the **Delete** button. The prompt is deleted.

Configuring Triggers

Triggers form the basis of the ObjectServer automation subsystem. Triggers enable the ObjectServer to automatically fire (execute a trigger action) when the ObjectServer detects an incident associated with a trigger. In a trigger, you can execute SQL commands and call procedures in response to the change.

Every trigger belongs to a *trigger group*, which is a collection of related triggers. You can easily enable or disable all triggers in a given group.

There are three types of triggers:

- Database
- Signal
- Temporal

This section contains instructions for configuring each type of trigger.



Note: For detailed information about triggers and how they work, see *Automation: Triggers and Trigger Groups* on page 201.

To configure triggers, click the **Automation** drop-down list; then, click the **Triggers** icon. The *Triggers* window contains a list of the triggers in the selected ObjectServer. An example is shown in Figure 12.

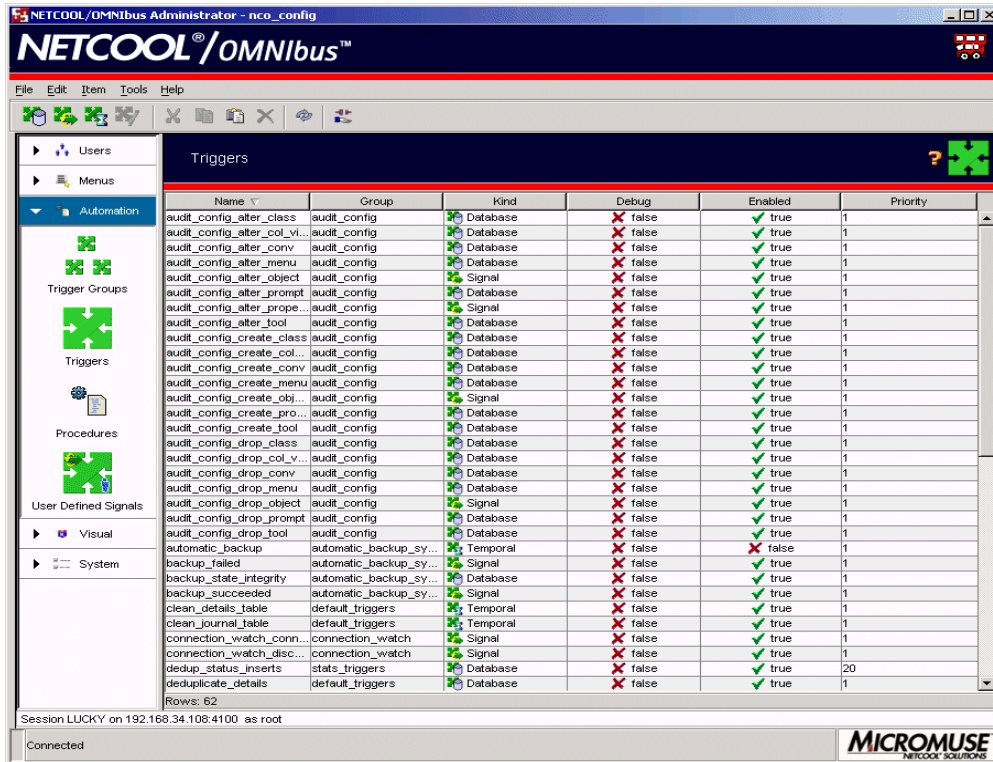


Figure 12: Netcool/OMNibus Administrator Window - Triggers

Creating and Editing Trigger Groups

To create or edit a trigger group:

1. On the Netcool/OMNIBUS Administrator window, click the **Automation** drop-down list.
2. Click the **Trigger Groups** icon. The *Trigger Groups* window is displayed, which lists the trigger groups in the selected ObjectServer.
3. Click the **Add Trigger Group** button, or select the trigger group to edit and click the **Edit Trigger Group** button. The *Trigger Group Details* window is displayed.

4. Complete or edit the following window items and click **OK**.

Table 32: Trigger Group Details Window Items

Window Item	Description
Group Name	Enter a unique name for this trigger group. Follow the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68. If you are editing a trigger group, you cannot change the name.
Enabled	Select to enable (activate) this trigger group. You can create a trigger group and enable it at a later time.

Deleting a Trigger Group

To delete a trigger group:

1. On the Netcool/OMNIbus Administrator window, click the **Automation** drop-down list.
2. Click the **Trigger Groups** icon. The *Trigger Groups* window is displayed, which lists the trigger groups in the selected ObjectServer.
3. Select the trigger group to delete.
4. Click the **Delete** button. The trigger group is deleted.



Note: You cannot delete a trigger group that contains triggers.

Creating and Editing Triggers

To create or edit a trigger:

1. On the Netcool/OMNIbus Administrator window, click the **Automation** drop-down list.
2. Click the **Triggers** icon. The *Triggers* window is displayed, which lists the triggers in the selected ObjectServer.
3. Click the toolbar button for the trigger type to create, or select the trigger to edit and click the **Edit Trigger** button.
4. If you are creating a new trigger, in the **Trigger Name** field, enter a unique trigger name. Follow the naming conventions described in *ObjectServer Object Naming Conventions* on page 68. If you are editing a trigger, you cannot change the name.
5. Click the **Group** drop-down list to select the trigger group to which this trigger belongs. For information about trigger groups, see *Creating and Editing Trigger Groups* on page 84.

6. Complete or edit the window for the selected trigger type. Instructions for configuring each trigger type are included in the following sections. You can also use the helper buttons and syntax described in *SQL Commands and Variable Expressions in Tools, Automations, and Transient Event Lists* on page 302.
7. When you have completed the window, click **OK**. The trigger is saved and displayed in the *Triggers* window.

Creating and Editing Database Triggers

A database trigger fires when a triggering database modification occurs. For example, you can create a trigger to perform an action each time an insert takes place on the `alerts.status` table.

The *Database Trigger Details* window is comprised of the **Settings**, **When**, **Action**, and **Comment** tabs. Table 33 describes the **Settings** tab.

Table 33: Database Trigger Details Window - Settings Tab Items (1 of 2)

Window item	Description
On	Select the ObjectServer database and table that causes the trigger to fire.
Priority	Determines the order in which the ObjectServer fires triggers when this database modification causes more than one trigger to fire. Select 1-20, with 1 being the highest priority.
Pre database action Post database action	<p>Indicate whether the trigger action is executed:</p> <ul style="list-style-type: none"> • <i>Before</i> the database modification that caused the trigger to fire occurs (Pre Database Action) • <i>After</i> the database modification that caused the trigger to fire occurs (Post Database Action) <p>For example, you can select Pre Database Action to evaluate a user name before a row in the <code>alerts.status</code> is deleted. In the trigger, you can detect whether the user is allowed to delete from the <code>alerts.status</code> table and, if not, prevent the database modification from taking place. If you select Post Database Action, the database modification always takes place.</p> <p>Also, select the database modification:</p> <ul style="list-style-type: none"> • Insert • Reinsert • Update • Delete
Apply to	<p>Select Row to set the trigger to fire once for each row that matches the trigger condition.</p> <p>Select Statement to set the trigger to fire once regardless of the number of matched rows in the table.</p>

Table 33: Database Trigger Details Window - Settings Tab Items (2 of 2)

Window item	Description
State	<p>Select Enabled to enable (activate) this trigger. You can create a trigger and enable it at a later time. A disabled trigger does not fire when the associated database modification occurs.</p> <p>Select Debug to send debugging information to the ObjectServer message log each time the trigger fires.</p>

Table 34 describes the **When**, **Action**, and **Comment** tabs.

Table 34: Database Trigger Details Window - Other Tab and Item Descriptions

Window item	Description
When tab	<p>The optional <code>WHEN</code> clause allows you to test for a particular condition before the action is executed. If the condition is not met, the action is not executed.</p> <p>For information about conditions, see <i>Conditions</i> on page 162.</p>
Action tab	<p>Enter the SQL commands for this trigger. The body of a trigger contains a set of SQL commands and programming constructs that manipulate data in the ObjectServer. The body of a trigger is enclosed within the keywords <code>BEGIN</code> and <code>END</code>. Each statement, except the last one, must be separated by a semi-colon (<code>;</code>).</p> <p>You can optionally define (declare) local variables for use within a trigger. A local variable is a placeholder for values used during the execution of the trigger. Local variable declarations within a trigger must be separated by semi-colons (<code>;</code>).</p> <p>For information about trigger statements and local variable declaration syntax, see <i>Syntax Elements Common to All Types of Triggers</i> on page 203.</p> <p>The trigger body has the following syntax:</p> <pre>[DECLARE variable_declaration;...[;]] BEGIN trigger_statement_list END;</pre>
Comment tab	<p>An optional text comment for this trigger. This may be useful to anyone who is trying to understand how the trigger works.</p>

Creating and Editing Signal Triggers

Signal triggers fire when a system or user defined signal is raised. *System* signals are raised spontaneously by the ObjectServer when it detects changes to the system; you do not have to do anything to create or configure them. *User defined* signals are explicitly created, raised, and dropped, as described in *Configuring User Defined Signals* on page 111.

For example, you can create a signal trigger to send an email to an operator when the ObjectServer starts or stops, since a system signal is generated when this occurs.

The *Signal Trigger Details* window is comprised of the **Settings**, **When**, **Evaluate**, **Action**, and **Comment** tabs. Table 35 describes the **Settings** tab.

Table 35: New Signal Trigger Details Window - Settings Tab Items

Window item	Description
Signal	Select the signal that causes this trigger to fire.
Priority	Determines the order in which the ObjectServer fires triggers when this signal causes more than one trigger to fire. Select 1-20, with 1 being the highest priority.
State	Select Enabled to enable (activate) this trigger. You can create a trigger and enable it at a later time. A disabled trigger does not fire when the associated signal is raised. Select Debug to send debugging information to the ObjectServer message log each time the trigger fires.

Table 36 describes the **When**, **Evaluate**, **Action**, and **Comment** tabs.

Table 36: New Signal Trigger Details Window- Other Tab and Item Descriptions (1 of 2)

Window item	Description
When tab	The optional WHEN clause allows you to test for a particular condition before the action is executed. If the condition is not met, the action is not executed. For information about conditions, see <i>Conditions</i> on page 162.
Evaluate tab	You can optionally build a temporary result set from a single SELECT statement to be processed during the trigger action (defined on the Action tab). Use the format: <code>EVALUATE SELECT_cmd</code> In the Bind As field, enter the name of the temporary table in which to store the result set.

Table 36: New Signal Trigger Details Window- Other Tab and Item Descriptions (2 of 2)

Window item	Description
Action tab	<p>Enter the SQL commands for this trigger. The body of a trigger contains a set of SQL commands and programming constructs that manipulate data in the ObjectServer. The body of a trigger is enclosed within the keywords <code>BEGIN</code> and <code>END</code>. Each statement, except the last one, must be separated by a semi-colon (;).</p> <p>You can optionally define (declare) local variables for use within a trigger. A local variable is a placeholder for values used during the execution of the trigger. Local variable declarations within a trigger must be separated by semi-colons (;).</p> <p>For information about trigger statements and local variable declaration syntax, see <i>Syntax Elements Common to All Types of Triggers</i> on page 203.</p> <p>The trigger body has the following syntax:</p> <pre>[DECLARE variable_declaration;...[;]] BEGIN trigger_statement_list END;</pre>
Comment tab	An optional text comment for this trigger. This can be useful to anyone who is trying to understand how the trigger works.

Creating and Editing Temporal Triggers

Temporal triggers fire repeatedly based on a specified frequency. For example, you can use a temporal trigger to delete all clear rows (`Severity = 0`) from the `alerts.status` table that have not been modified within a certain period of time.

The *Temporal Trigger Details* window is comprised of the **Settings**, **When**, **Evaluate**, **Action**, and **Comment** tabs. Table 37 describes the **Settings** tab.

Table 37: Temporal Trigger Details Window - Settings Tab Items

Window item	Description
Every	Select when the trigger will fire. Enter the numeric value and select hours , minutes , or seconds .
Priority	Determines the order in which the ObjectServer fires triggers when more than one trigger occurs at this frequency. Select 1-20, with 1 being the highest priority.
State	<p>Select Enabled to enable (activate) this trigger. You can create a trigger and enable it at a later time. A disabled trigger does not fire.</p> <p>Select Debug to send debugging information to the ObjectServer message log each time the trigger fires.</p>

Table 38 describes the **When**, **Evaluate**, **Action**, and **Comment** tabs.

Table 38: Temporal Trigger Details Window - Other Tab and Item Descriptions

Window item	Description
When tab	<p>The optional WHEN clause allows you to test for a particular condition before the action is executed. If the condition is not met, the action is not executed.</p> <p>For information about conditions, see <i>Conditions</i> on page 162.</p>
Evaluate tab	<p>You can optionally build a temporary result set from a single SELECT statement to be processed during the trigger action (defined on the Actions tab). Use the format:</p> <pre>EVALUATE SELECT_cmd</pre> <p>In the Bind As field, enter the name of the temporary table in which to store the result set.</p>
Actions tab	<p>Enter the SQL commands for this trigger. The body of a trigger contains a set of SQL commands and programming constructs that manipulate data in the ObjectServer. The body of a trigger is enclosed within the keywords BEGIN and END. Each statement, except the last one, must be separated by a semi-colon (;).</p> <p>You can optionally define (declare) local variables for use within a trigger. A local variable is a placeholder for values used during the execution of the trigger. Local variable declarations within a trigger must be separated by semi-colons (;).</p> <p>For information about trigger statements and local variable declaration syntax, see <i>Syntax Elements Common to All Types of Triggers</i> on page 203.</p> <p>The trigger body has the following syntax:</p> <pre>[DECLARE variable_declaration;...[;]] BEGIN trigger_statement_list END;</pre>
Comment tab	<p>An optional text comment for this trigger. This may be useful to anyone who is trying to understand how the trigger works.</p>

Deleting a Trigger

To delete a trigger:

1. On the Netcool/OMNIBus Administrator window, click the **Automation** drop-down list.
2. Click the **Triggers** icon. The *Triggers* window is displayed, which lists the triggers in the selected ObjectServer.
3. Select the trigger to delete.
4. Click the **Delete** button. The trigger is deleted.

Configuring Procedures

A procedure is an executable SQL object that can be called to perform common operations. The types of procedures are:

- SQL procedures, which manipulate data in an ObjectServer database
- External procedures, which run an executable on a remote system

Once you create a procedure in the ObjectServer, you can execute it from iSQL or in a trigger using the `EXECUTE PROCEDURE` command described in *Executing Procedures* on page 199.

Instructions for creating both procedure types are described in the following sections.



Note: For detailed information about procedures, see *Procedures* on page 189.

To configure procedures, click the Automation drop-down list; then, click the **Procedures** icon. The *Procedures* window contains a list of the procedures in the selected ObjectServer. An example is shown in Figure 13.

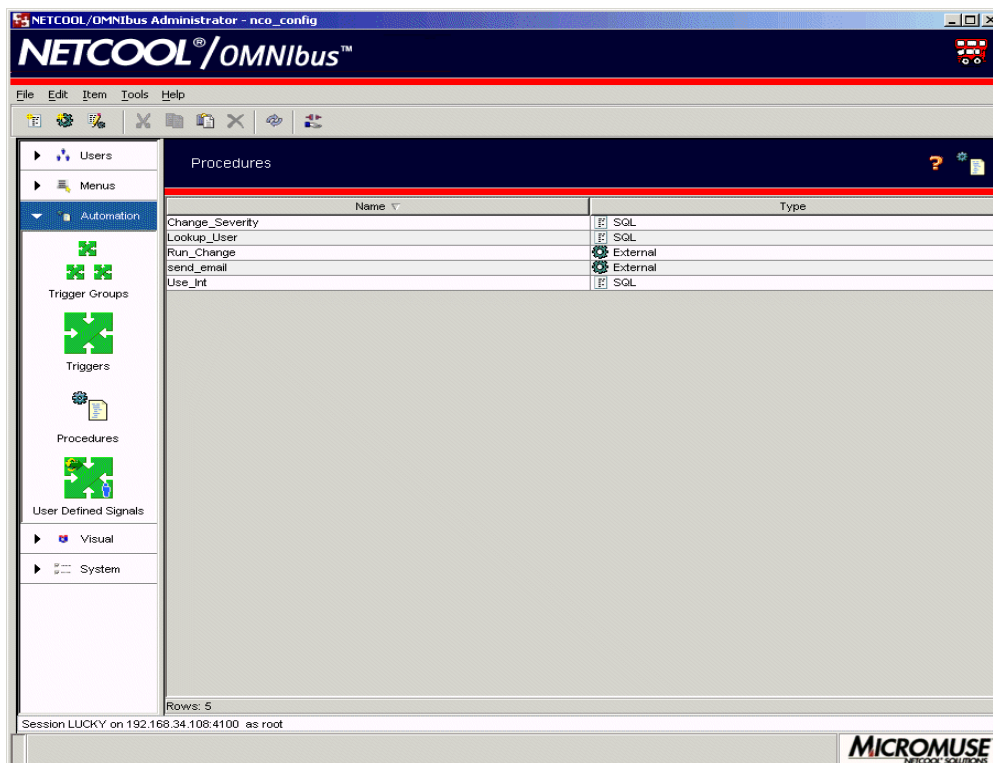


Figure 13: Netcool/OMNIBus Administrator Window - Procedures

Creating and Editing Procedures

To create or edit a procedure:

1. On the Netcool/OMNIBus Administrator window, click the **Automation** drop-down list.
2. Click the **Procedures** icon. The *Procedures* window is displayed, which lists all procedures in the selected ObjectServer.
3. Click either the **Add SQL Procedure** or **Add External Procedure** toolbar button, or select the procedure to edit and click the **Edit Procedure** button.

4. Complete the window for the selected procedure type. Instructions for creating each procedure type are included in the following sections.
5. When you have completed the procedure window, click **OK**. The procedure is saved and displayed in the *Procedures* window.

Creating and Editing SQL Procedures

SQL procedures have the following major components:

- Parameters
- Local variable declarations, declared before the procedure body
- Procedure body

The *SQL Procedure Details* window items are described in Table 39.

Table 39: SQL Procedure Details Window Items (1 of 2)

Window item	Description
Procedure Name	Enter a unique name for the procedure. Follow the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68. If you are editing a procedure, you cannot change the name.
Parameters list	<p>Displays the parameters to pass into and out of the procedure. Use the In/Out, Name, and Data Type fields to create the parameters.</p> <p>Parameters are values passed into or out of a procedure. You declare the procedure parameters when you create the procedure and specify what values are passed as parameters when you execute the procedure.</p> <p>Click the Add Parameter to the list button to add the parameter to the list.</p> <p>You can use the up and down arrows to change the order of the selected parameter, or click the Remove parameter button to remove the selected parameter from the list.</p> <p>For information about procedure parameters, see <i>SQL Procedure Parameters</i> on page 191.</p>
In/Out	Each procedure parameter has a mode, which can be in , out , or in out . Depending on the mode you choose for your parameters, you can use them in different ways (as described in <i>SQL Procedure Parameters</i> on page 191).
Name	Enter the parameter name. Parameter names must be unique within the procedure and comply with the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68.
Data Type	Select the type of data the parameter can pass into or out of the procedure. The data type can be any valid ObjectServer data type as described in Table 53 on page 136, except VARCHAR or INCR.

Table 39: SQL Procedure Details Window Items (2 of 2)

Window item	Description
Array	If you selected the in mode (in the In/Out field), you can select this check box to pass an array of the selected data type into the procedure.
Add parameter to the list button	After completing the In/Out , Name , Data Type , and (if necessary) Array fields, click this button to add the parameter to the parameter list.
Actions	<p>Enter the SQL commands for this procedure. The body of a procedure contains a set of SQL commands and programming constructs that manipulate data in the ObjectServer. The body of a procedure is enclosed within the keywords BEGIN and END. Each statement, except the last one, must be separated by a semi-colon (;).</p> <p>You can optionally define (declare) local variables for use within a procedure. A local variable is a placeholder for values used during the execution of the procedure. Local variable declarations within a procedure must be separated by semi-colons (;).</p> <p>For information about procedure statements and local variable declaration syntax, see <i>SQL Procedure Body</i> on page 193 and <i>SQL Procedure Variable Declarations</i> on page 192.</p>

You can also use the helper buttons and syntax described in *SQL Commands and Variable Expressions in Tools, Automations, and Transient Event Lists* on page 302.



Example SQL Procedure

The following example SQL procedure generates a report on the total number of alerts received (and deduplicated) for a given node.

```
create file node_report_file '/tmp/node_report';
create procedure node_report( node_name char(255) )
declare
    tally_total integer;
begin
    set tally_total = 0;
    for each row tmprow in alerts.status where tmprow.Node = node_name
    begin
        set tally_total = tally_total + tmprow.Tally;
    end;
    write into node_report_file values ( 'Total tally for node ', node_name, ' = ',
tally_total );
end;
```

Creating and Editing External Procedures

You can create external procedures to run an executable file on a local or remote system. The *External Procedure Details* window items are described in Table 40.

Table 40: External Procedure Details Window Items

Window item	Description
Procedure Name	Enter a unique name for the procedure. Follow the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68. If you are editing a procedure, you cannot change the name.
Parameters list	Displays the procedure parameters. These can be base type variables, arrays of base type values, or rows of named tables. Use the In/Out , Name and Data Type fields, and the Array check box, to create the parameters. Click the Add Parameter to the list button to add the parameter to the list. You can use the up and down arrows to change the order of the selected parameter, or click the Remove parameter button to remove the selected parameter from the list.
In/Out	External procedure parameters are always IN parameters. You can use an IN parameter in expressions to help calculate a value, but you cannot assign a value to the parameter.
Name	Enter the parameter name. Parameter names must be unique within the procedure and comply with the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68.
Data Type	Select the type of data the parameter can pass into the procedure. The data type can be any valid ObjectServer data type as described in Table 53 on page 136, except VARCHAR or INCR .
Array	Select to pass an array of the selected data type into the procedure.
Add parameter to the list button	After completing the In/Out , Name , Data Type , and (if necessary) Array fields, click this button to add the parameter to the parameter list.
Executable	Enter the full path for the command to run.
Arguments	Enter any command line arguments for the command to run.
Host	Enter the host on which to execute the procedure.
User ID	Enter the (UNIX) user ID under which to run the executable.
Group ID	Enter the (UNIX) group ID under which to run the executable.



Note: To execute an external procedure, you must have a process control agent daemon (`nco_pad`) running on the host where the executable resides. For more information on process control, see Chapter 4: *Process Control* on page 223.



Example External Procedure

The following example external procedure calls a program called `nco_mail`, which sends email about unacknowledged critical alerts.

```
create or replace procedure send_email
  (in node character(255), in severity integer, in subject character(255),
  in email character(255), in summary character(255), in hostname character(255))
  executable '$OMNIHOME/utils/nco_mail'
  host 'localhost'
  user 0
  group 0
  arguments '\'' + node + '\'', severity, '\'' + subject + '\'',
           '\'' + email + '\'', '\'' + summary + '\'';
```

This example also shows how to pass text strings to an executable. Strings must be enclosed in quotes, and the quotes must be escaped with backslashes. All quotes in this example are single quotes.

Deleting Procedures

To delete a procedure:

1. Click the **Automation** drop-down list.
2. Click the **Procedures** icon. The *Procedures* window is displayed, which lists all procedures in the selected ObjectServer.
3. Select the procedure to delete.
4. Click the **Delete** button. The procedure is deleted.

Note: You cannot delete a procedure if it is being used in a trigger.



Configuring Conversions

In the event list, conversions translate integer values into strings for readability. Conversions are associated with columns in the ObjectServer `alerts.status` table.

For example, default conversions exist for event list alert severities. If an alert has a severity of 4, the text `Major` is displayed for the alert severity in the event list.

To configure conversions, click the **Visual** drop-down list; then, click the **Conversions** icon. The *Conversions* window contains a list of the existing conversions for each column. An example is shown in Figure 14.

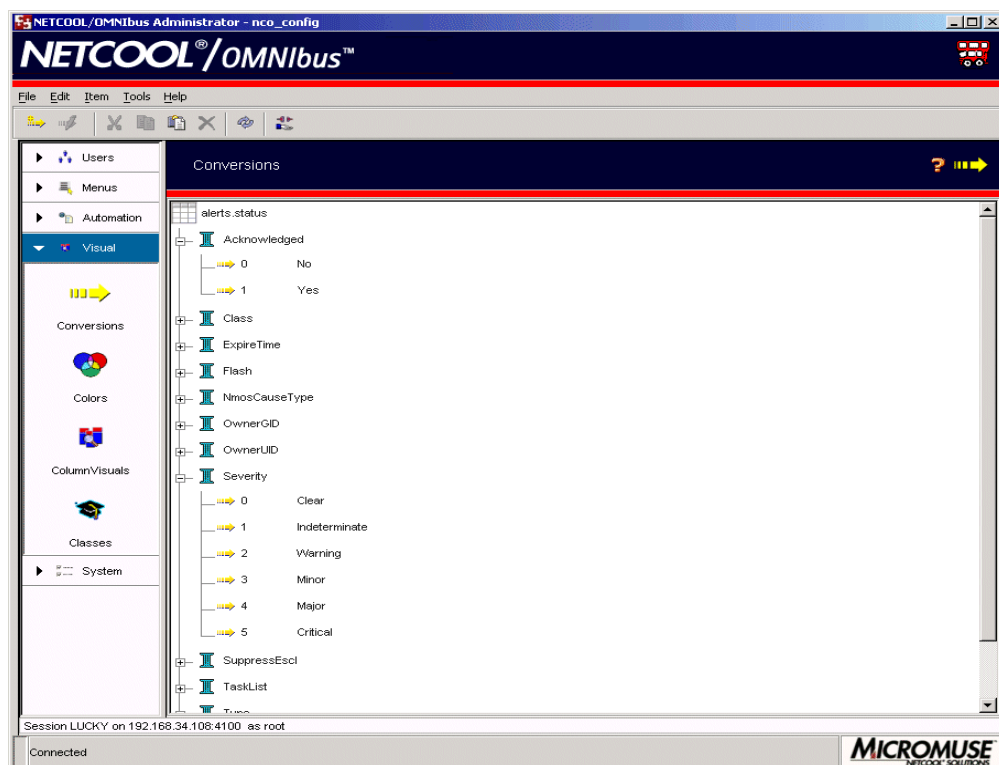


Figure 14: Netcool/OMNIBus Administrator Window - Conversions

To see existing conversions, click the symbol next to a column or double-click the column name.

Creating and Editing Conversions

To create or edit a conversion:

1. On the Netcool/OMNIBus Administrator window, click the **Visual** drop-down list.
2. Click the **Conversions** icon. The *Conversions* window is displayed, which lists the conversions in the selected ObjectServer.
3. Click the **Add Conversion** button, or select the conversion to edit and click the **Edit Conversion** button. The *Conversion Details* window is displayed.

4. Complete or edit the following window items and click **OK**.

Table 41: Conversion Details Window Items

Window Item	Description
Column	Select the name of the column containing the data to be converted.
Value	Enter the integer value which is to be converted.
Conversion	Enter the string to display instead of the value.

Deleting Conversions

To delete a conversion:

1. On the Netcool/OMNIBus Administrator window, click the **Visual** drop-down list.
2. Click the **Conversions** icon. The *Conversions* window is displayed, which lists the conversions in the selected ObjectServer.
3. Select the conversion to delete.
4. Click the **Delete** button. The conversion is deleted.

Configuring Event List Alert Severity Colors (Windows Event Lists Only)

You can view, create, and modify the alert severity colors used in Windows event lists. You can select different colors for acknowledged and unacknowledged alerts.

To configure alert severity colors in Windows event lists, click the **Visual** drop-down list; then, click the **Colors** icon. The *Color Details* window contains a list of the colors in the selected ObjectServer. An example is shown in Figure 15.

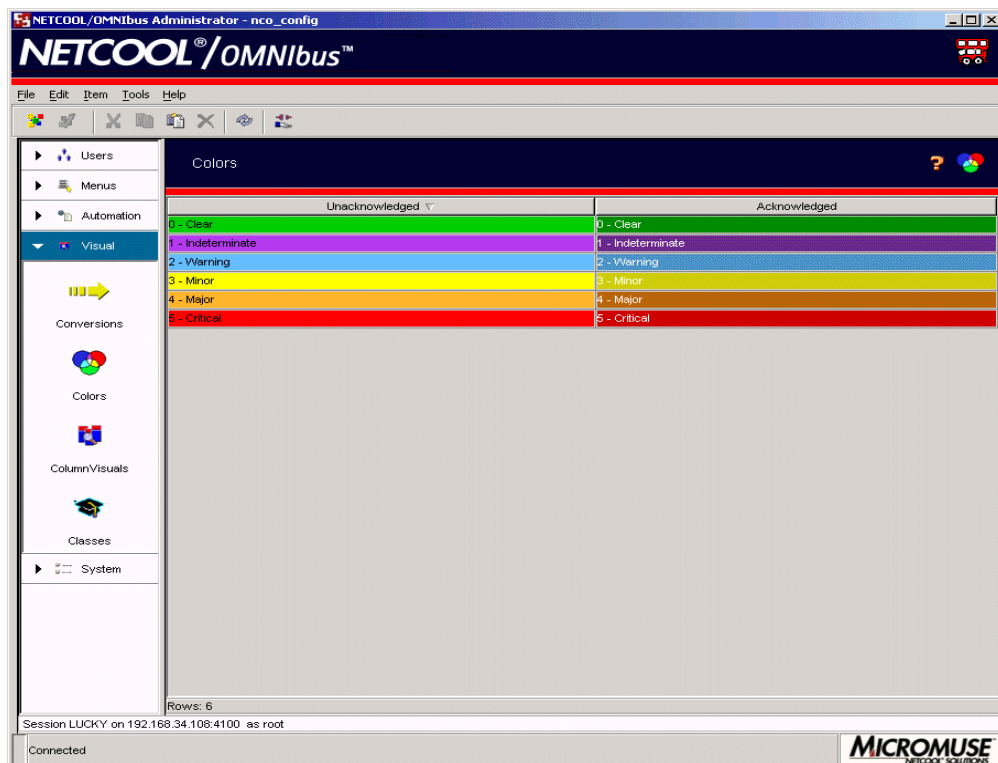


Figure 15: Netcool/OMNIBus Administrator Window - Colors

Creating and Editing Alert Severity Colors (Windows Event Lists Only)

To create or edit alert severity colors in Windows event lists:

1. On the Netcool/OMNIBus Administrator window, click the **Visual** drop-down list.
2. Click the **Colors** icon. The *Colors* window is displayed.
3. Click the **Add Color Definition** button to add a new severity color, or select the color to edit and click the **Edit Color Definition** button. The *Color Details* window is displayed.

4. Complete or edit the following window items and click **OK**.

Table 42: Color Details Window Items

Window Item	Description
Severity	If you are creating a new color, enter the alert severity value.
Conversion	Displays the conversion for this alert severity. Conversions are used to translate integer values into strings for readability. For example, the default conversion for a severity of 4 is major . Note: You cannot use the <i>Color Details</i> window to edit conversions. For information about creating and editing conversions, see <i>Configuring Conversions</i> on page 96.
Unacknowledged	Displays the color for this alert severity when it is unacknowledged in event lists. The default alert severity colors for unacknowledged alerts are: 0 Green 1 Violet 2 Blue 3 Yellow 4 Orange 5 Red
Acknowledged	Displays the color for this alert severity when it is acknowledged in event lists. The default alert severity colors for acknowledged alerts are: 0 Dark Green 1 Dark Violet 2 Dark Blue 3 Dark Yellow 4 Dark Orange 5 Dark Red
Color selection button	Click the color selection button to select the severity color for both acknowledged and unacknowledged alerts. You can choose the color using its swatch, HSB, and RGB values.

Configuring Column Visuals

When alert information is displayed in event lists, the visual appearance is defined by the settings of the column visuals. For each column, you can set the title text, default and maximum widths, and title and data justification.

To configure column visuals, click the **Visual** drop-down list; then, click the **Column Visuals** icon. The *Column Visuals* window contains a list of the column visuals in the selected ObjectServer. An example is shown in Figure 16.

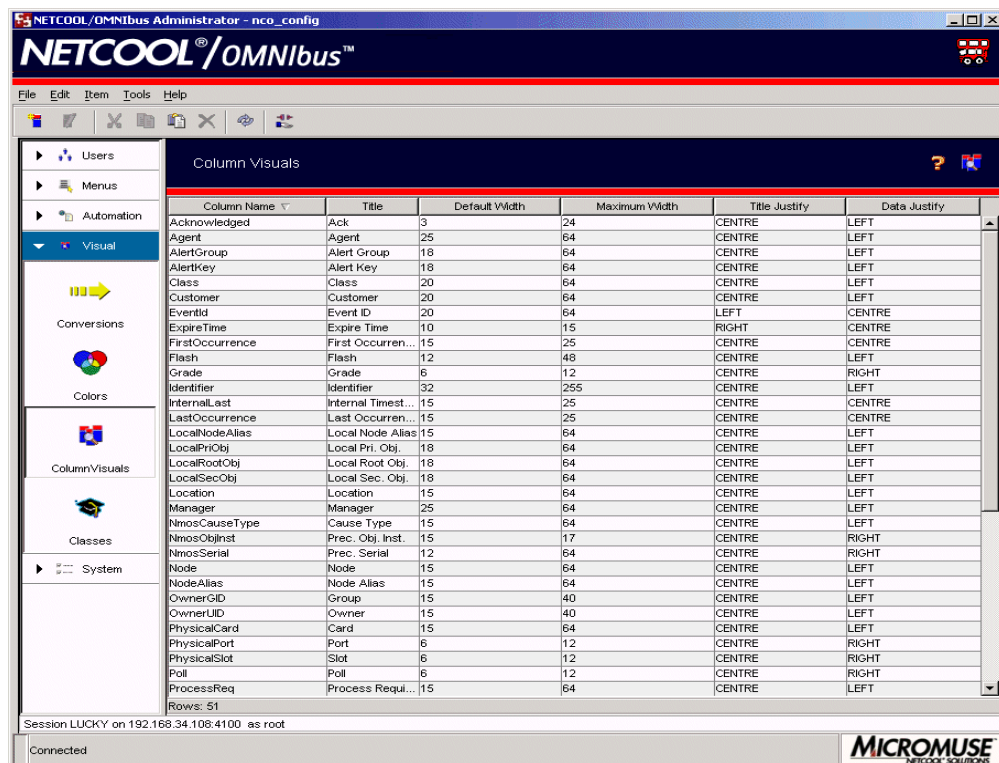


Figure 16: Netcool/OMNIBus Administrator Window - Column Visuals

Creating and Editing Column Visuals

To create or edit a column visual:

1. On the Netcool/OMNIBus Administrator window, click the **Visual** drop-down list.
2. Click the **Column Visuals** icon. The *Column Visuals* window is displayed, which lists the column visuals in the selected ObjectServer.
3. Click the **Add Column Visual** button, or select the column visual to edit and click the **Edit Column Visual** button. The *Column Visuals* window is displayed.

4. Complete or edit the following window items and click **OK**.

Table 43: Column Visual Details Window Items

Window Item	Description
Column	If you are creating a new column visual, click the drop-down arrow to select the column for which you are adding the visual.
Title	Enter the title that is to appear at the top of the column in Netcool/OMNIBus event lists.
Default (Size)	Enter the default column width (in characters).
Maximum (Size)	Enter the maximum column width (in characters).
Title (Justification)	Select the justification of the column title. Choose Left , Right , or Center from the drop-down list.
Column (Justification)	Select the justification of the information in the column. Choose Left , Right , or Center from the drop-down list.

Deleting Column Visuals

To delete a column visual:

1. On the Netcool/OMNIBus Administrator window, click the **Visual** drop-down list.
2. Click the **Column Visuals** icon. The *Column Visuals* window is displayed, which lists the column visuals in the selected ObjectServer.
3. Select the column visual to delete.
4. Click the **Delete** button. The column visual is deleted.

Configuring Classes

Alerts in the ObjectServer have a class assigned by the probe. Each class can be associated with an event list tool menu that contains useful tools for alerts of that class. For information about configuring tools, see *Configuring Tools* on page 75.

To configure classes, click the **Visual** drop-down list; then, click the **Classes** icon. The *Classes* window contains a list of the classes in the selected ObjectServer. An example is shown in Figure 17.

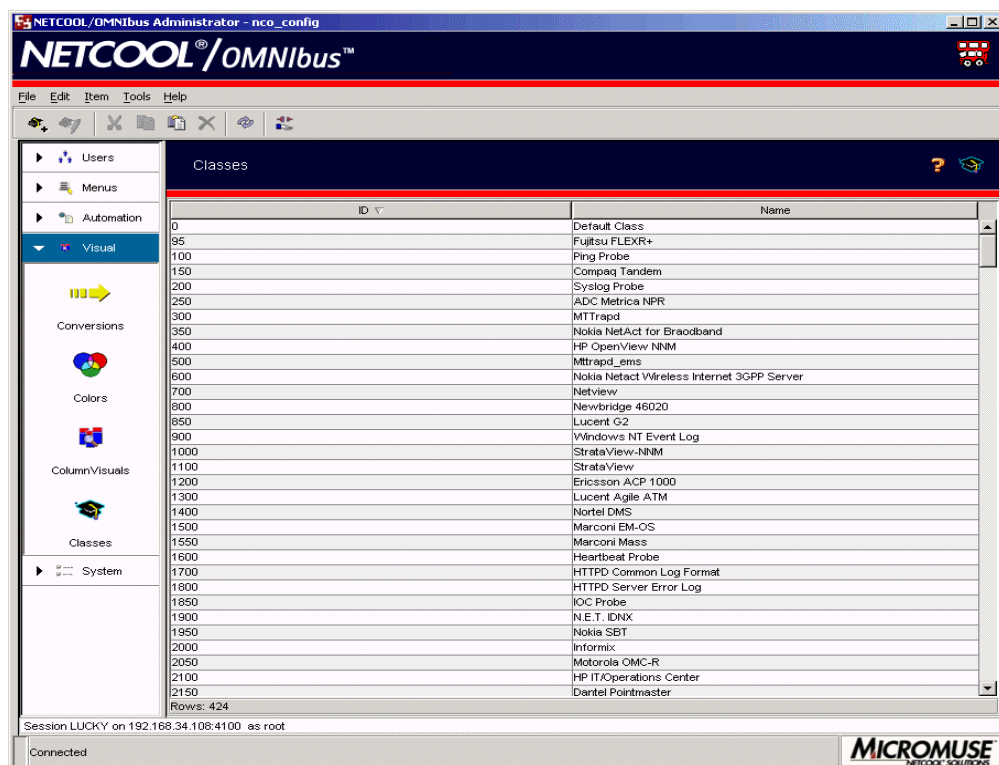


Figure 17: Netcool/OMNIBus Administrator Window - Classes

Creating and Editing Classes

To create or edit a class:

1. On the Netcool/OMNIBus Administrator window, click the **Visual** drop-down list.
2. Click the **Classes** icon. The *Classes* window is displayed, which lists the classes in the selected ObjectServer.
3. Click the **Add Class** button, or select the class to edit and click the **Edit Class** button. The *Class Details* window is displayed.

4. Complete or edit the following window items and click **OK**.

Table 44: Class Details Window Items

Window Item	Description
Identifier	Enter the class identifier for the new class. Alerts in the ObjectServer are assigned a class identifier by the probe. Class identifiers are defined by Micromuse Support for particular equipment types.
Name	Enter a label for the equipment type to be associated with the class number.

Deleting Classes

To delete a class:

1. On the Netcool/OMNIBus Administrator window, click the **Visual** drop-down list.
2. Click the **Classes** icon. The *Classes* window is displayed, which lists the classes in the selected ObjectServer.
3. Select the class to delete.
4. Click the **Delete** button. The class is deleted.

Viewing and Changing ObjectServer Properties

ObjectServer properties help determine the behavior of the ObjectServer. The default location of the ObjectServer properties file is `$OMNIHOME/etc/servername.props`. The ObjectServer reads the properties file when it starts.

You can view and change ObjectServer properties using Netcool/OMNIBus Administrator.



Note: You cannot add ObjectServer properties; you can only edit existing ones. It is essential that you are familiar with the properties before modifying them. Incorrect configuration can negatively impact system performance and functionality. For detailed information about ObjectServer properties, see *ObjectServer Properties and Command Line Options* on page 19.

To view ObjectServer properties, click the **System** drop-down list; then, click the **Properties** icon.

The *ObjectServer Properties* window contains a list of the current properties and settings for the selected ObjectServer. An example is shown in Figure 18.

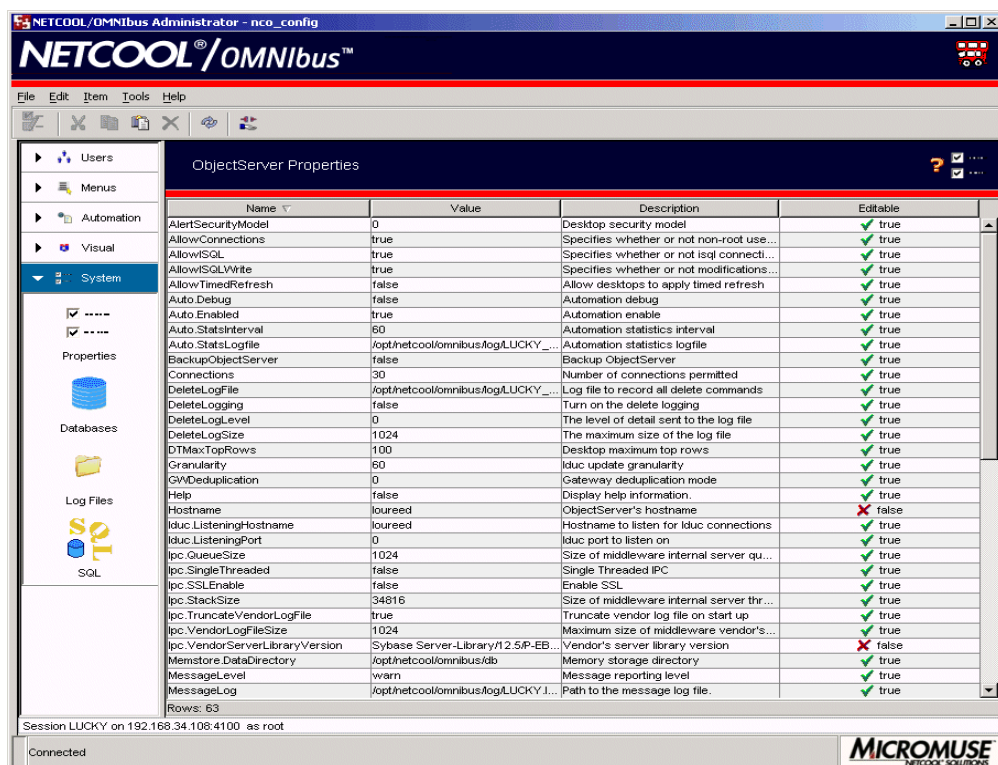


Figure 18: Netcool/OMNIBus Administrator Window - ObjectServer Properties

Changing ObjectServer Properties

To change the value of an ObjectServer property:

1. On the Netcool/OMNIBus Administrator window, click the **System** drop-down list.
2. Click the **Properties** icon. The ObjectServer Properties window is displayed, which contains the name, value, and description of each ObjectServer property, and whether the property is editable.

Note: View-only properties have the text `false` in the **Editable** column.



3. Double-click the property value to edit. The *Property Details* window is displayed.
4. Make any changes to the property value and click **OK**. Your changes are saved automatically.



Tip: Changes to some ObjectServer properties do not take effect until you restart the ObjectServer. To view a list of these properties, use iSQL (described in *Accessing an ObjectServer Using the SQL Interactive Interface (iSQL)* on page 119) to enter the following query: `select PropName from catalog.properties where IsImmediate=FALSE;`

Configuring Databases

You can use Netcool/OMNibus Administrator to create and manage ObjectServer databases. You can create and drop databases, and you can create, drop, and edit (alter) database tables.



Note: You are not permitted to make changes to system databases. System databases have a lock icon next to them. For information about system databases, see *System-Initialized Databases* on page 134.

For detailed information about ObjectServer objects and using SQL to manage them, see Chapter 3: *ObjectServer SQL* on page 123.

To configure databases, click the **System** drop-down list; then, click the **Databases** icon. The *Databases, Tables and Columns* window contains a list of the databases and tables in the selected ObjectServer.

An example alerts .status table is shown in Figure 19.

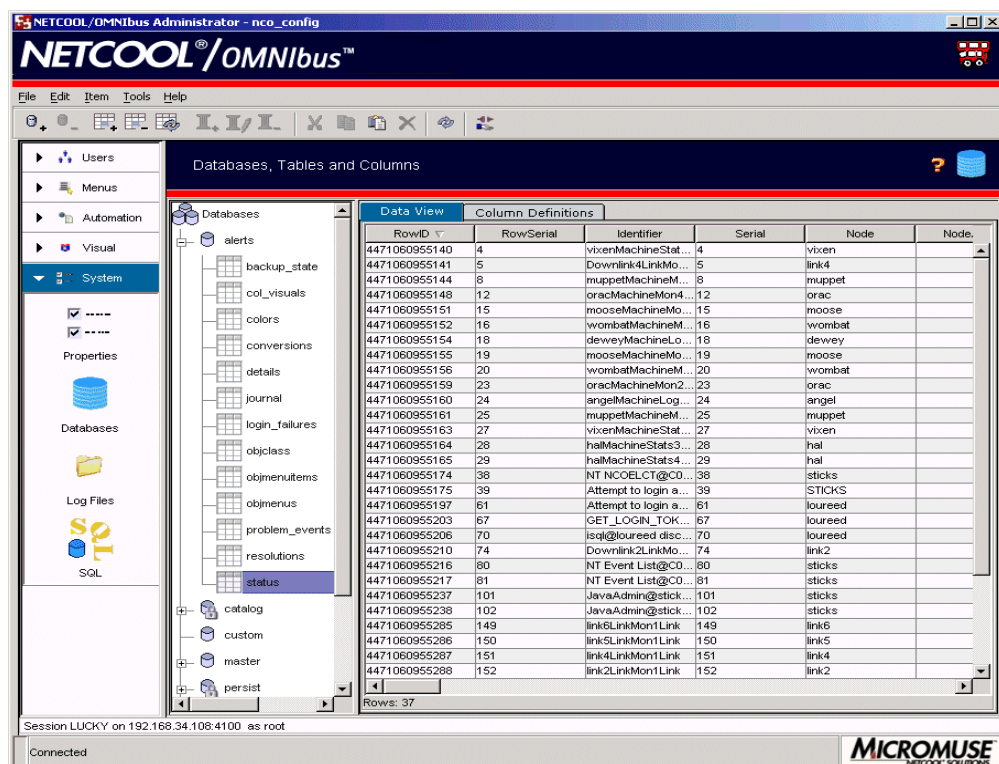


Figure 19: Netcool/OMNIBus Administrator Window - Databases, Tables, and Columns

You can view and configure table and column information for each database listed. Use the **Data View** and **Column Definitions** tabs to view data contained in the database.

To refresh any displayed information, click the icon for the currently selected database or table, or click the **Refresh** toolbar button.

Creating Databases

To create an ObjectServer database:

1. On the Netcool/OMNIBus Administrator window, click the **System** drop-down list.
2. Click the **Databases** icon. The *Databases, Tables and Columns* window is displayed.
3. Click the **Create Database** button. The *Database Details* window is displayed.

4. In the **Name** field, enter the database name. Follow the naming conventions described in *ObjectServer Object Naming Conventions* on page 68.
5. Click **OK**. The database is added to the ObjectServer and is displayed in the *Databases, Tables and Columns* window.

You can now add tables to the database. For information, see *Creating Tables* on page 108.

Deleting Databases

To delete a database:

1. On the Netcool/OMNIBus Administrator window, click the **System** drop-down list.
2. Click the **Databases** icon. The *Databases, Tables and Columns* window is displayed.
3. Select the database to delete.
4. Click the **Drop Database** button. The database is removed from the ObjectServer.



Note: You cannot delete a database if it contains any database objects, such as tables. Also, you cannot delete system databases, which have a lock icon next to them.

Creating Tables

To create a table:

1. On the Netcool/OMNIBus Administrator window, click the **System** drop-down list.
2. Click the **Databases** icon. The *Databases, Tables and Columns* window is displayed.
3. Select the database in which you are creating the table.
4. Click the **Create Table** button. The *Table Details* window is displayed.
5. In the **Name** field, enter the table name. Follow the naming conventions described in *ObjectServer Object Naming Conventions* on page 68.
6. Select the table **Type**:
 - **Persistent**—When the ObjectServer restarts, a persistent table is recreated with all data.
 - **Virtual**—When the ObjectServer restarts, a virtual table is recreated with the same table description, but without any data.
7. Use the column buttons to create, edit, and drop columns in this table. Use the arrow buttons to change the order of the selected column in the table.

For *Column Details* window field descriptions, see Table 45 on page 110.



Tip: You can use the **Data View** and **Column Definitions** tabs on the *Databases, Tables and Columns* window to view the table data and detailed information about the columns in the table.

Dropping Tables

To drop a table:

1. On the Netcool/OMNIbus Administrator window, click the **System** drop-down list.
2. Click the **Databases** icon. The *Databases, Tables and Columns* window is displayed.
3. Select the database containing the table to drop.
4. Select the table to drop.
5. Click the **Drop Table** button. The table is removed from the database.



Note: You cannot drop a table if it contains any database objects, such as a row, or if the table has any dependents.

Adding and Editing Table Columns

To add or edit a table column:

1. On the Netcool/OMNIbus Administrator window, click the **System** drop-down list.
2. Click the **Databases** icon. The *Databases, Tables and Columns* window is displayed.
3. Select the table in which you are adding or editing the column.
4. Click the **Column Definitions** tab.
5. If you are editing a column, select the column to edit.
6. Click the **Add Column** or **Edit Column** button. The *Column Details* window is displayed.

7. Complete or edit the following window items and click **OK**.

Table 45: Column Details Window Items

Window Item	Description
Column Name	Enter the column name. Follow the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68. If you are editing the column, you cannot change the name.
Data Type	<p>Each column value in the ObjectServer has an associated data type. The data type determines how the ObjectServer processes the data in the column. If you are editing the column, you cannot change the data type.</p> <p>You can select from the following data types:</p> <p>Integer—32-bit signed integer.</p> <p>UTC—Time, stored as the number of seconds since midnight January 1, 1970.</p> <p>VarChar—Variable size character string, up to 8192 Bytes in length.</p> <p>Incr—32-bit unsigned auto-incrementing integer that can only be updated by the system.</p> <p>Char—Fixed size character string, up to 8192 Bytes in length.</p> <p>Unsigned—32-bit unsigned integer.</p> <p>Boolean—TRUE or FALSE.</p> <p>Real—64-bit signed floating point number.</p> <p>Integer64—64-bit signed integer.</p> <p>Unsigned64—64-bit unsigned integer.</p>
Attributes	<p>Select from the following column attributes:</p> <p>Primary Key—Indicate whether this column is a primary key.</p> <p>No Modify—If this option is selected, users cannot modify data in this column.</p> <p>No Default—If this option is selected, a value must be specified for this column in any <code>INSERT</code> command.</p>



Note: The maximum number of columns in a table is 512, excluding the system-maintained columns. The maximum row size for a table, which is the sum of the length of the columns in the row, is 64 KBytes.



Tip: You can use the **Data View** tab on the *Databases, Tables and Columns* window to view the table data.

Dropping Table Columns

To drop a table column:

1. On the Netcool/OMNIbus Administrator window, click the **System** drop-down list.
2. Click the **Databases** icon. The *Databases, Tables and Columns* window is displayed.
3. Select the table containing the column to drop.
4. Click the **Column Definitions** tab.
5. Select the column to drop.
6. Click the **Drop Column** button. The column is removed from the table.

Configuring User Defined Signals

A signal is an occurrence within the ObjectServer that can be detected and acted upon. Signals comprise part of the automation subsystem and can have triggers attached to them, so that the ObjectServer can automatically respond when a signal is raised. For information about configuring triggers that fire when a signal is raised, see *Configuring Triggers* on page 83.

ObjectServers automatically raise *system signals* when certain changes in the system occur; for example, during system startup or a connection failure. You cannot create or modify system signals.

You can use Netcool/OMNIbus Administrator to create your own *user defined signals*.

For detailed information about system signals and user defined signals, see *Signals and Signal Triggers* on page 211.

To configure user defined signals, click the **Automation** drop-down list; then, click the **User Defined Signals** icon. The *User Defined Signals* window contains a list of the user defined signals in the selected ObjectServer. An example is shown in Figure 20.

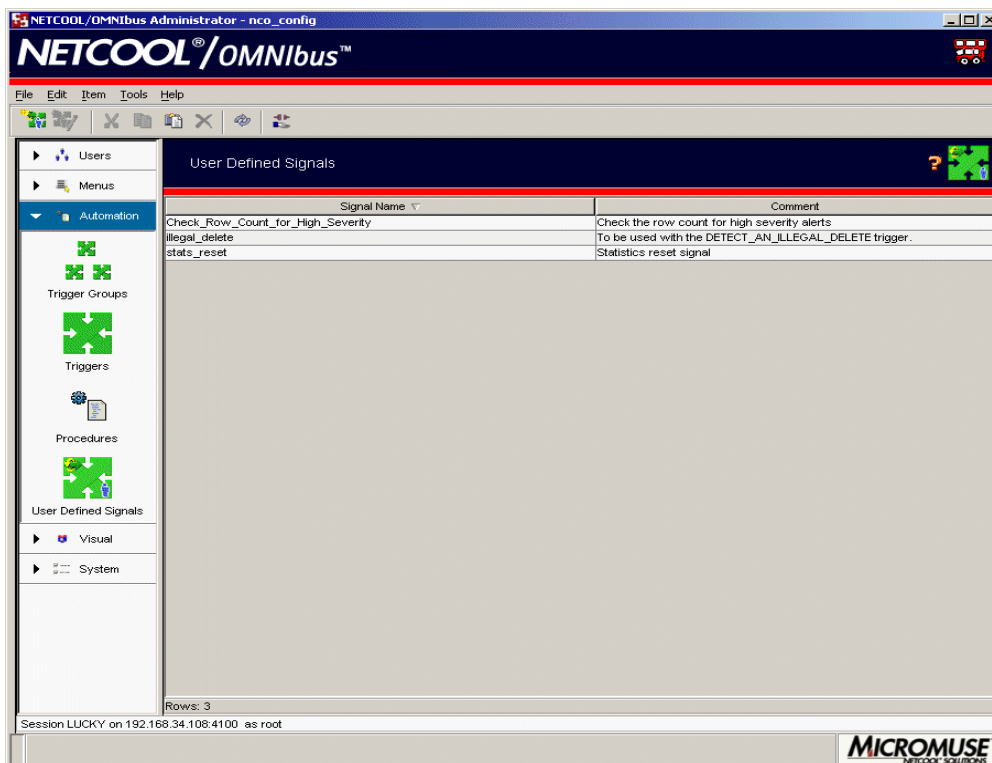


Figure 20: Netcool/OMNIBus Administrator Window - User Defined Signals

Creating and Editing User Defined Signals

To create or edit a user defined signal:

1. On the Netcool/OMNIBus Administrator window, click the **Automation** drop-down list.
2. Click the **User Defined Signals** icon. The *User Defined Signals* window is displayed, which lists the user defined signals in the selected ObjectServer.
3. Click the **Add User Defined Signal** button, or select the user defined signal to edit and click the **Edit User Defined Signal** button. The *User Defined Signal Details* window is displayed.

4. Complete the following window items and click **OK**.

Table 46: User Defined Signal Details Window Items

Window Item	Description
Signal Name	Enter a unique name for this signal. Follow the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68. If you are editing a signal, you cannot change the name.
Comment	Enter a text comment for this signal. For example, you can indicate the trigger that fires when this signal is raised.
Parameters list	<p>Displays the parameters that comprise this user defined signal.</p> <p>Use the Name, Data Type, and Data Length fields to create the parameters. Click the Add Parameter to the list button to add the parameter to the list.</p> <p>You can use the up and down arrows to change the order of a selected parameter. The order in which the parameters appear must match the order that they appear in the trigger's <code>RAISE SIGNAL</code> command.</p> <p>Click the Remove parameter button to remove a selected parameter from the list.</p> <p>See <i>Example User Defined Signal and Trigger</i>, in this section.</p>
Name	Enter the parameter name.
Data Type	<p>Select from the following data types for the indicated parameter.</p> <p>Integer—32-bit signed integer.</p> <p>UTC—Time, stored as the number of seconds since midnight January 1, 1970.</p> <p>Char—Fixed size character string, up to 8192 Bytes in length.</p> <p>Unsigned—32-bit unsigned integer.</p> <p>Boolean—TRUE or FALSE.</p> <p>Real—64-bit signed floating point number.</p> <p>Integer64—64-bit signed integer.</p> <p>Unsigned64—64-bit unsigned integer.</p>
Data Length	For Char data types only, enter the parameter length.
Add parameter to the list button	After completing the Name , Data Type , and (if necessary) Data Length fields, click this button to add the parameter to the parameter list.



Example User Defined Signal and Trigger

This section contains an example user defined signal called `illegal_delete`, as well as the database trigger in which it is used, called `DETECT_AN_ILLEGAL_DELETE`. The database trigger uses the signal to trap deletes that occur outside of standard office hours.

Figure 21 contains a *User Defined Signal Details* window with the `illegal_delete` user defined signal.

Figure 21: Example User Defined Signal

In the `DETECT_AN_ILLEGAL_DELETE` pre-insert database trigger, shown below, the `raise signal` command is shown in bold.

```
create trigger DETECT_AN_ILLEGAL_DELETE
group default_triggers
priority 1
before delete on alerts.status
for each row
begin
    if( ( (hourofday() > 17) and (minuteofhour() > 30) ) or (hourofday() < 9) ) then
        raise signal ILLEGAL_DELETE %user.user_name, Summary;
        cancel;
    end if;
end;
```

This trigger raises the `illegal_delete` user defined signal. Normally, the raised signal would then be detected and acted upon, for example, by another trigger.

Deleting User Defined Signals

To delete a user defined signal:

1. On the Netcool/OMNIbus Administrator window, click the **Automation** drop-down list.
2. Click the **User Defined Signals** icon. The *User Defined Signals* window is displayed, which lists the user defined signals in the selected ObjectServer.
3. Select the user defined signal to delete.
4. Click the **Delete** button. The user defined signal is deleted.



Note: You cannot delete a user defined signal if it is being used by a signal trigger.

Configuring ObjectServer Files

ObjectServer files are user defined storage objects for log or report data. An ObjectServer file has a corresponding file or set of files on the physical file system. You can define ObjectServer file sizes and the number of physical files in a set.

You can use Netcool/OMNIbus Administrator to create ObjectServer files. To write to an ObjectServer file, you must use the `WRITE INTO SQL` command. For example, you can create a trigger that writes an entry in an ObjectServer file each time a user makes a connection to an ObjectServer.

For information about creating triggers, see *Configuring Triggers* on page 83.

To configure ObjectServer files, click the **System** drop-down list; then, click the **Log Files** icon. The *Log Files* window contains a list of the ObjectServer files in the selected ObjectServer. An example is shown in Figure 22.

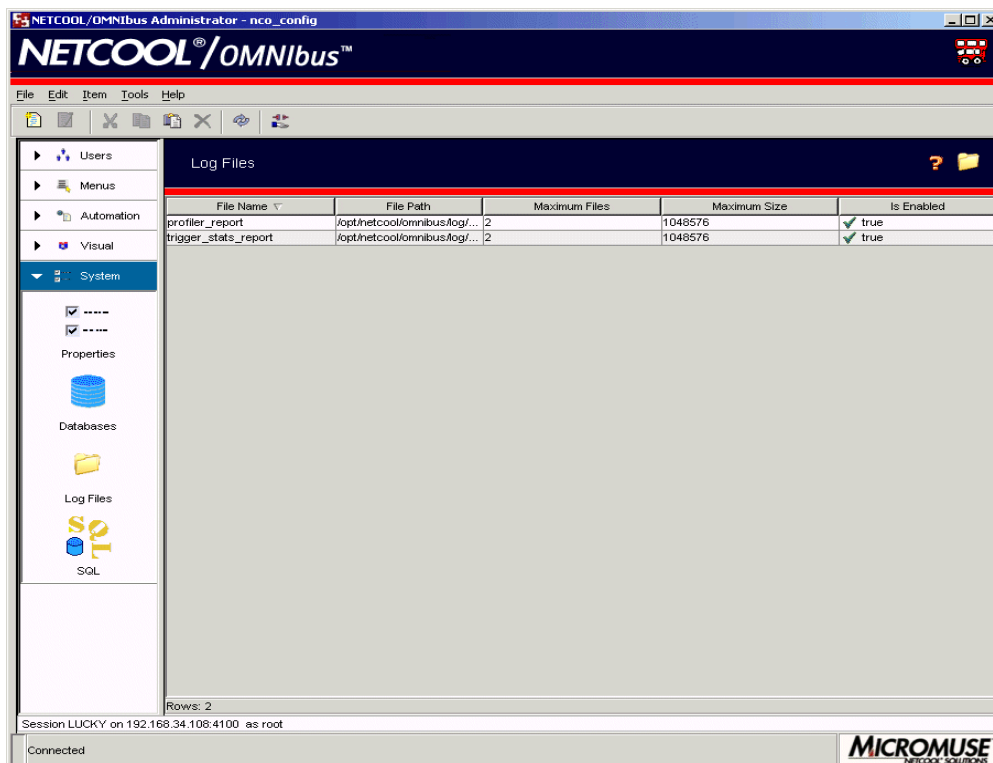


Figure 22: Netcool/OMNibus Administrator Window - ObjectServer Files

ObjectServer File Creation Sequence

Each file in a file set is indicated by a number appended to the file name (or file extension, if there is one). For example, if you create a file named `logfile` in the `/log` directory and specify that its maximum size is 20 KBytes and the maximum number of files in the set is 3, the following sequence of files is created and used:

1. When you click **OK** to create the file, the ObjectServer creates an empty file named `logfile1` in the `/log` directory.
2. The ObjectServer writes data to `logfile1` until it exceeds the maximum file size (20 KBytes).
3. The ObjectServer renames `logfile1` to `logfile2`. It then creates a new `logfile1` and writes to it until it exceeds the maximum size.

4. The ObjectServer renames logfile2 to logfile3 and renames logfile1 to logfile2. It then creates a new logfile1 and writes to it until it exceeds the maximum size.
5. The ObjectServer deletes the oldest file (logfile3). It then renames logfile2 to logfile3 and renames logfile1 to logfile2. It creates a new file named logfile1 and writes to it until it exceeds the maximum size.

This sequence is repeated until the file is altered or dropped.

Creating and Editing ObjectServer Files

To create or edit an ObjectServer file:

1. On the Netcool/OMNIbus Administrator window, click the **System** drop-down list.
2. Click the **Log Files** icon. The *Log Files* window is displayed, which lists all ObjectServer files in the selected ObjectServer.
3. Click the **Add Log File** button, or select the ObjectServer file to edit and click the **Edit Log File** button. The *File Details* window is displayed.
4. Complete the following window items and click **OK**.

Table 47: File Details Window Items (1 of 2)

Window Item	Description
File Name	Enter a unique name for this ObjectServer file. Follow the naming conventions described in <i>ObjectServer Object Naming Conventions</i> on page 68. If you are editing a file, you cannot change the name. Note: This is not the file name as it will be created on the file system. To specify this, use the File Path field, described in this table.
File Path	Enter the full path and file name of the physical file. For example, <code>/opt/netcool/omnibus/log/status.log</code> . Note: A number is automatically appended to the file name on the file system, as described in <i>ObjectServer File Creation Sequence</i> on page 116.
Enabled	Select this check box to enable this ObjectServer file. A disabled file exists on the file system; however, you cannot write to it until it is enabled. You can enter the ObjectServer file information and enable it at a later time.
Truncate File	Clears any information that has been written to the physical file, but does not delete the file. In situations where there is more than one physical file in a set, only the file that is currently being written to on the file system is truncated.

Table 47: File Details Window Items (2 of 2)

Window Item	Description
Max. Size	<p>Select the maximum ObjectServer file size. Additionally, select one of the following from the drop-down list:</p> <ul style="list-style-type: none"> • BYTE • KBYTE • MBYTE • GBYTE <p>Note: The minimum file size is 1 KByte. The maximum file size is 4 GBytes; the operating system may place further restrictions on the maximum size of a single file.</p>
Max. Files	Select the maximum number of ObjectServer files to create.

Deleting ObjectServer Files

To delete an ObjectServer file:

1. On the Netcool/OMNIbus Administrator window, click the **System** drop-down list.
2. Click the **Log Files** icon. The *Log Files* window is displayed, which lists all ObjectServer files in the selected ObjectServer.
3. Select the ObjectServer file to delete.
4. Click the **Delete** button. The ObjectServer file is deleted. The ObjectServer no longer writes information to this file.

Deleting a file deletes the ObjectServer file; it does not delete any of the physical files created in the file system.

Note: You cannot drop a file if it is being used, for example, in a trigger.



2.7 Accessing an ObjectServer Using the SQL Interactive Interface (iSQL)

This section describes how to use iSQL to connect to an ObjectServer and use SQL commands to interact with and configure the ObjectServer.



Note: Only users that are members of a group granted the `ISQL` role can connect to an ObjectServer using the SQL interactive interface. Only users that are members of a group granted the `ISQLWrite` role can update ObjectServer data using the SQL interactive interface. For more information, see *Adding and Removing Role Permissions* on page 57.

Opening iSQL

To open iSQL:

1. On the Netcool/OMNIBus Administrator window, click the **System** drop-down list.
2. Click the **SQL** icon. The **SQL** window is displayed, as shown in Figure 23.

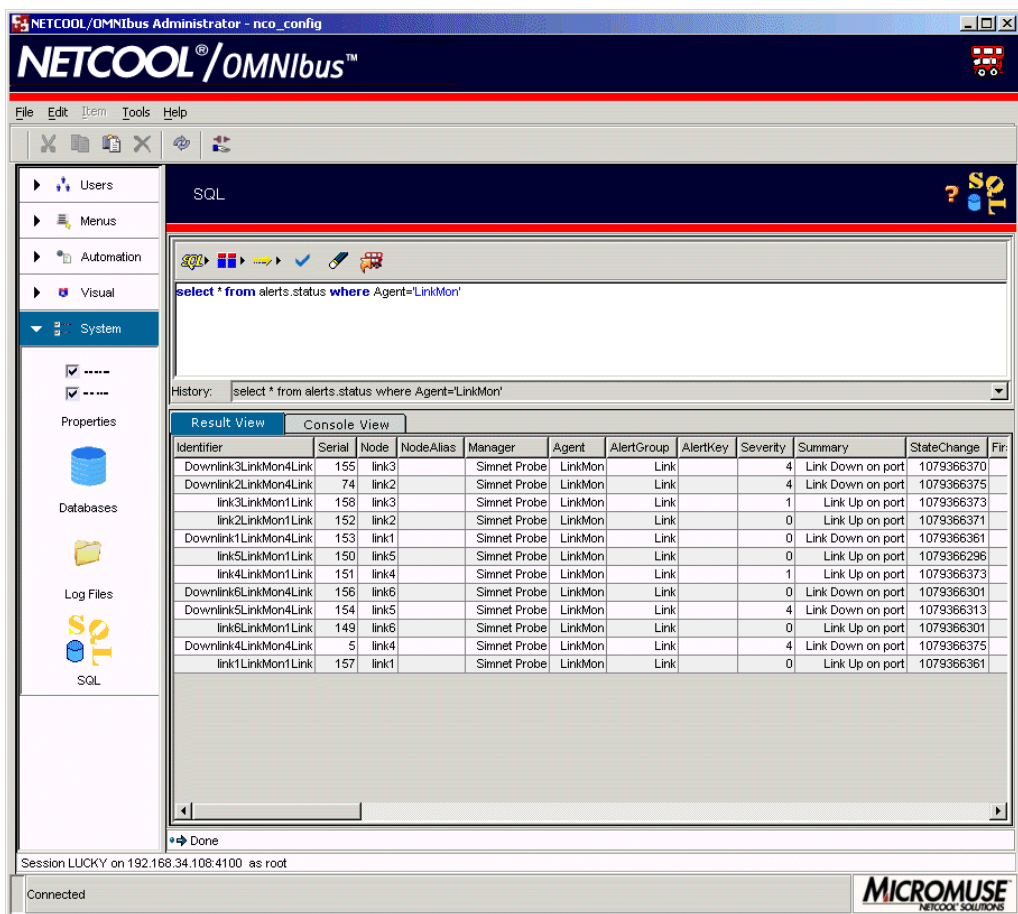


Figure 23: SQL Window

To issue a command, type the command in the text field at the top of the window and click the **Go** button. You can use a semicolon to separate multiple commands. You can also use the helper buttons to facilitate the creation of SQL commands. See *SQL Commands and Variable Expressions in Tools, Automations, and Transient Event Lists* on page 302.

After issuing the command, a visual representation of the table on which you performed the SQL command is displayed on the **Result View** tab. A command history is displayed on the **Console View** tab. You can also click the **History** drop-down arrow to run a previously issued command.

To clear the SQL window, click the **Clear text** button (to the right of the **Submit** button).

For information about the syntax of ObjectServer SQL commands, see *ObjectServer SQL* on page 123.

Chapter 3: ObjectServer SQL

The ObjectServer is the database server at the core of Netcool/OMNIbus. This chapter describes how alerts are stored and managed in the ObjectServer. It also describes the data structures of the ObjectServer and the syntax of ObjectServer SQL. It contains the following sections:

- *Alert Processing in the ObjectServer* on page 124
- *Introduction to ObjectServer SQL* on page 126
- *Using the SQL Interactive Interface* on page 128
- *Storage Structures and Data Definition Language SQL Commands* on page 133
- *Reserved Words* on page 147
- *SQL Building Blocks: Operators, Functions, Expressions, and Conditions* on page 150
- *Data Manipulation Language SQL Commands* on page 164
- *System and Session SQL Commands* on page 174
- *Security SQL Commands* on page 177
- *Procedures* on page 189
- *Automation: Triggers and Trigger Groups* on page 201
- *Automation Examples* on page 220

3.1 Alert Processing in the ObjectServer

The ObjectServer is the in-memory database server at the core of Netcool/OMNIbus. Alert information is forwarded to the ObjectServer from external programs such as probes, monitors, and gateways, stored and managed in database tables, and displayed in the event list.

Figure 24 shows an example Netcool/OMNIbus system architecture.

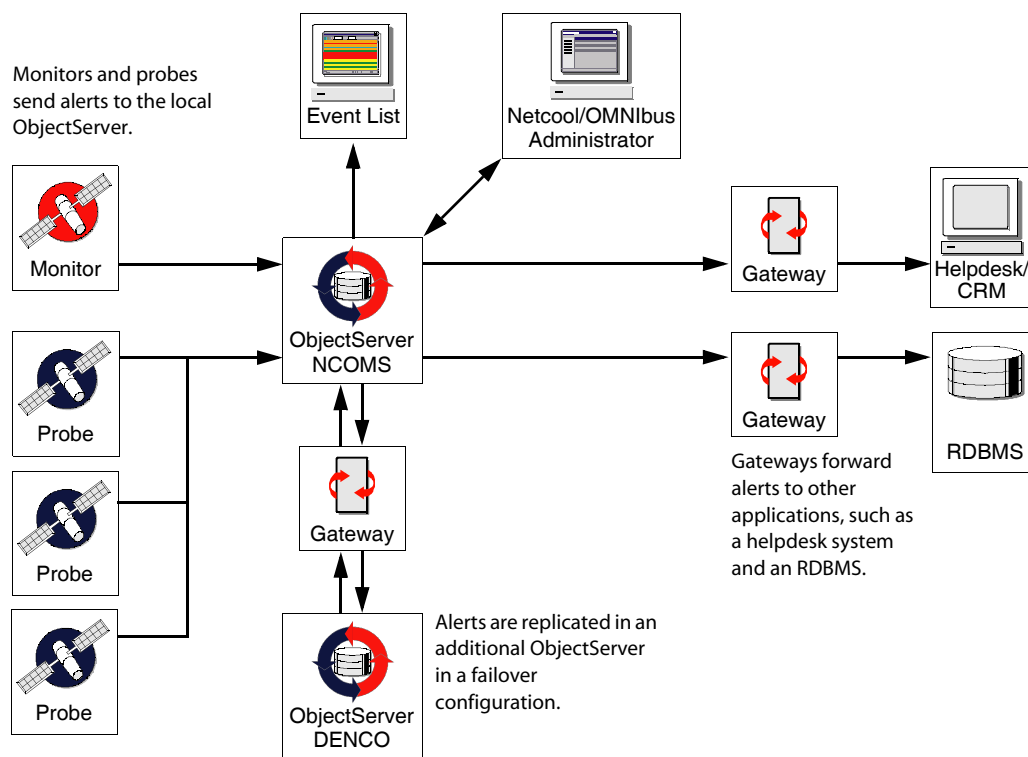


Figure 24: System Architecture

Introduction to Deduplication

A single device may generate the same error repeatedly until the problem is dealt with. The ObjectServer uses *deduplication* to ensure that alert information generated from the same source is not duplicated. Repeated alerts are identified and stored as a single alert to reduce the amount of data in the ObjectServer.

Deduplicating Alerts

Alerts are stored as rows (entries) in the ObjectServer `alerts.status` table. Each alert has an `Identifier` field, which uniquely identifies the problem source. The identifier is generated by the probe according to the specification in the probe rules file, as described in the Netcool/OMNIBus Probe and Gateway Guide.

When an alert is forwarded to the ObjectServer, the `alerts.status` table is searched for a matching `Identifier` field. If no entry with the same identifier is found, a new alert entry is inserted. The entry contains detailed information about the problem. For example, the `FirstOccurrence` field indicates the time that the problem first occurred.

If an entry with the same identifier is found, deduplication occurs. By default, the `LastOccurrence` field of the existing entry is updated with the time of the new alert and the `Tally` field is incremented.

The fields of the `alerts.status` table are described in *Alerts Tables* on page 252.

Additional Alert Processing Capabilities

You can export alert information to other applications through a *gateway*, as described in the Netcool/OMNIBus Probe and Gateway Guide.

A bidirectional ObjectServer Gateway can provide failover support to another ObjectServer. For more information, refer to the Netcool/OMNIBus Installation and Deployment Guide and the guide for the ObjectServer Gateway.

The ObjectServer can also respond automatically to specified alerts. This is called *automation*, and is described in *Automation: Triggers and Trigger Groups* on page 201.

3.2 Introduction to ObjectServer SQL

The ObjectServer provides an SQL interface for defining and manipulating relational database objects such as tables and views. ObjectServer SQL commands include:

- Data Definition Language (DDL) commands to create, alter, and drop database objects
- Data Manipulation Language (DML) commands to query and manipulate data in existing database objects
- System commands to alter the configuration of an ObjectServer
- Session control commands to alter settings in client sessions
- Security commands to control user access to database objects

The ObjectServer also provides procedural language commands, which give you programming constructs for defining actions that will take place when specified incidents occur and conditions that you define are met. You can use procedures and triggers to form automations, enabling you to process alerts automatically.

You can use the SQL interactive interface (called `nc_o_sql` on UNIX and `isql` on Windows), to connect to an ObjectServer and execute ObjectServer SQL commands.



Tip: When entering an SQL command, you must specify the keywords in the order listed in the syntax descriptions. For additional information on syntax notation, see *Typographical Notation* on page 5.

Many of the tasks described in this chapter can also be performed using Netcool/OMNIBus Administrator, described in Chapter 2: *Netcool/OMNIBus Administrator* on page 43.

ObjectServer Naming Conventions

The name of an ObjectServer must consist of 11 or fewer uppercase letters.

The names of the following ObjectServer objects must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (`_`) characters, up to 40 characters in length:

- Databases
- Tables
- Procedures
- Signals
- Triggers
- Trigger groups

- ObjectServer files
- Restriction filters
- Views

User, group, and role names can be any text string enclosed in quotes, up to 64 characters in length.

Names of ObjectServer objects and identifiers are case-sensitive.

SQL keywords are not case-sensitive.

3.3 Using the SQL Interactive Interface

This section describes how to use the SQL interactive interface (called `nco_sql` on UNIX and `isql` on Windows) to connect to an ObjectServer and use SQL commands to interact with and configure the ObjectServer. The SQL interactive interface enables you to perform tasks such as creating a new database table or stopping the ObjectServer.

Many of the tasks described in this chapter can also be performed using Netcool/OMNIBus Administrator, described in Chapter 2: *Netcool/OMNIBus Administrator* on page 43.



Note: Only users that are members of a group granted the `ISQL` role can connect to an ObjectServer using the SQL interactive interface. Only users that are members of a group granted the `ISQLWrite` role can modify ObjectServer data using the SQL interactive interface. The `GRANT` command and system permissions are described in *The GRANT Command* on page 183. Default groups and roles are described in the Netcool/OMNIBus Installation and Deployment Guide.

Starting the SQL Interactive Interface

Use the SQL interactive interface to connect to an ObjectServer as a specific user. For example:

Table 48: Starting the SQL Interactive Interface

On...	Enter the following command...
UNIX	<code>\$OMNIHOME/bin/nco_sql -server servername -user username</code>
Windows	<code>%OMNIHOME%\bin\redist\isql.exe -S servername -U username</code>

In these commands, *servername* is the name of the ObjectServer and *username* is a valid user name. If you do not specify an ObjectServer name, the default name `NCOMS` is used. If you do not specify a user name, the default is the user running the command. You must enter a valid password for the user, either when prompted or by specifying the `-password` command line option (`-P` on Windows).



Tip: On Windows you must specify the ObjectServer name and user name.



Warning: It is not recommended to specify the password on the command line because this makes the password visible. If not specified, the user is prompted for the password.

The command line options for the SQL interactive interface are described in Table 49.

Table 49: nco_sql Command Line Options

Option	Description
-help	Displays help information about the command line options and exits.
-nosecure	When specified, login information is not encrypted when it is transmitted between components. You must use this option to connect to ObjectServers and gateways for releases previous to Netcool/OMNIBus version 3.5. See <i>Using the Secure SQL Interactive Interface</i> on page 132 for more information.
-password <i>password</i> -P <i>password</i> on Windows	Specifies the password for the user. Warning: It is not recommended to specify the password on the command line because this makes the password visible. If not specified, the user is prompted for the password.
-secure	When specified, login information is automatically encrypted when it is transmitted between components. This is the default for releases of Netcool/OMNIBus version 3.5 and above. See <i>Using the Secure SQL Interactive Interface</i> on page 132 for more information.
-server <i>servername</i> -S <i>servername</i> on Windows	Specifies the name of the ObjectServer to which to connect. The default is NCOMS.
-user <i>username</i> -U <i>username</i> on Windows	Specifies the name of a Netcool/OMNIBus user. The default is the user running the command. Note: The SQL interactive interface does not allow spaces in user names.

Once connected, you can enter ObjectServer SQL commands, as described in the next section.

Executing SQL Commands in the SQL Interactive Interface

After connecting with a user name and password, a numbered prompt is displayed:

1>

Enter ObjectServer SQL commands at the prompt. When entering text:

- Commands can be split over multiple lines.
- Commands are not processed until you enter the keyword `go` in lowercase letters at the beginning of a new line and press return.

- Multiple commands, separated by a semicolon, can be executed with a single `go`.
- To cancel a command, enter `reset` at the beginning of a new line or `Control + C` anywhere on a line. Any commands that have not been executed are discarded.



Example SQL Interactive Interface Session on UNIX

```
nco_sql -server OS1 -username myuser -password mypass
```

```
1> select * from alerts.status;  
2> go
```

The results of the command are displayed.

To run the default editor (as defined by the `EDITOR` environment variable) in `nco_sql`, enter `vi` at the beginning of a new line.

To read in a file, enter `:r filename` at the beginning of a new line. Do not include the `go` command in the file. Instead, enter the `go` command at the beginning of a new line.

To run an operating system command, enter `!!` followed by the command (for example, `!!ls`) at the beginning of a new line.



Tip: You can enter a command of up to 4094 characters.

Specifying Paths in the SQL Interactive Interface

Some SQL commands require you to enter path names. For example, on UNIX platforms, you can create a file using the following command:

```
create file TESTFILE01 '/tmp/testfile01';
```

On Windows platforms, you must escape the backslash character or it will not be interpreted correctly. For example, you can create an ObjectServer file on Windows using the following command:

```
create file TESTFILE01 'c:\\temp\\testfile01.txt';
```

You can also use the equivalent UNIX path on Windows platforms. The following UNIX path is also interpreted correctly on Windows:

```
create file TESTFILE01 'c:/temp/testfile01.txt';
```

Using Text Files for Input and Output

You can redirect text files using the SQL interactive interface. This is useful when you need to perform repetitive tasks. The text file must contain only SQL commands and be terminated with the `go` keyword; otherwise, the commands will not be processed.

For example, to execute the SQL commands in a text file named `my_SQL_file.txt` from a UNIX command line, enter the following command:

```
nco_sql -server OS1 -username myuser -password mypass < my_SQL_file.txt
```

You can also direct the output to a file, for example:

```
nco_sql -server OS1 -username myuser -password mypass< my_SQL_file.txt > output.txt
```

Exiting the SQL Interactive Interface

To exit the SQL interactive interface:

Table 50: Exiting the SQL Interactive Interface

On...	Enter the following command...
UNIX	Press Control + D or enter <code>quit</code> or <code>exit</code> at the beginning of a new line.
Windows	Enter <code>quit</code> or <code>exit</code> at the beginning of a new line.

You are disconnected from the ObjectServer and returned to the operating system prompt.

Using the Secure SQL Interactive Interface

When an ObjectServer runs in secure mode, it requires clients such as probes, desktops, gateways, and the SQL interactive interface to connect using valid user names and passwords. The login information is automatically encrypted when it is transmitted between components to make snooping ineffective. Versions of Netcool/OMNIBus previous to 3.5 do not support this transmission encryption.

The SQL interactive interface runs in secure mode unless you specify the `-nosecure` option. You can use the `-nosecure` option to connect to versions of the ObjectServer previous to 3.5.

When you run the SQL interactive interface in secure mode, it uses the `nco_get_login_token` utility to encrypt its login data for transmission. The utility produces a token that can be used only once to log in to the ObjectServer. The token has a time limit after which it expires and becomes invalid.

Encrypting Passwords in UNIX `nco_sql` Scripts

You can use the `nco_sql_crypt` utility to encrypt plain text login passwords so that they are not exposed in UNIX scripts that run `nco_sql`. Passwords encrypted using `nco_sql_crypt` are decrypted by the ObjectServer when the connection is made.

To encrypt a plain text password:

1. Enter the following:

```
$OMNIHOME/bin/nco_sql_crypt password
```

In this command, *password* is the unencrypted form of the password.

The `nco_sql_crypt` utility displays an encrypted version of the password.

2. Copy the encrypted password into the script.

3.4 Storage Structures and Data Definition Language SQL Commands

The ObjectServer stores, manages, and processes alert data collected by external applications such as probes, monitors, and gateways. The default storage structures (objects) are created according to SQL definition files.

This section describes how you can use Data Definition Language (DDL) commands to create, modify, and drop ObjectServer objects.

Table 51: ObjectServer Objects and Associated DDL Commands

ObjectServer Object	Allowed DDL Commands	Described on...
DATABASE	CREATE DATABASE DROP DATABASE	page 134 page 134
TABLE	CREATE TABLE ALTER TABLE DROP TABLE	page 135 page 138 page 139
VIEW	CREATE VIEW DROP VIEW	page 140 page 141
RESTRICTION FILTER	CREATE RESTRICTION FILTER DROP RESTRICTION FILTER	page 142 page 143
FILE	CREATE FILE ALTER FILE DROP FILE	page 143 page 145 page 145

You can use the Data Manipulation Language (DML) commands described in *Data Manipulation Language SQL Commands* on page 164 to query and manipulate data in existing database objects.

Security commands enable you to control user access to ObjectServer objects. These are described in *Security SQL Commands* on page 177.

Databases

A database is a structured collection of data organized for quick access to desired information. A relational database uses *tables* as logical containers to store this data in rows and columns.

Creating a Database

Use the `CREATE DATABASE` command to create a database.



Syntax

```
CREATE DATABASE database_name;
```

The database name must be unique within the ObjectServer and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

A database is always persistent.



Example

```
create database mydb;
```

Dropping a Database

Use the `DROP DATABASE` command to drop an existing database.



Syntax

```
DROP DATABASE database_name;
```

You cannot drop a database if it contains any objects. You cannot drop the `security` or `catalog` databases, described in *System-Initialized Databases* on page 134.



Example

```
drop database mydb;
```

System-Initialized Databases

When you initialize an ObjectServer as described in *Creating the ObjectServer* on page 11, the following databases are created:

Table 52: System-Initialized Databases (1 of 2)

Database Name	Type of Database	Description
<code>security</code>	System	Contains information about the security system, including users, roles, groups, and permissions.

Table 52: System-Initialized Databases (2 of 2)

Database Name	Type of Database	Description
catalog	System	Contains metadata about ObjectServer objects.
alerts	User	Contains alert status information, forwarded to the ObjectServer by probes and gateways.
service	User	Used to support Netcool/ISMs.
custom	User	Can be used for tables added by users.
persist	System	Records internal ObjectServer state information.
transfer	System	Used internally by the ObjectServer bidirectional gateway to synchronize security information between ObjectServers.
master	User	Used for compatibility with prior releases of Netcool/OMNIBus. Tables in the <code>master</code> database also support the desktop ObjectServer architecture described in the Netcool/OMNIBus Installation and Deployment Guide.
tools	User	Used for compatibility with prior releases of Netcool/OMNIBus.

The tables in these databases are described in Appendix A: *ObjectServer Tables* on page 251.



Note: The ObjectServer maintains system databases; you can view, but not modify, the data in them.

Tables

The ObjectServer's main storage structure is the table. A table has a fixed number of data-typed columns. The name of each column is unique to the table. A table contains zero or more rows of data in the format defined by the table's column list.

The fully-qualified table name includes the database name and the table name, separated by a period. For example, the `status` table in the `alerts` database is identified as `alerts.status`.

Creating a Table

Use the `CREATE TABLE` command to create a table.



Syntax

```
CREATE TABLE [database_name.] table_name
  PERSISTENT | VIRTUAL
  (column_name data_type [ PRIMARY KEY | NODEFAULT | NOMODIFY | HIDDEN ], ...
  [, PRIMARY KEY(column_name,...) ] );
```

The table name must be unique within the database and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

The storage type is either PERSISTENT or VIRTUAL. A persistent table is recreated, complete with all its data, when the ObjectServer restarts. A virtual table is recreated with the same table description, but without any data, when the ObjectServer restarts.

When you define columns, in addition to the column name, you must specify the data type and optional properties, as described next.

Column Data Types

Each column value in the ObjectServer has an associated data type. The data type determines how the ObjectServer processes the data in the column. For example, the plus operator (+) adds integer values or concatenates string values, but does not act on boolean values. The data types supported by the ObjectServer are listed in Table 53:

Table 53: ObjectServer Data Types (1 of 2)

SQL Type	Description	Default Value	ObjectServer ID for Data Type
INTEGER	32 bit signed integer.	0	0
INCR	32 bit unsigned auto-incrementing integer. Applies to table columns only, and can only be updated by the system.	1	5
UNSIGNED	32 bit unsigned integer.	0	12
BOOLEAN	TRUE or FALSE .	FALSE	13
REAL	64 bit signed floating point number.	0 . 0	14
TIME	Time, stored as the number of seconds since midnight January 1, 1970. This is the Coordinated Universal Time (UTC) international time standard.	Thu Jan 1 01:00:00 1970	1

Table 53: ObjectServer Data Types (2 of 2)

SQL Type	Description	Default Value	ObjectServer ID for Data Type
CHAR(<i>integer</i>)	Fixed size character string, <i>integer</i> characters long (8192 Bytes is the maximum). The <code>char</code> type is identical in operation to <code>varchar</code> , but performance is better for mass updates that change the length of the string.	' '	10
VARCHAR(<i>integer</i>)	Variable size character string, up to <i>integer</i> characters long (8192 Bytes is the maximum). The <code>varchar</code> type uses less storage space than the <code>char</code> type and the performance is better for deduplication, scanning, insert, and delete operations.	' '	2
INTEGER64	64 bit signed integer.	0	16
UNSIGNED64	64 bit unsigned integer.	0	17



Note: You can only display columns of type CHAR, VARCHAR, INCR, INTEGER, and TIME in the event list. Do not add columns of any other type to the `alerts.status` table. The columns in the `alerts.status` table are described in *alerts.status Table* on page 252.

Column Properties

The optional column properties are described in Table 54.

Table 54: Column Properties

Column Property	Description
PRIMARY KEY	The column is created as a primary key. The primary key column or columns uniquely identify each row. A primary key column must have a default value and cannot be hidden.
NODEFAULT	The value of this column must be specified in the initial <code>INSERT</code> command.
NOMODIFY	The value of this column cannot be changed after the initial <code>INSERT</code> command.
HIDDEN	Data is not written to or read from a hidden column when inserting or selecting a row. The column name must be specified explicitly to insert data into or select from it. Hidden columns contain system information or information that is not applicable to most users.

As an alternative to using the `PRIMARY KEY` column property, you can specify one or more columns that make up the primary key by including a comma-separated list of columns in a `PRIMARY KEY` clause following the column definitions.



Tip: You can use the `PRIMARY KEY` column property, the `PRIMARY KEY` clause, or a combination of both, to indicate which columns make up the primary key.

The maximum number of columns in a table is 512, excluding the system-maintained columns. The maximum row size for a table, which is the sum of the length of the columns in the row, is 64 KBytes.



Example

```
create table mydb.mytab persistent
  (col1 integer primary key, col2 varchar(20));
```

Altering a Table

Use the `ALTER TABLE` command to change the characteristics of an existing table and its columns.



Syntax

```
ALTER TABLE [database_name.] table_name
  ADD [COLUMN] column_name data_type [ NODEFAULT | NOMODIFY | HIDDEN ]
  DROP [COLUMN] column_name
  ALTER [COLUMN] column_name SET NOMODIFY { TRUE | FALSE }
  ALTER [COLUMN] column_name SET HIDDEN { TRUE | FALSE }
  ALTER [COLUMN] column_name SET NODEFAULT { TRUE | FALSE };
```

You cannot alter system tables, described in *System Tables* on page 139.

To add or drop columns from an existing table use the `ADD COLUMN` and `DROP COLUMN` settings, respectively. The syntax for a new column definition is described in *Creating a Table* on page 135.

You cannot add primary keys to an existing table.

You cannot drop a column if the column is a primary key. If you drop a column that has views, triggers, procedures, or restriction filters that depend on it, these dependent objects are also deleted, and a warning is written to the ObjectServer log file.

To alter the `NOMODIFY`, `HIDDEN`, and `NODEFAULT` properties of an existing column set the appropriate property to `TRUE` or `FALSE` using the `ALTER COLUMN` setting. A primary key column must have a default value and cannot be hidden.

You can change more than one setting in a single `ALTER TABLE` command.



Example

```
alter table mytab add col3 real;
```

Dropping a Table

Use the DROP TABLE command to drop an existing table.



Syntax

```
DROP TABLE [database_name.] table_name;
```

You cannot drop a table if it is referenced by other objects, such as triggers, or if it contains any data. You cannot drop system tables, described in *System Tables* on page 139.



Example

To delete all rows of a table:

```
delete from mytab;
```

To drop the table:

```
drop table mytab;
```

System Tables

System tables are special tables maintained by the ObjectServer that contain metadata about ObjectServer objects. System tables are identified by the database name `catalog`. For example, the `catalog.columns` table contains metadata about all the columns of all the tables in the ObjectServer.

You can view information in the system tables using the SELECT and DESCRIBE commands, but you cannot add, modify, or delete system tables or their contents using ObjectServer SQL. For descriptions of system tables and their contents, see *System Catalog Tables* on page 262.

Table DML Commands

The following DML commands are provided for tables:

```
SELECT  
INSERT  
UPDATE  
DELETE
```

These commands are described in *Data Manipulation Language SQL Commands* on page 164.

Views

A view is a virtual table projected from selected rows and columns of a real table, allowing subsets of table data to be easily displayed and manipulated. For example, if you want a group of users to see only certain relevant columns in a table, you can create a view that contains only those columns. You can also have *virtual columns*, composed using expressions on columns in the underlying table. For more information on expressions, see *Expressions* on page 161.

Creating a View

Use the `CREATE VIEW` command to create a view.



Syntax

```
CREATE [ OR REPLACE ] VIEW [database_name.]view_name
[ (view_column_name,...) ]
[ TRANSIENT | PERSISTENT ]
AS SELECT_cmd;
```

If a view may already exist with the same name as the one you want to create, use the optional `OR REPLACE` keywords. If the view exists, it is replaced by the one you are creating. If the view does not exist, a new one is created.

The view name must be unique within the database and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126. The following additional restrictions apply to the creation of views:

- If you do not specify a database name, the view is created in the `alerts` database.
- You cannot create a view on a view.
- You cannot create a view on any table in the `catalog` database.

You can specify either a `TRANSIENT` or `PERSISTENT` storage type, depending on your data storage requirements. A transient view is destroyed when the client that created it disconnects. A persistent view is mirrored on disk. When the ObjectServer restarts, the view is recreated.

The *SELECT_cmd* is any `SELECT` command (including aggregate `SELECT` commands) as described in *Viewing Table Data: The SELECT Command* on page 167, with the following restrictions:

- You must specify all of the column names explicitly, rather than using a wildcard (*), in the selection list.
- If you include virtual columns, you cannot update them.
- If you do not specify a database name, the default is `alerts`.

- You cannot specify a GROUP BY clause.
- You can only have a subquery containing a WHERE clause in an aggregate SELECT statement.
- You cannot use virtual columns in an aggregate SELECT statement.
- If you create an aggregate view, you cannot perform an aggregate SELECT on it.
- If you create an aggregate view, you cannot perform an UPDATE or DELETE on it.



Example

```
create view alerts.myview persistent as select Severity, LastOccurrence, Summary
from alerts.status order by Severity, LastOccurrence;
```

Dropping a View

Use the DROP VIEW command to drop an existing view.



Syntax

```
DROP VIEW [database_name.]view_name;
```

If you do not specify a database name, the view is dropped from the `alerts` database.

You cannot drop a view if it is referenced by other objects.



Example

```
drop view myview;
```

View DML Commands

The following DML commands are provided for views:

```
SELECT
UPDATE
DELETE
```

These commands are described in *Data Manipulation Language SQL Commands* on page 164.

Restriction Filters

A restriction filter provides a way to restrict the rows that are displayed when a user views a table. Once the filter has been assigned to a user or group, the filter controls the data that can be displayed and modified from client applications, and modified in `INSERT`, `UPDATE`, and `DELETE` commands. Only rows meeting the criteria specified in the filter condition are returned.

For information on assigning restriction filters to users and groups, see *The ALTER USER Command* on page 178 and *The ALTER GROUP Command* on page 180. You can only assign one restriction filter per table to a user or a group. If multiple filters apply to a user or group, the resulting data is a combination of all applicable filters for the user or group.

Creating a Restriction Filter

Use the `CREATE RESTRICTION FILTER` command to create a restriction filter.



Syntax

```
CREATE [ OR REPLACE ] RESTRICTION FILTER filter_name
ON [database_name.] table_name WHERE condition;
```

If a restriction filter may already exist with the same name as the one you want to create, use the optional `OR REPLACE` keywords. If the restriction filter exists, it is replaced by the one you are creating. If the restriction filter does not exist, a new one is created.



Note: If you are replacing an existing filter only the *condition* can be changed. A filter can be replaced even if it has been assigned to any users or groups.

The restriction filter name must be unique and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

The *condition* is an expression or expressions that returns a subset of rows of the table. Conditions are described in *Conditions* on page 162.

A restriction filter is always persistent.



Example

```
create restriction filter myfilter on alerts.status where Severity = 5;
```

You can also create restriction filters in the Filter Builder.

Dropping a Restriction Filter

Use the `DROP RESTRICTION FILTER` command to drop an existing restriction filter.



Syntax

```
DROP RESTRICTION FILTER filter_name;
```

You cannot drop a restriction filter if it has been assigned to any users or groups.



Example

```
drop restriction filter myfilter;
```

Restriction Filter DML Commands

Restriction filters are automatically assigned in the following DML commands:

```
SELECT
INSERT
UPDATE
DELETE
```

These commands are described in *Data Manipulation Language SQL Commands* on page 164.

Files

An ObjectServer file provides a way to log or report information about ObjectServer events. To log information to an ObjectServer file, use the `WRITE INTO` command, described in *Logging to Files: The WRITE INTO Command* on page 171. For example, you might want to add a record each time a particular user makes a connection to the ObjectServer.

An ObjectServer file is a logical file, which has a corresponding file or set of files on the physical file system.

Creating a File

Use the `CREATE FILE` command to create an ObjectServer file.



Syntax

```
CREATE [ OR REPLACE ] FILE file_name 'path_to_physical_file'
  [ MAXFILES number_files ]
  [ MAXSIZE file_size { GBYTES | MBYTES | KBYTES | BYTES } ];
```

If an ObjectServer file may already exist with the same name as the one you want to create, use the optional `OR REPLACE` keywords. If the ObjectServer file does not exist, a new one is created. If the ObjectServer file exists, it is replaced by the one you are creating.



Note: If you do not use the `OR REPLACE` keywords, you must specify a physical file that does not already exist. If you use the `OR REPLACE` keywords, and the physical file already exists, the physical file is overwritten if there is no ObjectServer file associated with it.

The file name must be unique and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

The *path_to_physical_file* is the full path and name of the corresponding file on the physical file system, for example, `/log/out.log`. On Windows platforms, you must escape the backslash character (`\`) or use the equivalent UNIX path (for example: `c:\\tmp\\testfile.txt` or `c:/tmp/testfile.txt`).



Note: A number, starting with 1 and incremented depending on the number of files in the file set, is always appended to the specified file name (or file extension if there is one).

You can optionally set `MAXFILES` to specify the number of files in the file set. The default is 1.

If you set `MAXFILES` to a value greater than 1, when the first file exceeds the maximum size, a new file is created. When that file exceeds the maximum size, another new file is created and the process is repeated until the maximum number of files in the set is reached. Then the oldest file is deleted and the process repeats.

You can optionally set `MAXSIZE` to specify the maximum file size. After a record is written to the file that meets or exceeds that size, a new file is created. The default setting is 0. If set to 0, there is no maximum file size, and therefore the file set always consists of one file.

The minimum file size is 1 KByte. The maximum size is 4 GBytes.

If the ObjectServer is restarted, new data is appended to the existing file.



Example

```
create file logit '/log/logfile'  
maxfiles 3  
maxsize 20 KBytes;
```

The following sequence of files are created and used:

1. When the command is executed, the ObjectServer creates an empty file named `logfile1` in the `/log` directory.
2. The ObjectServer writes data to `logfile1` until it exceeds the maximum file size (20 KBytes).
3. The ObjectServer renames `logfile1` to `logfile2`. It then creates a new `logfile1` and writes to it until it exceeds the maximum size.
4. The ObjectServer renames `logfile2` to `logfile3` and renames `logfile1` to `logfile2`. It then creates a new `logfile1` and writes to it until it exceeds the maximum size.
5. The ObjectServer deletes the oldest file (`logfile3`). It then renames `logfile2` to `logfile3` and renames `logfile1` to `logfile2`. It creates a new file named `logfile1` and writes to it until it exceeds the maximum size.

This sequence is repeated until the file is altered or dropped.

Altering a File

Use the `ALTER FILE` command to change the configuration of an existing ObjectServer file.



Syntax

```
ALTER FILE file_name
  TRUNCATE |
  SET ENABLED { TRUE | FALSE };
```

The `TRUNCATE` setting clears any information that has been written to the physical file. When there is more than one physical file, the file that is currently being written to is truncated; the other files in the set are deleted.

The `ENABLED` setting turns logging on and off. If `TRUE`, a `WRITE INTO` command writes data to the file. If `FALSE`, `WRITE INTO` commands are ignored and nothing is written to the file. Disabling a file is useful when want to stop logging temporarily but do not want to discard the file you have configured.



Example

```
alter file logit truncate;
```

Dropping a File

Use the `DROP FILE` command to drop an existing ObjectServer file.



Syntax

```
DROP FILE file_name;
```

Dropping a file deletes the ObjectServer file; it does not delete any of the physical files created in the file system.

You cannot drop a file if it is being used, for example, in a trigger.



Example

```
drop file logit;
```

File DML Command

The `WRITE INTO DML` command is provided for files. This command is described in *Data Manipulation Language SQL Commands* on page 164.

3.5 Reserved Words

In the ObjectServer, certain words are reserved as SQL or ObjectServer keywords. You are not allowed to use these reserved words as object names in ObjectServer SQL:

Table 55: ObjectServer Reserved Words (1 of 3)

ADD	AFTER	ALL	ALTER	AND
ANY	APR [IL]	ARGUMENTS	ARRAY	AS
ASC	ASSIGN	AUG [UST]	AUTHORIZE AUTHORISE	AVERAGE AVG
BACKUP	BEFORE	BEGIN	BETWEEN	BIDIRECTIONAL
BINARY	BIND	BOOL [EAN]	BREAK	BY
CACHE	CALL	CANCEL	CASE	CHAR [ACTER]
CHECK	CHECKPOINTING	COLUMN	COMMENT	COMMIT
CONN [ECTION]	COUNT	CREATE	CURRENT	DATABASE
DATE [TIME]	DEBUG	DEC [EMBER]	DECLARE	DEFERRED
DELAYED	DELETE	DESC	DESCENT	DESCRIBE
DETACHED	DISABLE	DIST	DISTINCT	DO
DOUBLE	DROP	EACH	EDGE	ELSE
ELSEIF	EMPTY	END	ENCRYPTED	EVALUATE
EVENT	EVERY	EXECUTABLE	EXEC [UTE]	EXTENSION
EXTERNAL	FALSE	FANP	FEB [RUARY]	FILE
FILTER	FLUSH	FOR	FORMAT	FRI [DAY]
FROM	FULL	GET	GRANT	GROUP
HARD	HAVING	HIDDEN	HOST	HOURS
ID	IDUC	IF	IMMEDIATE	IN
INCLUDING	INCR	INCREMENT	INITIAL	INSERT
INT [EGER]	INT [EGER] 64	INTO	ISQL	JAN [UARY]

Table 55: ObjectServer Reserved Words (2 of 3)

JOIN	JUL [Y]	JUN [E]	LEAVE	LIKE
LIMIT	LINK	LOAD	LOCK [PH]	LOGIN
MAR [CH]	MAX	MAXFILES	MAXSIZE	MAY
MEMSTORE	MESSAGE	METRIC	MIN	MINUTES
MON [DAY]	NAMING	NEXT	NO	NODEFAULT
NOMODIFY	NOT	NOTIFY	NOV [EMBER]	OCT [OBER]
OF	ON	ONCE	ONLY	OPTION
OR	ORDER	OUT	PAM	PASSWORD
PERSISTENT	PRIMARY	PRIORITY	PRIVILEGE	PROCEDURE
PROP [S]	PROTECT	PUBLISH	QUERY	RAISE
REAL	REGISTER	REINSERT	REMOVE	REPLACE
RESTRICTION	RESYNC	RETRY	REVOKE	ROLE
ROW	ROWOF	SAT [URDAY]	SAVE	SECONDS
SELECT	SELF	SEND	SEP [TEMBER]	SESSION
SET	SHORT	SHOW	SIGNAL	SKIP
SOFT	SQL	STATEMENT	STORE	SUBSCRIBE
SUM	SUN [DAY]	SVC	SYNC	SYSTEM
TABLE	TEMPORAL	TEMP [ORARY]	THEN	THU [RSDAY]
TIME	TO	TOKEN	TOP	TRANS [ACTION]
TRANSIENT	TRIGGER	TRUE	TRUNCATE	TUE [SDAY]
TYPEOF	UNIDIRECTIONAL	UNION	UNIQUE	UNLOAD
UNREGISTER	UNSIGNED	UNSIGNED64	UNSUBSCRIBE	UNTIL
UPDATE	UPDATING	USE	USER	UTC
VALUES	VARCHAR [ACTER]	VERBOSE	VIA	VIEW

Table 55: ObjectServer Reserved Words (3 of 3)

VIRTUAL	WAIT	WED [NESDAY]	WHEN	WHERE
WITH	WORK	WRITE	YES	XST

3.6 SQL Building Blocks: Operators, Functions, Expressions, and Conditions

The following building blocks enable you to manipulate data in ObjectServer SQL commands:

- Operators
- Functions
- Expressions
- Conditions

Operators

You can use *operators* to compute values from data items. An operator processes (adds, subtracts, and so on) a data item or items. The data items on which the computation is performed are *operands*. Together operators and operands form *expressions*, described in more detail on page 161. In the expression $7 + 3$, the plus symbol (+) is the operator and 7 and 3 are operands.

Operators can be unary or binary. Unary operators act on only one operand. For example, the minus (-) operator can be used to indicate negation. Binary operators act on two operands. For example, the same minus (-) operator can be used to subtract one operand from another.

Some operators, such as the plus (+) operator, are polymorphic; they can be assigned a different meaning in different contexts. For example, you can use the plus (+) operator to add two numbers (7+3) or to concatenate two strings ('The ObjectServer ' + 'started').

Operators used in ObjectServer SQL are divided into the following categories:

- Math and String Operators
- Binary Comparison Operators
- List Comparison Operators
- Logical Operators

Each of these categories and their corresponding operators are described in the following sections.

Operator precedence is described on page 157.

Math and String Operators

You can use math operators to add, subtract, multiply, and divide numeric operands in expressions. Table 56 describes the math operators supported by the ObjectServer.

Table 56: Math Operators

Operator	Description	Example
+ -	Unary operators indicating a positive or negative operand.	<code>SELECT * FROM london.status WHERE Severity = -1;</code>
* /	Binary operators used to multiply (*) or divide (/) two operands.	<code>SELECT * FROM london.status WHERE Tally * Severity > 10;</code>
+ -	Binary operators used to add (+) or subtract (-) two operands.	<code>SELECT * FROM london.status WHERE Severity = Old_Severity - 1;</code>

You can use string operators to manipulate character strings (VARCHAR and CHAR data types). Table 57 describes the string operator supported by the ObjectServer.

Table 57: String Operator

Operator	Description	Example
+	Binary operator used to concatenate two strings.	<code>UPDATE mydb.mystatus SET Location = Node + NodeAlias;</code>

Binary Comparison Operators

You can use binary comparison operators to compare numeric and string values for equality and inequality. Table 58 describes the comparison operators supported by the ObjectServer.

Table 58: Comparison Operators (1 of 2)

Operator	Description	Example
=	Tests for equality.	<code>SELECT * FROM london.status WHERE Severity = 3;</code>
!= <>	Tests for inequality.	<code>SELECT * FROM london.status WHERE Severity <> 1;</code>

Table 58: Comparison Operators (2 of 2)

Operator	Description	Example
< > <= >=	Tests for greater than (>), less than (<), greater than or equal to (>=) or less than or equal to (<=). These operators perform case-sensitive string comparisons. In standard ASCII case-sensitive comparisons, uppercase letters come before lowercase letters.	<pre>SELECT * FROM london.status WHERE Severity > 5;</pre>
%= %!= %<>	Tests for equality (%=) or inequality (%!=, %<>) between strings, ignoring case. To be equal, the strings must contain all of the same characters, in the same order, but they do not need to have the same capitalization.	<pre>SELECT * FROM london.status WHERE Location %= 'New York';</pre>
%< %> %<= %>=	Compares the lexicographic relationship between two strings, ignoring case. This comparison determines whether strings come before (%<) or after (%>) other strings alphabetically. You can also find strings which are less than or equal to (%<=) or greater than or equal to (%>=) other strings. For example, aaa comes before AAB because alphabetically aaa is less than (comes before) AAB when the case is ignored.	<pre>SELECT * FROM london.status WHERE site_code %< 'UK3';</pre>
[NOT] LIKE	The LIKE operator performs string comparisons. The string following the LIKE operator, which can be the result of a regular expression, is the pattern to which the column expression is compared. A regular expression can include the pattern matching metacharacters described in Table 59 on page 153. The NOT keyword inverts the result of the comparison.	<pre>SELECT * FROM london.status WHERE Summary LIKE 'down';</pre> The result is all rows in which Summary contains the substring down.

The LIKE and NOT LIKE comparison operators allow special regular expression pattern-matching metacharacters in the string being compared to the column expression. Regular expressions are made up of normal characters and metacharacters. Normal characters include uppercase and lowercase letters and numbers. Regular expression pattern matching can be performed with either a single character or a pattern of one or more characters in parentheses, called a *character pattern*. Metacharacters have special meanings, described in Table 59.

Table 59: Pattern Matching Metacharacters (1 of 2)

Pattern Matching Metacharacter	Description	Example
*	Matches zero or more instances of the preceding character or character pattern.	The pattern <code>goo*</code> matches <code>my godness</code> , <code>my goodness</code> , and <code>my goodness</code> , but not <code>my gdness</code> .
+	Matches one or more instances of the preceding character or character pattern.	The pattern <code>goo+</code> matches <code>my goodness</code> and <code>my goodness</code> , but not <code>my godness</code> .
?	Matches zero or one instance of the preceding character or character pattern.	The pattern <code>goo?</code> matches <code>my godness</code> and <code>my goodness</code> , but not <code>my goodness</code> or <code>my gdness</code> .
\$	Matches the end of the string.	The pattern <code>end\$</code> matches <code>the end</code> , but not <code>the ending</code> .
^	Matches the beginning of the string.	The pattern <code>^severity</code> matches <code>severity level 5</code> , but not <code>The severity is 5</code> .
.	Matches any single character.	The pattern <code>b.at</code> matches <code>baat</code> , <code>bBat</code> , and <code>b4at</code> , but not <code>bat</code> or <code>bB4at</code> .
[abcd]	Matches any characters in the square brackets or in the range of characters separated by a hyphen (-), such as [0-9].	<code>^[A-Za-z]+\$</code> matches any string that contains only uppercase or lowercase letter characters.
[^abcd]	Matches any character except those in the square brackets or in the range of characters separated by a hyphen (-), such as [0-9].	<code>[^0-9]</code> matches any string that does not contain any numeric characters.
()	Indicates that the characters in parentheses should be treated as a character pattern.	<code>A(boo)+Z</code> matches <code>AbooZ</code> , <code>AboobooZ</code> , and <code>AboobooobooZ</code> , but not <code>AboZ</code> or <code>AboooZ</code> .
	Matches one of the characters or character patterns on either side of the vertical bar.	<code>A(B C)D</code> matches <code>ABD</code> and <code>ACD</code> , but not <code>AD</code> , <code>ABCD</code> , <code>ABBD</code> , or <code>ACCD</code> .

Table 59: Pattern Matching Metacharacters (2 of 2)

Pattern Matching Metacharacter	Description	Example
\	The backslash escape character indicates that the metacharacter following should be treated as a regular character. The metacharacters in this table require a backslash before them if they appear in a regular expression.	To match an opening square bracket, followed by any digits or spaces, followed by a closed bracket, use the regular expression <code>\[[0-9]*\]</code> .

List Comparison Operators

You can use list comparison operators to compare a value to a list of values. Conditions using list comparison operators use the comparison operators described in *Binary Comparison Operators* on page 151 with ANY, ALL, IN, or NOT IN operators.



Syntax

The syntax of a list comparison expression is either:

```
expression comparison_operator { ANY | ALL } ( expression, ... )
```

or

```
expression [ NOT ] IN ( expression, ... )
```

If you use the ANY keyword, the list comparison condition evaluates to TRUE if the comparison of the left hand expression to the right hand expressions returns TRUE for any of the values. If you use the ALL keyword, the list comparison condition evaluates to TRUE if the comparison of the left hand expression to the right hand expressions returns TRUE for all of the values.

An IN comparison returns the same results as the =ANY comparison. A NOT IN comparison returns the same results as the <>ANY comparison.



Example

In the query:

```
select * from mystatus where Severity - 1 IN (Old_Severity, 5)
```

The query returns the rows in which Severity - 1 is equal to the value of Old_Severity or the number 5.



Note: The ANY and ALL operators are not supported in subqueries.

Logical Operators

You can use logical operators on boolean values to form expressions that resolve to TRUE or FALSE. The ObjectServer supports the following operators:

- NOT
- AND
- OR
- XOR (exclusive or)

You can combine comparisons using logical operators.



Example

```
SELECT * from alerts.status where Node = 'node1' and Severity > 4 and
Summary like 'alert on .*'
```

The following truth tables show the results of logical operations on boolean values. In the sample truth tables, A and B represent any value or expression.

A NOT expression is TRUE only if its input is FALSE, as shown in Table 60.

Table 60: Truth Table for NOT Operator

A	NOT A
FALSE	TRUE
TRUE	FALSE



Example

The following query returns all rows in table t1 where the value for col1 is not equal to 0:

```
select * from t1 where NOT(col1 = 0);
```

An AND expression is true only if all of its inputs are TRUE, as shown in Table 61.

Table 61: Truth Table for AND Operator

A	B	A AND B
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

An OR expression is TRUE if any of its inputs are TRUE, as shown in Table 62.

Table 62: Truth Table for OR Operator

A	B	A OR B
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

An XOR expression is TRUE if either of its inputs, but not both, are TRUE, as shown in Table 63.

Table 63: Truth Table for XOR Operator

A	B	A XOR B
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

Operator Precedence

If an expression contains multiple operators, the ObjectServer uses operator precedence to determine the order in which to evaluate the expression. Operators are evaluated from those with the highest precedence to those with the lowest precedence. For example, the binary plus (+) operator has a lower precedence than the multiplication operator (*). In the expression

$3 + 5 * 2$, the result is 13 because 5 is multiplied by 2 before the result (10) is added to 3.

Use parentheses in an expression to change the order in which the items are evaluated. The contents of parentheses are always evaluated before anything outside of the parentheses. In the expression $(3 + 5) * 2$, the result is 16 because 3 is added to 5 before the result (8) is multiplied by 2.

If operators have equal precedence, they are evaluated in order from left to right. Table 64 shows the order of precedence of all ObjectServer operators.

Table 64: Operator Precedence

Highest Precedence
Unary + -
Math * /
Binary + -
Comparison operators (including list comparisons)
NOT
AND
XOR
OR
Lowest Precedence

Functions

A function processes a data item or items in an SQL command and returns a value. The syntax notation for a function is:

```
function(operand, ...)
```



Tip: The parentheses are optional if there are no operands to the function.

Table 65 describes the functions supported by the ObjectServer.

Table 65: ObjectServer Functions (1 of 4)

Function	Description	Example
<code>array_len(array)</code>	Returns the number of elements in an array. This function can only be used in procedures or triggers.	If the array <code>myarray</code> has ten elements, <code>array_len(myarray)</code> returns 10. For an example using this function in a procedure, see <i>FOR Loop</i> on page 196.
<code>ceil(real)</code>	Takes a real argument and returns the smallest integral value not less than the argument.	<code>ceil(2.01)</code> returns 3.000000
<code>dayasnum(time)</code>	Takes a time argument and extracts the day of the week as an integer. If no argument is specified, the argument is assumed to be the current time.	<code>select dayasnum>LastOccurrence) from mytab;</code> Sunday is 0, Monday is 1, and so on.
<code>dayname(time)</code>	Takes a time argument and returns the name of the day. If no argument is specified, the argument is assumed to be the current time.	<code>select dayname>LastOccurrence) from mytab;</code> The output is Monday, Tuesday, and so on.
<code>dayofmonth(time)</code>	Takes a time argument and extracts the day of the month as an integer. If no argument is specified, the argument is assumed to be the current time.	<code>select dayofmonth>LastOccurrence) from mytab;</code>
<code>dayofweek(time)</code>	Takes a time argument and extracts the day of the week as an integer. If no argument is specified, the argument is assumed to be the current time.	<code>select dayofweek>LastOccurrence) from mytab;</code> Unlike <code>dayasnum</code> , Sunday is 1, Monday is 2, and so on.
<code>getdate()</code>	Takes no arguments and returns the current date and time as a Coordinated Universal Time (UTC) value (the number of seconds since 1 January 1970). See Table 53 on page 136 for more information on the <code>time</code> data type.	To return all rows in the <code>alerts.status</code> table that are more than ten minutes old: <code>select Summary, Severity from alerts.status where LastOccurrence < getdate - 600;</code>
<code>getenv(string)</code>	Returns the value of the specified environment variable as a string.	<code>getenv('OMNIHOME')</code> returns a directory name, for example, <code>/opt/netcool/omnibus</code> .

Table 65: ObjectServer Functions (2 of 4)

Function	Description	Example
<code>get_prop_value(string)</code>	Returns the value of the specified ObjectServer property as a string.	<code>get_prop_value('Name')</code> returns the ObjectServer name, for example, NCOMS.
<code>hourofday(time)</code>	Takes a time argument and extracts the hour of the day as an integer. If no argument is specified, the argument is assumed to be the current time.	<pre>select hourofday(LastOccurrence) from mytab;</pre>
<code>is_env_set(string)</code>	Returns 1 if the specified environment variable is set; 0 otherwise.	When the OMNIHOME environment variable is set, <code>is_env_set('OMNIHOME')</code> returns 1.
<code>log_2(real)</code>	Takes a positive real argument and returns the logarithm to base 2.	<code>log_2(4.0)</code> returns 2.000000
<code>lower(string)</code>	Converts a character string argument into lowercase characters.	<code>lower('LIMA')</code> returns lima
<code>minuteofhour(time)</code>	Takes a time argument and extracts the minute of the hour as an integer. If no argument is specified, the argument is assumed to be the current time.	<pre>select minuteofhour(LastOccurrence) from mytab;</pre>
<code>mod(int1, int2)</code>	Returns the integer remainder of <i>int1</i> divided by <i>int2</i> .	<code>mod(12, 5)</code> returns 2
<code>monthasnum(time)</code>	Takes a time argument and extracts the month of the year as an integer. If no argument is specified, the argument is assumed to be the current time.	<pre>select monthasnum(LastOccurrence) from mytab;</pre> January is 0, February is 1, and so on.
<code>monthname(time)</code>	Takes a time argument and returns the name of the month. If no argument is specified, the argument is assumed to be the current time.	<pre>select monthname(LastOccurrence) from mytab;</pre> The output is January, February, and so on.
<code>monthofyear(time)</code>	Takes a time argument and extracts the month of the year as an integer. If no argument is specified, the argument is assumed to be the current time.	<pre>select monthofyear(LastOccurrence) from mytab;</pre> Unlike <code>monthasnum</code> , January is 1, February is 2, and so on.
<code>power(real1, real2)</code>	Takes two real arguments and returns <i>real1</i> raised to the power of <i>real2</i> .	<code>power(2.0, 3.0)</code> returns 8.000000

Table 65: ObjectServer Functions (3 of 4)

Function	Description	Example
<code>secondofminute (time)</code>	Takes a time argument and extracts the second of the minute as an integer. If no argument is specified, the argument is assumed to be the current time.	<pre>select secondofminute (LastOccurrence) from mytab;</pre>
<code>to_char (argument [, conversion_specification])</code>	Converts the argument to a string. The argument can be of any data type except a string. If the argument is a time type, you can specify a second argument consisting of a conversion specification to format the output. This format is determined by the POSIX <code>strptime</code> function. The default format is <code>%a %b %d %T %Y</code> .	<pre>to_char (73) returns 73 to_char (FirstOccurrence) returns a string such as Thu Dec 11 16:02:05 2003</pre>
<code>to_int (argument)</code>	Converts the argument to an integer. The argument can be of any data type except integer. If the argument is a string, but it does not contain numeric characters, the function returns 0.	<pre>to_int ('73') returns 73 to_int ('3F') returns 3 to_int ('UK') returns 0</pre>
<code>to_real (argument)</code>	Converts the argument to a 64 bit real number. The argument can be of any data type except real. If the argument is a string, but it does not contain numeric characters, the function returns 0.	<pre>to_real ('7.3') returns 7.300000 to_real ('3F') returns 3.000000 to_real ('UK') returns 0</pre>
<code>to_time (argument [, conversion_specification])</code> <code>to_date (argument [, conversion_specification])</code>	Converts the argument to a time type. The argument can be of any data type except a time type. If the argument is a string type, you can specify a second argument consisting of a conversion specification to format the output. This format is determined by the POSIX <code>strptime</code> function. The default format is <code>%a %b %d %T %Y</code> .	<pre>update mytab set my_utc_col = to_time ('Thu Dec 11 16:00:00 2003')</pre>

Table 65: ObjectServer Functions (4 of 4)

Function	Description	Example
<code>to_unsigned(argument)</code>	Converts the argument to a 64 bit unsigned integer. The argument can be of any data type except a 64 bit unsigned integer. If the argument is a string, but it does not contain numeric characters, the function returns 0.	<code>to_unsigned('73')</code> returns 73 <code>to_unsigned(73)</code> returns 73 <code>to_unsigned('UK')</code> returns 0
<code>upper(string)</code>	Converts a character string argument into uppercase characters.	<code>upper('Vancouver')</code> returns VANCOUVER
<code>year(time)</code>	Takes a time argument and extracts the year as an integer. If no argument is specified, the argument is assumed to be the current time.	<code>select year>LastOccurrence) from mytab;</code>



Tip: In previous versions of Netcool/OMNIBus, date functions were called date literals.

Expressions

An expression is a syntactic combination of values and operations combined to compute new values. Expressions can be simple or complex.

Simple Expressions

A simple expression is a single constant or variable value, column name, or variable reference. This can be any of the following:

- A quoted string (`'Node XB1'`)
- A number (`9`)
- A column name (`Severity`)
- An ObjectServer property (`ServerName`)
- An environment variable (`OMNIHOME`)
- A variable that holds a temporary value in a procedure or a trigger

Complex Expressions

A complex expression is created from simple expressions combined using operators (`Severity - 1`) and SQL functions (`get_prop_value (ServerName)`). You can combine simple or complex expressions with other simple or complex expressions to create increasingly complex expressions, such as `-(Severity + Tally)`.



Note: Complex expressions are subject to type constraints. For example, the expression `5 * 'Node XB1'` is not valid because you cannot multiply an integer and a string. For more information about data types, see Table 53 on page 136.

Conditions

A condition is a combination of expressions and operators that evaluate to TRUE or FALSE. You can use conditions to search, filter, and test rows in the following:

- Restriction filters
- The WHERE clause of the SELECT, UPDATE, and DELETE SQL commands
- The HAVING clause of the SELECT GROUP BY command
- The WHEN clause in triggers
- The IF THEN ELSE, CASE WHEN, and FOR EACH ROW statements in procedures and triggers

Conditions can contain comparison operators (`Severity < 3`), logical operators (`NOT (Is_Enabled)`), and list comparison operators (`Severity IN ANY (0, 5)`). Operators and regular expressions are described in *Operators* on page 150. Expressions are described in *Expressions* on page 161.

The following are valid conditions:

```
TRUE | FALSE
( condition )
NOT condition
condition AND condition
condition OR condition
expression operator expression
expression operator ANY ( expression, ... )
expression operator ALL ( expression, ... )
expression [ NOT ] IN ( subquery )
expression operator ANY ( subquery )
expression [ NOT ] IN ( expression, ... )
expression [ NOT ] LIKE regexp_pattern
expression [ NOT ] LIKE ANY ( regexp_pattern, ... )
expression [ NOT ] LIKE ALL ( regexp_pattern, ... )
```




Note: The ANY and ALL operators are not supported in subqueries.

You can combine conditions into increasingly complex conditions.



Example

```
(Severity > 4) AND (Node = 'node%')
```

The following example shows the use of a condition in a subquery:

```
select * from alerts.status where Serial in (select Serial from alerts.journal);
```

3.7 Data Manipulation Language SQL Commands

Data manipulation language commands enable you to query and modify data using ObjectServer SQL. ObjectServer SQL provides the following commands to enable you to manipulate data in existing tables, views, and files.

Table 66: ObjectServer Objects and Associated DML Commands

ObjectServer Object	Allowed DML Commands	Described on page...
TABLE	SELECT INSERT UPDATE DELETE DESCRIBE SVC	page 167 page 164 page 165 page 166 page 172 page 172
VIEW	SELECT UPDATE DELETE DESCRIBE SVC	page 167 page 165 page 166 page 172 page 172
FILE	WRITE INTO	page 171



Tip: Restriction filters are automatically applied in SELECT, INSERT, UPDATE, and DELETE commands.

Adding Table Data: The INSERT Command

Use the INSERT command to insert a new row of data into an existing table.



Syntax

```
INSERT INTO [database_name.] table_name
[ (column_name, ...) ] VALUES (expression, ...);
```

You must specify a value for every primary key column in the table.

If you are inserting values for every column in the row, specify the `VALUES` keyword followed by a comma-separated list of column values in parentheses. Enter the values in sequential column order.

If you are *not* inserting values for every column in the row, specify a comma-separated list of columns being inserted in parentheses, followed by the `VALUES` keyword, followed by a comma-separated list of column values in parentheses. Enter the values in the same sequence as the specified columns. All other columns are populated with default values.



Tip: You cannot assign values to system-maintained columns such as `Serial`.



Example

To insert an alert into the `mydb.mystatus` table specifying the values in the indicated columns, enter:

```
insert into mydb.mystatus (Identifier, Severity, LastOccurrence)
values ('MasterMachineStats15', 5, getdate);
```

Modifying Table Data: The UPDATE Command

Use the `UPDATE` command to update one or more columns in an existing row of data in a table or view.



Syntax

```
UPDATE [database_name.]object_name
[ VIA (value_of_primary_key_column,...) ]
SET column_name = expression,...
[ WHERE condition ];
```

You cannot update system-maintained columns such as `Serial`, or columns where the `NOMODIFY` property, described in Table 54 on page 137, is set to `TRUE`.

If you are updating a single row and you know the value of the primary key for the row you wish to update, specify the value using the optional `VIA` clause to improve the performance of the update command. If there is more than one primary key column, the values must be specified in order, separated by commas and in parentheses. Columns that have string values must be enclosed in quotes.

If you include a `WHERE` clause, only rows meeting the criteria specified in the *condition* are updated. Conditions are described in *Conditions* on page 162. If no condition is specified in the `WHERE` clause, all rows are updated.



Examples

To set the `Severity` to 0 for rows of the `alerts.status` table where the `Node` is equal to `Fred`, enter:

```
update alerts.status set Severity = 0 where Node = 'Fred';
```

You can use column values in calculations. In the following example, `Severity` is set to 0 when an alert has been acknowledged:

```
update status set Severity=(1-Acknowledged)*Severity;
```

To search for rows where the `Severity` is equal to 1 and the `Node` is equal to `Fred`, and then set the `Severity` to 0 and change the `Summary` field to the string `Discarded`, enter:

```
update alerts.status set Severity = 0, Summary = 'Discarded'
where Severity = 1 and Node = 'Fred';
```

Removing Table Data: The DELETE Command

Use the `DELETE` command to delete one or more rows of data from an existing table or view.



Syntax

```
DELETE FROM [database_name.]object_name
[ VIA ('value_of_primary_key_column',...) ]
[ WHERE condition ];
```

If you are deleting a single row and you know the value of the primary key for the row you wish to delete, specify the value using the optional `VIA` clause to improve the performance of the delete command. If there is more than one primary key column, the values must be specified in order, separated by commas and in parentheses. If the column has a string value, it must be enclosed in quotes.

If you include a `WHERE` clause, only rows meeting the criteria specified in the *condition* are deleted. Conditions are described in *Conditions* on page 162. If no `WHERE` clause is specified, all rows are deleted.



Example

To remove all the rows of the `alerts.status` table where the value of the `Node` field is equal to `Fred`, enter:

```
delete from alerts.status where Node = 'Fred' ;
```

Viewing Table Data: The SELECT Command

Use the `SELECT` command to retrieve one or more rows, or partial rows, of data from an existing table or view, and to perform grouping functions on the data.

Basic (Scalar) Select

The scalar `SELECT` command retrieves columns and rows from a table based on specified criteria.



Syntax

```
SELECT [ TOP num_rows ] { * | scalar_column_expr [ AS alias_name ],... }
FROM [ database_name. ]object_name
[ WHERE condition ]
[ ORDER BY column_name_or_alias [ ASC | DESC ] ,... ];
```

Use the optional `TOP` keyword to display only the first *num_rows* number of rows of the query results that match the selection criteria. If you include the `TOP` keyword, you should also include an `ORDER BY` clause to order the selected rows.

Use an `*` to retrieve all non-hidden columns in the table. Otherwise, specify a comma-separated list of columns to retrieve. You can also create *virtual columns* using:

- Simple expressions (`Severity`)
- Complex expressions that contain math or string operators (`Severity + Tally`)
- Functions (`getdate - 60`)

Expressions are described in *Expressions* on page 161. Math and string operators are listed in *Math and String Operators* on page 151. Functions are listed in Table 65 on page 158.

Following a column or virtual column, you can include the `AS` keyword followed by an alias. This alias is a replacement heading for the column or virtual column name, and is displayed in the query results. If you specify a column alias, use that alias in any references in the `ORDER BY` clause. The maximum length of a column name or alias is 40 characters.

If you include a `WHERE` clause, only rows meeting the criteria specified in the *condition* are returned. Conditions are described in *Conditions* on page 162.

Use the optional `ORDER BY` clause to display the results in sequential order depending on the values of one or more column names, in either descending (`DESC`) or ascending (`ASC`), order. If the `ORDER BY` clause is not specified, no ordering is used. If you have specified a column alias using the `AS` keyword, use that alias in any references in the `ORDER BY` column list rather than the corresponding column name.



Examples

To select all rows of the `alerts.status` table where the `Severity` is equal to 4, enter:

```
select * from alerts.status where Severity = 4;
```

To select all rows of the `alerts.status` table where the `Node` contains the string `terminal` followed by any other characters, enter:

```
select * from alerts.status where Node like 'terminal.*';
```

For information on pattern-matching metacharacters used in the `LIKE` comparison, see Table 59 on page 153.

In the following example, the virtual column `Severity + Tally` is populated by adding the values of the two columns together:

```
select Severity, Severity + Tally from alerts.status;
```

The following example is the same as the previous example, except the virtual column `Severity + Tally` is renamed `Real_Severity`.

```
select Severity, Severity + Tally as Real_Severity from alerts.status;
```

Aggregate Select

An aggregate `SELECT` command differs from a scalar `SELECT` command because it performs a calculation on a number of rows and returns a single value.



Syntax

```
SELECT aggr_expression [ AS alias_name ], ...
FROM [ database_name ] table_name
[ WHERE condition ];
```

The aggregate functions described in Table 67 are supported.

Table 67: Aggregate Functions (1 of 2)

Function	Result Returned
<code>max(<i>scalar_column_expr</i>)</code>	Returns the maximum numeric value for the column expression from the rows that meet the <code>SELECT</code> condition.
<code>min(<i>scalar_column_expr</i>)</code>	Returns the minimum numeric value for the column expression from the rows that meet the <code>SELECT</code> condition.

Table 67: Aggregate Functions (2 of 2)

Function	Result Returned
<code>avg (scalar_column_expr)</code>	Returns the average numeric value for the column expression from the rows that meet the <code>SELECT</code> condition.
<code>sum (scalar_column_expr)</code>	Returns the sum (total) of the numeric values for the column expression from the rows that meet the <code>SELECT</code> condition.
<code>count (scalar_column_expr)</code> <code>count (*)</code>	Returns the total number of rows that meet the <code>SELECT</code> condition.
<code>dist (scalar_column_expr, value)</code>	Returns the total number of rows for which the column equals the specified value. The result of <code>dist (scalar_column_expr, value)</code> is equivalent to <code>SELECT count (scalar_column_expr) FROM table_name</code> <code>WHERE scalar_column_expr = value;</code>

Following an aggregate expression, you can include the `AS` keyword followed by an alias. This alias is a replacement heading for the aggregate expression, and is displayed in the query results.

The maximum length of a column name or alias is 40 characters.

If you include a `WHERE` clause, only rows meeting the criteria specified in the *condition* are returned. Conditions are described in *Conditions* on page 162.



Examples

The following example returns the highest `Severity` value, the average `Severity` value, and the number of rows for which the `Severity` is equal to 4:

```
select MAX(Severity), AVG(Severity), DIST(Severity, 4) from alerts.status;
```

The following example returns the number of rows for which the value of `Node` is `myhost`:

```
select DIST(Node, 'myhost') from alerts.status;
```

The following examples do comparisons using the `getdate` function, which returns the current time:

```
select MAX(getdate-LastOccurrence) from alerts.status;
```

```
select AVG((getdate-LastOccurrence)/60) as ResponseTime from alerts.status where OwnerUID=34;
```

Functions are described in Table 65 on page 158.

Group By Select

A `SELECT` command containing a `GROUP BY` clause enables you to group into a single row all rows that have identical values in a specified column or combination of columns. Used with aggregate functions, described in the preceding section, you can find the aggregate value for each group of column values.



Syntax

```
SELECT [ TOP num_rows ] scalar_column_expr_and_aggr_column_expr [ AS alias_name ] ,...
FROM [database_name.]table_name
[ WHERE condition ]
GROUP BY scalar_column_expr_or_alias ,... [ HAVING condition ]
[ ORDER BY aggr_expr_or_alias [ { ASC | DESC } ] ,... ];
```

The `GROUP BY` syntax combines scalar column expressions and aggregate expressions, described in *Basic (Scalar) Select* on page 167 and *Aggregate Select* on page 168, respectively. An `*` is allowed only in the `COUNT (*)` aggregate function.

Following a scalar or aggregate expression, you can include the `AS` keyword followed by an alias. This alias is a replacement heading for the scalar column or aggregate expression, and is displayed in the query results. You must specify an alias for every virtual column. This enables you to reference it in the `GROUP BY` clause. If you do not specify an alias for an aggregate expression, you cannot reference it in the aggregate expression in the `ORDER BY` clause.

The maximum length of a column name or alias is 40 characters.

The `GROUP BY` clause gathers all of the rows together that contain data in the specified columns and allows aggregate functions to be performed on these columns based on column values. If you have specified a column alias using the `AS` keyword, use that alias in the `GROUP BY` column list rather than the corresponding column name or expression.



Note: The column list in the `GROUP BY` clause must match the column list being selected, and must not contain any of the aggregate expressions.

The *condition* following the optional `HAVING` keyword is an expression or expressions that returns a subset of rows of the table. Unlike other conditions in ObjectServer SQL, those in the `HAVING` clause can include aggregate functions. Conditions are described in *Conditions* on page 162.

Use the optional `ORDER BY` clause to display the results in sequential order depending on the values of one or more aggregate expressions, in either descending (`DESC`) or ascending (`ASC`), order. If the `ORDER BY` clause is not specified, no ordering is used. You must use the alias for the aggregate expression in the `ORDER BY` clause rather than the corresponding aggregate expression.



Examples

The following example returns the highest `Severity` value found for each node:

```
select Node, max(Severity) from alerts.status group by Node;
```

The following example returns the highest severity value found for each node except the node named `Sun1`, ordered from lowest to highest maximum severity:

```
select Node, max(Severity) as MAX_Sev from alerts.status where Node <> 'Sun1' group by Node order by MAX_Sev;
```

The column alias for `max(Severity)`, which is `MAX_Sev`, is displayed as the heading in the query results.

Logging to Files: The WRITE INTO Command

Use the `WRITE INTO` command to write logging information to ObjectServer files.



Syntax

```
WRITE INTO file_name [ VALUES ] (expression, ...);
```

An ObjectServer file is a logical file, which has a corresponding file or set of files on the physical file system. For more information on ObjectServer files, see *Files* on page 143.

A carriage return follows each message.



Example

```
WRITE INTO file1 VALUES
('User', %user.user_name, 'connected to the system at', getdate );
```

This command adds a message to the physical file associated with the ObjectServer file `file1` each time a user connects to a database.

The `%user.user_name` user variable used in this example is only available in procedures and triggers. For more information, see *Using Implicit User Variables in Procedures and Triggers* on page 197.

Viewing Table Column Information: The DESCRIBE Command

Use the DESCRIBE command to display information about the columns of the specified table or view.



Syntax

```
DESCRIBE [database_name.] object_name;
```

The output includes the column name, the data type (returned as the ObjectServer ID as listed in Table 53 on page 136), the length of the column, and whether the column is part of a primary key (1 if it is, 0 if it is not).

Hidden columns are not displayed because they are maintained by the system, and a typical user does not need to view or update them.



Example

```
describe catalog.tables;
```

Sample output for the preceding command is:

ColumnName	Type	Size	Key	
TableName		2	40	1
DatabaseName		2	40	1
Status		0	4	0
NumDependents		12	4	0
TableID		0	4	0
TableKind		0	4	0
StorageKind		0	4	0
ServerID		0	4	0

Adding and Modifying Service Status Data: The SVC Command

Use the SVC command to add or update the state of a service status alert in the `service.status` table for Netcool/Internet Service Monitors.



Syntax

```
SVC UPDATE 'name' integer;
```

In this command, *name* is the name of the profile element generating the alert and *integer* is its current status. Valid values for the service status are shown in Table 68.

If any other value is entered, the service level is set to 3 (unknown).

Table 68: Service Status Levels

Integer	Service Status Level
0	Good.
1	Marginal.
2	Bad.
3	Service level is unknown.



Example

```
svc update 'newservice' 2;
```

3.8 System and Session SQL Commands

The ObjectServer includes a system command that enables you to change the default and current settings of the ObjectServer. The `ALTER SYSTEM` command enables you to:

- Shut down the ObjectServer
- Set one or more ObjectServer properties
- Drop one or more user connections
- Back up the ObjectServer

The ObjectServer also has session control commands that you can use in an `nco_sql` session:

- The `SET DATABASE` (also `USE DATABASE`) command enables you to set the default database in an `nco_sql` session.
- The `CHECK SYNTAX` command enables you to check the syntax of the specified SQL.

Managing the ObjectServer: The ALTER SYSTEM Command

Use the `ALTER SYSTEM` command to shut down or back up the ObjectServer, change the settings for ObjectServer properties, or drop user connections.



Syntax

```
ALTER SYSTEM
{
  SHUTDOWN |
  SET 'property_name' = value [ ... ] |
  DROP CONNECTION connection_id [, ... ] |
  BACKUP 'directory_name'
}
;
```

You can stop the ObjectServer with the `ALTER SYSTEM SHUTDOWN` command.

You can set ObjectServer properties with the `SET` keyword, followed by the property name enclosed in quotes and a value for the property. You can change more than one property in a single command. In addition to updating the `catalog.properties` table, the changed properties are written to the properties file. ObjectServer properties are described in *ObjectServer Properties and Command Line Options* on page 19.

You can drop user connections with the `ALTER SYSTEM DROP CONNECTION` command. Specify one or more connection identifiers in a comma-separated list. You can find the identifiers for all current connections by querying the `catalog.connections` system table. The `ConnectionID` column contains the connection identifier.

You can back up the ObjectServer with the `ALTER SYSTEM BACKUP` command. Specify an existing directory name in quotes.



Note: The directory cannot be the one in which ObjectServer data files are stored, which is `$OMNIHOME/db/server_name` by default.

The backup generates copies of the ObjectServer `.tab` files in the specified directory.



Tip: The triggers in the `automatic_backup_system` trigger group, defined in the `automation.sql` file, use the `ALTER SYSTEM BACKUP` command to provide an automatic backup facility. The `automatic_backup` trigger is disabled by default; you must enable it to create backups automatically. You can also customize this trigger to suit your environment. For example, you can change the number of backups saved.

To recover the ObjectServer to the point in time at which the `BACKUP` command was issued, copy the ObjectServer `.tab` files into the ObjectServer data file directory. The backup files can only be used on a machine with the same architecture as the machine on which they were created. The `.tab` files are described in *Checkpoint File Storage* on page 39.



Examples

```
alter system set 'Auto.StatsInterval' = 15 set 'AlertSecurityModel' = 1;

alter system shutdown;
```

Changing the Default Database: The SET and USE DATABASE Commands

The `SET DATABASE` and `USE DATABASE` commands perform the same function. After you set the default database with the `SET DATABASE` or `USE DATABASE` command in an `nco_sql` session, you can specify an object name without preceding it with the database name. The default database setting lasts for the length of the session in which it is set.



Syntax

```
{ SET | USE } DATABASE database_name;
```

You cannot use this command in triggers or procedures.



Note: The default database is not applied in the `CREATE VIEW` and `DROP VIEW` commands. If no database name is specified in these commands, the view is always created or dropped in the `alerts` database.



Examples

```
use database newthings;  
  
set database mydb;
```

Verifying SQL Syntax: The CHECK STATEMENT Command

The `CHECK STATEMENT` command parses and checks the syntax of the SQL commands enclosed in quotes and returns either a success message or a description of any errors.



Syntax

```
CHECK STATEMENT 'command; command; ...';
```

Because `CHECK STATEMENT` does not execute the SQL commands, runtime errors are not detected. Additionally, some spurious errors may be displayed if there is a series of commands that relies on the preceding commands being executed.

3.9 Security SQL Commands

The SQL commands listed in this section control access to ObjectServer objects through *groups* (collections of users), and *roles* (collections of system and object permissions) granted to groups.

Permissions control access to objects and data in the ObjectServer. By combining one or more permissions into roles, you can manage access quickly and efficiently.

Each user is assigned to one or more groups. You can then assign groups permission to perform actions on database objects by granting one or more roles to the group. You can create logical groupings such as super users or system administrators, physical groupings such as London or New York NOCs, or any other groupings to simplify your security setup.

For example, creating automations requires knowledge of Netcool/OMNIBus operations and the way a particular ObjectServer is configured. You do not typically want all of your users to be allowed to create or modify automations. One solution is to create a role named `AutoAdmin`, with permissions to create and alter trigger groups, files, procedures, external procedures, and signals. You can then grant that role to a group of administrators who will be creating and updating triggers.

The `security.sql` SQL script, described in the Netcool/OMNIBus Installation and Deployment Guide, contains default groups and roles for Netcool operators and administrators. You can also use this as a template to create your own groups and roles.

Creating, Modifying, and Deleting Users and Groups

This section describes how to create, modify, and drop users and groups. Once you have created groups of users, you can assign roles to them to allow them access to ObjectServer objects.

The CREATE USER Command

Use the `CREATE USER` command to add a user to the ObjectServer.



Syntax

```
CREATE USER 'user_name'
  [ ID identifier ]
  FULL NAME 'full_user_name'
  [ PASSWORD 'password' [ ENCRYPTED ] ]
  [ PAM { TRUE | FALSE } ];
```

The user name is a text string containing the name of the user being added. Ownership of each alert in the `alerts.status` table is assigned to a user when the row is inserted. By default, probes assign rows to the `nobody` user.



Note: User names are case-sensitive, and must be enclosed in quotes. Any leading or trailing whitespace is discarded.

The *identifier* is an integer value that uniquely identifies the user. If you do not specify an identifier, one is automatically assigned. The identifier for the `root` user is 0. The identifier for the `nobody` user is 65534. Identifiers for other users can be set to any value between 1 and 65533.

The *full_user_name* is a text string containing a descriptive name for the user.

You can specify the user password using the `PASSWORD` keyword. The default is an empty string. If you add the keyword `ENCRYPTED`, the password is assumed to be encrypted.

If you are using Pluggable Authentication Modules (PAM) to authenticate the user, set `PAM` to `TRUE`. The `Sec . UsePam` ObjectServer property must also be set to `TRUE`. If disabled for the ObjectServer or for the individual user, PAM is not used to authenticate the user. For more information on PAM, see the Netcool/OMNibus Installation and Deployment Guide.



Example

```
create user 'joe' id 1 full name 'Joseph R. User';
```

The ALTER USER Command

Use the `ALTER USER` command to change the settings, such as the password, for the specified user.



Syntax

```
ALTER USER 'user_name'
  SET PASSWORD 'password' [ AUTHORIZE PASSWORD 'old_password' ] [ ENCRYPTED ]
  SET FULL NAME 'full_user_name'
  SET ENABLED { TRUE | FALSE }
  SET PAM { TRUE | FALSE }
  ASSIGN [ RESTRICTION ] FILTER restriction_filter_name
  REMOVE [ RESTRICTION ] FILTER restriction_filter_name ;
```

The user name is a text string containing the name of the user being modified.

Use the `PASSWORD` setting to change the password for the specified user. When changing a password stored on an external system and accessed using PAM, you must also specify the old password following the `AUTHORIZE PASSWORD` keywords.

Use the `ENABLED` setting to activate or deactivate the specified user.

Use the PAM setting to enable or disable the use of Pluggable Authentication Modules (PAM) to authenticate the user. The `Sec.UsePam.ObjectServer` property must also be set to `TRUE`. If disabled for the ObjectServer or for the individual user, PAM is not used to authenticate the user. For more information on PAM, see the Netcool/OMNIBus Installation and Deployment Guide.

Use the `ASSIGN` or `REMOVE RESTRICTION FILTER` settings to assign or remove the restriction filters that apply to the user. Only one restriction filter per table can be assigned to a user. Restriction filters are described in *Restriction Filters* on page 142.

You can change more than one setting in a single `ALTER USER` command.



Example

```
alter user 'joe' set password 'topsecret';
```

The DROP USER Command

Use the `DROP USER` command to delete the specified user.



Syntax

```
DROP USER 'user_name';
```

The user name is a text string containing the name of the user being dropped.



Example

```
drop user 'joe';
```

The CREATE GROUP Command

Use the `CREATE GROUP` command to create a group of one or more users.



Syntax

```
CREATE GROUP 'group_name'
  [ ID identifier ]
  [ COMMENT 'comment_string' ]
  [ MEMBERS 'user_name', ... ] ;
```

The group name is a text string containing the name of the group being created.



Note: Group names are case-sensitive, and must be enclosed in quotes. Any leading or trailing whitespace is discarded.

The *identifier* is an integer value that uniquely identifies the group. If you do not specify an identifier, one is automatically assigned. Identifiers 0 through 9 are reserved for system groups. Identifiers for other groups can be set to any value between 10 and 65534.

Use the optional COMMENT setting to add a description of the group you are creating.

When you create a group, you can specify one or more users as group members following the MEMBERS keyword.



Example

```
create group 'AutoAdmin' id 3 COMMENT 'Group to manage Automations'
  members 'joe', 'bob';
```

The ALTER GROUP Command

Use the ALTER GROUP command to change user settings, such as the included users, for the specified group.



Syntax

```
ALTER GROUP 'group_name'
  SET COMMENT 'comment_string'
  ASSIGN [ RESTRICTION ] FILTER restriction_filter_name
  REMOVE [ RESTRICTION ] FILTER restriction_filter_name
  ASSIGN MEMBERS 'user_name', ...
  REMOVE MEMBERS 'user_name', ... ;
```

The group name is a text string containing the name of the group being altered.

Use the ASSIGN or REMOVE RESTRICTION FILTER setting to assign or remove restriction filters which apply to the group. Only one restriction filter per table can be assigned to a group. Restriction filters are described in *Restriction Filters* on page 142.

Use the SET COMMENT setting to modify the description of the group.

Use the ASSIGN or REMOVE MEMBERS setting to assign or remove members of the group.

You can change more than one setting in a single ALTER GROUP command.



Example

```
alter group 'AutoAdmin' assign members 'sue';
```

The DROP GROUP Command

Use the `DROP GROUP` command to delete the specified group.



Syntax

```
DROP GROUP 'group_name';
```

The group name is a text string containing the name of the user being dropped.



Note: The default groups `Normal`, `Administrator`, and `Super User` provide group row level security in the event list. These groups cannot be deleted or renamed. These and other default groups are created by the `security.sql` script, which is described in the *Netcool/OMNIbus Installation and Deployment Guide*.



Example

```
drop group 'LondonAdmin';
```

Creating, Modifying, and Deleting Roles

This section describes how to create, modify, and delete roles. Once you have created roles, you can assign permissions to them, as described in *Assigning System and Object Permissions to Roles* on page 183.

The role can then be granted to one or more groups using the `GRANT ROLE` command, described in *The GRANT ROLE Command* on page 187.

The CREATE ROLE Command

Use the `CREATE ROLE` command to create a role, which is a collection of permissions.



Syntax

```
CREATE ROLE 'role_name'  
  [ ID identifier ]  
  [ COMMENT 'comment_string' ];
```

The role name is a text string containing the name of the role being added.



Note: Role names are case-sensitive, and must be enclosed in quotes. Any leading or trailing whitespace is discarded.

The *identifier* is an integer value that uniquely identifies the role. If you do not specify an identifier, one is automatically assigned. The `Normal` role has the identifier 3. The `Administrator` role has the identifier 2. The `SuperUser` role has the identifier 1, and is granted all permissions on all objects. Identifiers for other roles can be set to any value between 4 and 65534.

Use the optional `COMMENT` setting to add a description of the role you are creating.



Tip: Default roles are created by the `security.sql` script, which is described in the Netcool/OMNIBus Installation and Deployment Guide.



Example

```
create role 'SuperAdmin' id 500
  comment 'only users with root access should be granted this role';
```

The ALTER ROLE Command

Use the `ALTER ROLE` command to change the comment for an existing role.



Syntax

```
ALTER ROLE 'role_name' SET COMMENT 'comment_string';
```

The role name is a text string containing the name of the role being altered.

Use the `COMMENT` setting to modify the description of the role.



Tip: Default roles are created by the `security.sql` script, which is described in the Netcool/OMNIBus Installation and Deployment Guide.



Example

```
alter role 'SuperAdmin' set comment 'enhanced description of role';
```

The DROP ROLE Command

Use the DROP ROLE command to drop an existing role.



Syntax

```
DROP ROLE 'role_name';
```

The role name is a text string containing the name of the role being dropped. When a role is dropped, all the related role grants are dropped.



Example

```
drop role 'AutoAdmin';
```

Assigning System and Object Permissions to Roles

This section describes how to assign permissions to roles. There are two types of permissions:

- *System permissions*, which control the commands that can be executed in the ObjectServer.
- *Object permissions*, which control access to individual objects, such as tables.

The GRANT Command

Use the GRANT command to assign system and object permissions to roles.



Syntax for Granting System Permissions

```
GRANT system_permission, ...
  TO ROLE 'role_name', ...
  [ WITH GRANT OPTION ];
```

A *system_permission* is any of the following:

```
ISQL
ISQL WRITE
ALTER SYSTEM DROP CONNECTION
ALTER SYSTEM SHUTDOWN
ALTER SYSTEM BACKUP
ALTER SYSTEM SET PROPERTY
CREATE DATABASE
CREATE FILE
CREATE RESTRICTION FILTER
CREATE SQL PROCEDURE
```

```

CREATE EXTERNAL PROCEDURE
CREATE SIGNAL
CREATE TRIGGER GROUP
CREATE USER
CREATE GROUP
CREATE ROLE
ALTER USER
ALTER GROUP
ALTER ROLE
DROP USER
DROP GROUP
DROP ROLE
GRANT ROLE
REVOKE ROLE

```



Tip: You can query the `catalog.security_permissions` table to view information about permissions. For example, to view each system permission, use the following SQL command:

```

SELECT * FROM catalog.security_permissions WHERE Object = 'SYSTEM'
ORDER BY Permission;

```



Example

```

grant create database to role 'DDL_Admin';

```



Syntax for Granting Object Permissions

```

GRANT object_permission,... ON permission_object object_name
TO ROLE 'role_name',...
[ WITH GRANT OPTION ];

```

Each *permission_object* has permissions associated with it. The owner of the object (its creator) automatically has the grant and revoke permissions associated with that object, and can grant and revoke those permissions to other roles. Table 69 lists objects and the permissions the owner has and is able to grant to others.

Table 69: Objects and Associated Permissions (1 of 2)

Objects	Permissions
DATABASE	DROP CREATE TABLE CREATE VIEW

Table 69: Objects and Associated Permissions (2 of 2)

Objects	Permissions
TABLE	DROP ALTER SELECT INSERT UPDATE DELETE
VIEW	DROP ALTER SELECT UPDATE DELETE
TRIGGER GROUP	DROP ALTER CREATE TRIGGER
TRIGGER	DROP ALTER
FILE	DROP ALTER WRITE
SQL PROCEDURE EXTERNAL PROCEDURE	DROP ALTER EXECUTE
SIGNAL	DROP ALTER RAISE
RESTRICTION FILTER	DROP ALTER

The `WITH GRANT OPTION` option enables the roles to whom the permission is granted to grant the permission to other roles.



Tip: In commands where you can replace an existing object using the `CREATE OR REPLACE` syntax, you need `ALTER` permission to replace an existing object. Some objects can only be altered using the `CREATE OR REPLACE` syntax; for example, there is no `ALTER VIEW` command, but you can replace an existing view if you have `ALTER` permission on the view.

Inheritance of Object Permissions

When a new object is created, permissions are automatically granted on the new object based on the permissions currently granted on its parent. Table 70 lists the parent of each ObjectServer object.

Table 70: Inheritance of Object Permissions

Parent Object	Child Objects
System	DATABASE TRIGGER GROUP FILE SQL PROCEDURE EXTERNAL PROCEDURE SIGNAL RESTRICTION FILTER
DATABASE	TABLE VIEW
TRIGGER GROUP	TRIGGER

For example, if SuperAdmin has CREATE_DATABASE permission, and LondonAdmin creates a database, by default SuperAdmin has all object permissions on the database LondonAdmin created.

If the permissions on the parent are changed after the child object is created, this has no effect on the permissions on the child.



Example

```
grant drop on database testdb to role 'DDL_Admin';
```

The REVOKE Command

Use the REVOKE command to revoke system and object permissions from roles.



Syntax for Revoking System Permissions

```
REVOKE system_permission, ...  
FROM ROLE 'role_name', ... ;
```

System permissions are listed in *The GRANT Command* on page 183.

Each role name is a text string containing the name of a role from which the permission is being revoked.



Example

```
revoke create table from role 'DDL_Admin';
```



Syntax for Revoking Object Permissions

```
REVOKE object_permission,...
ON permission_object object_name
FROM ROLE 'role_name',... ;
```

Objects and their associated permissions are listed in Table 69 on page 184.

Each role name is a text string containing the name of a role from which the permission is being revoked.



Example

```
revoke drop on database testdb from role 'DDL_Admin';
```



Note: The REVOKE does not cascade within the permission hierarchy. For example, if you revoke the CREATE TABLE permission from the SuperAdmin role after SuperAdmin has granted this permission to the LondonAdministrators role, then LondonAdministrators retains CREATE TABLE permission after it is revoked from SuperAdmin.

Granting Roles to Groups

Once you have created roles as collections of permissions, you can grant them to groups and revoke them from groups.

The GRANT ROLE Command

Use the GRANT ROLE command to grant one or more roles to one or more groups. Role grants take effect in the next client session.



Syntax

```
GRANT ROLE 'role_name',...
TO GROUP 'group_name',... ;
```

Each role name is a text string containing the name of a role being granted.

Each group name is the group that is being granted the role or roles.



Example

```
grant role 'AutoAdmin' to group 'LondonAdministrators';
```

The REVOKE ROLE Command

Use the REVOKE ROLE command to revoke one or more roles from one or more groups.



Syntax

```
REVOKE ROLE 'role_name', ...  
FROM GROUP 'group_name', ... ;
```

Each role name is a text string containing the name of a role being revoked.

Each group name is the group that will no longer be granted the role or roles.



Note: The REVOKE ROLE command does not cascade within the role hierarchy. For example, if you revoke the AutoAdmin role from the group SuperAdmin after SuperAdmin has granted the AutoAdmin role to the group LondonAdministrators, then LondonAdministrators still has the AutoAdmin role.



Example

```
revoke role 'AutoAdmin' from group 'LondonAdministrators';
```

3.10 Procedures

A procedure is an executable SQL object that can be called to perform common operations. The types of procedures are:

- SQL procedures, which manipulate data in an ObjectServer database
- External procedures, which run an executable on a remote system

Once you create a procedure in the ObjectServer, you can execute it from `ncso_sql` or in a trigger using the `EXECUTE PROCEDURE` command described on page 199.

Creating SQL Procedures

An SQL procedure is a set of parameterized SQL commands, or code fragments, with programming language constructs that enable you to perform complex tasks on database objects. You can create a procedure containing a logical set of commands, such as a set of queries, updates, or inserts, that make up a task.

Procedures expand SQL syntax so you can:

- Pass parameters into and out of a procedure
- Create local variables and assign values to them
- Perform condition testing
- Perform scanning operations over tables and views

Use the `CREATE PROCEDURE` command to create SQL procedures. This command defines the structure and operation of the procedure, including the types of parameter passed into and out of the procedure, and the local variables, condition testing, row operations, and assignments performed in the procedure.



Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name ([ procedure_parameter, ... ])
[ DECLARE variable_declaration;...[;] ]
BEGIN
    procedure_body_statement;...[;]
END
```

If a procedure may already exist with the same name as the one you want to create, use the optional `OR REPLACE` keywords. If the procedure exists, it is replaced by the one you are creating. If the procedure does not exist, a new one is created.

The procedure name must be unique within the ObjectServer and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

Components of an SQL Procedure

This section provides an overview of the structure of an SQL procedure. The detailed syntax of each component is described in the following sections.

SQL procedures have the following major components:

- Parameters
- Local variable declarations
- Procedure body

Parameters are values passed into or out of a procedure. You declare the parameters of the procedure when you create the procedure and specify what values are passed as parameters when you execute the procedure. The name of the variable that contains a parameter is called a *formal parameter*, while the value of the parameter when the procedure is executed is called an *actual parameter*.

The values you pass to the procedure must be of the same data type as in the parameter declaration.



Example

In the procedure declaration:

```
CREATE PROCEDURE calculate_average_severity
  ( IN current_severity INTEGER )
```

The formal parameter is the variable `current_severity`. When you execute the procedure, you pass an actual parameter. For example, in the procedure call:

```
EXECUTE PROCEDURE calculate_average_severity(5);
```

The actual parameter is the value 5, which is assigned to the formal parameter `current_severity`.

You can also create *local variables* for use within the procedure to hold and change temporary values in the body of the procedure. Local variables and values are always discarded when the procedure exits. For example, you can create an integer counter as a local variable.



Note: Because both parameters and local variables contain data that can change, both parameters and local variables are referred to as variables within procedures.

The *body* of a procedure contains a set of statements that test conditions and manipulate data in the database. The body of a procedure is enclosed within the keywords `BEGIN` and `END`.

SQL Procedure Parameters

Use the parameter declaration to specify the parameters that can be passed into or out of a procedure.



Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
( [
  [ IN | OUT | IN OUT ] parameter_name
  { parameter_type | ARRAY OF parameter_type }, ...
] )
[ DECLARE variable_declaration;...[;] ]
BEGIN
  procedure_body_statement;...[;]
END
```

You must include parentheses following the procedure name even if the procedure has no parameters.

Each procedure parameter has a mode, which can be IN, OUT, or IN OUT. Depending on the mode you choose for your parameters, you can use them in different ways.

An IN parameter is a read-only variable. You can use an IN parameter in expressions to help calculate a value, but you cannot assign a value to the parameter. IN parameters are useful for passing variable values into a procedure that you do not want to change within the procedure. This is the default if you do not specify the parameter mode.

An OUT parameter is a write-only variable. You can use an OUT parameter to assign a value to the parameter, but you cannot read from it within the body of the procedure. Therefore, this type of parameter cannot be used in an expression. OUT parameters are useful for passing values that are computed within a procedure out of the procedure.

An IN OUT parameter is a read and write variable, with none of the constraints of an IN or OUT parameter. This is useful for variables you want to change within the procedure and pass out of the procedure.

The *parameter_name* must be unique within the procedure and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

The *parameter_type* defines the type of data the parameter can pass into or out of the procedure. The data type can be any valid ObjectServer data type as described in Table 53 on page 136, except VARCHAR or INCR.



Example

```
CREATE PROCEDURE add_or_concat
( IN counter INTEGER, IN one_char_string CHAR(1) )
```

An `ARRAY OF parameter_type` is an array of any valid parameter type.



Example

In the following example an array of integers is passed into the procedure and used to calculate the average severity of a subset of alerts:

```
CREATE PROCEDURE calculate_average_severity
  ( IN severity_arr ARRAY OF INTEGER)
```

An array of integers is passed into the procedure when you execute it. These integers are assigned to an array named `severity_arr`.

SQL Procedure Variable Declarations

In the optional `DECLARE` section of a procedure, you can define (declare) local variables for use within a procedure. A local variable is a placeholder for values used during the execution of the procedure.



Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name ([ procedure_parameter, ... ])
  [ DECLARE variable_declaration;...[;] ]
  BEGIN
    procedure_body_statement; ... [;]
  END
```

Local variable declarations must be separated by semi-colons. Variable names must be unique within the procedure and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

The variable declaration can include simple variables:

```
variable_name variable_type
```

or array variables:

```
variable_name variable_type [ ARRAY ] [ integer ]
```

Define the size of an array by specifying an integer value greater than 1 in square brackets.



Note: The square brackets in bold type around the integer value are required to specify the size of the array; they do not indicate syntax notation for an optional keyword or clause.

A *variable_type* is any valid ObjectServer data type as described in Table 53 on page 136, except `VARCHAR` or `INCR`.



Example

To create a boolean variable used in the procedure to indicate when a severity exceeds a particular value:

```
DECLARE SeverityTooHigh BOOLEAN;
```

To create an array of twenty integer values used in the procedure to store node names:

```
DECLARE NodeNameArray INTEGER [20];
```

SQL Procedure Body

The body of a procedure contains a set of SQL statements and programming constructs that manipulate data in the ObjectServer. The body of a procedure is enclosed between the keywords BEGIN and END. Each statement, except the last one, must be separated by a semi-colon.



Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name ([ procedure_parameter, ... ])
[ DECLARE variable_declaration; ... [;] ]
BEGIN
    procedure_body_statement; ... [;]
END
```

Statements in the procedure can include SQL commands and additional programming constructs.

You can execute the following SQL commands in a procedure:

```
ALTER SYSTEM BACKUP
ALTER SYSTEM SET
ALTER SYSTEM DROP CONNECTION
ALTER TRIGGER
ALTER TRIGGER GROUP
ALTER USER
UPDATE
INSERT
DELETE
WRITE INTO
RAISE SIGNAL
{ EXECUTE | CALL } PROCEDURE
```

The user creating the procedure must have appropriate permissions to execute the commands in the procedure body.



Warning: You cannot have circular dependencies in procedures or triggers; for example, you must not create a procedure that calls a procedure which then calls the original procedure.

You can use the following additional programming constructs, described next, in the procedure body:

- SET assignment statement
- IF THEN ELSE statement
- CASE WHEN statement
- FOR EACH ROW loop
- FOR loop

Set Statement

Use a SET assignment statement to write the value of an expression to a variable or parameter.



Syntax

```
SET { parameter_name | variable_name } = expression
```

You can assign a value to a parameter, variable, or a row reference in a FOR EACH ROW loop.

The expression is any valid expression as described in *Expressions* on page 161.



Note: The value returned by the expression must be of a type compatible with the variable into which you write the value.

IF THEN ELSE Statement

The IF THEN ELSE statement performs one or more actions based on the specified conditions.



Syntax

```
IF condition THEN action_command_list
[ ELSEIF condition THEN action_command_list ]
...
[ ELSE action_command_list ]
END IF;
```

If the first condition is met (evaluates to TRUE), the commands following the THEN keyword are executed in sequence until an ELSEIF, ELSE, or END IF is reached. If the first condition is not met and there is an ELSEIF statement for which the condition is met, the commands following that ELSEIF statement are executed until the next keyword is reached. If an ELSE statement exists, and no previous conditions have been met, the statements following the ELSE are executed until the END IF is reached. Conditions are described in *Conditions* on page 162.

CASE WHEN Statement

The CASE WHEN statement performs one or more actions based on a condition. If the condition is not met, you can optionally perform a different action.



Syntax

```
CASE
  WHEN condition THEN action_command_list
  ...
  [ ELSE action_command_list ]
END CASE;
```

If the first condition is met (evaluates to TRUE), the statements following the THEN keyword are executed in sequence until a WHEN, ELSE, or END CASE is reached. Otherwise, if there is any WHEN statement for which the condition is met, the statements following the THEN keyword are executed until a WHEN, ELSE, or END CASE is reached. If no previous condition is met and there is an ELSE statement, the statements following the ELSE are executed until an END CASE is reached. Conditions are described in *Conditions* on page 162.

FOR EACH ROW Loop

The FOR EACH ROW loop performs actions on a set of rows that match a certain condition.



Syntax

```
FOR EACH ROW variable_name in database_name.table_name
  [ WHERE condition ]
BEGIN
  action_command_list;
END;
```

In this statement the variable name is declared implicitly as a row reference. Therefore you do not need to declare the variable at the start of the procedure. This means that any changes made to the columns referenced by the variable directly affect the referenced rows in the ObjectServer. When the END is reached, the implicitly declared variable is discarded and cannot be used elsewhere in the procedure.



Note: Only base tables (not views) can be updated in the `FOR EACH ROW` loop. You cannot insert into the table being processed within the `FOR EACH ROW` loop.

If you include a `WHERE` clause, only rows meeting the criteria specified in the condition are returned. Conditions are described in *Conditions* on page 162.

A `BREAK` command exits from the current loop. Execution continues with the next statement in the procedure.

A `CANCEL` command stops the execution of a procedure.



Warning: Do not use the `CANCEL` command when using a desktop ObjectServer in `DualWrite` mode. For more information on desktop ObjectServers, see the Netcool/OMNIBus Installation and Deployment Guide.



Example

To increase the severity of all alerts in the `alerts.status` table that have a severity of 3 to a severity of 4:

```
FOR EACH ROW alert_row in alerts.status WHERE alert_row.Severity=3
BEGIN
    SET alert_row.Severity = 4;
END;
```

When this statement is executed, the ObjectServer reads each row of the `alerts.status` table and tests to see if the value of `Severity` is 3. For each row that matches this condition, the statements within the `BEGIN` and `END` are executed, until all the rows are processed.

FOR Loop

The `FOR` loop performs actions a set number of times, based on a counter variable.



Syntax

```
FOR counter = 1 to integer DO
BEGIN
    action_command_list;
END;
```

A `BREAK` command exits from the current loop. Execution continues with the next statement in the procedure.

A CANCEL command stops the execution of a procedure.



Warning: Do not use the CANCEL command when using a desktop ObjectServer in `DualWrite` mode. For more information on desktop ObjectServers, see the Netcool/OMNIBus Installation and Deployment Guide.



Example

This procedure updates each row of the `alerts.status` table and sets the acknowledged flag to `TRUE`:

```
CREATE PROCEDURE ACKNOWLEDGE_TOOL( ids ARRAY OF CHAR(255) )
  DECLARE
    k INTEGER;
  BEGIN
    FOR k = 1 TO ARRAY_LEN( ids ) DO
      BEGIN
        UPDATE alerts.status VIA ( ids[k] ) SET Acknowledged = TRUE;
      END;
    END;
  END;
```

Using Implicit User Variables in Procedures and Triggers

You can use user variables to access information about connected users within an SQL expression in the body of a trigger or procedure. Use the `%user` notation to specify user variables, for example: `%user.attribute_name`.

The `%` symbol indicates that you are referencing an implicit variable. The `user` keyword references the current user. Table 71 lists the read-only attributes available in procedures and triggers.

Table 71: Implicit User Variables (1 of 2)

Variable Attribute	Data Type	Description
<code>%user.user_id</code>	INTEGER	User identifier of the connected user.
<code>%user.user_name</code>	STRING	Name of the connected user.
<code>%user.app_name</code>	STRING	Name of the connected application (such as <code>nco_sql</code>).
<code>%user.host_name</code>	STRING	Name of the connected host.
<code>%user.connection_id</code>	UNSIGNED	Connection identifier.
<code>%user.is_auto</code>	BOOLEAN	If <code>TRUE</code> , the current action was caused by the execution of an automation (such as a temporal trigger).

Table 71: Implicit User Variables (2 of 2)

Variable Attribute	Data Type	Description
<code>%user.is_gateway</code>	BOOLEAN	If TRUE, the current action was caused by a gateway client.
<code>%user.is_eventlist</code>	BOOLEAN	If TRUE, the current action was caused by an event list client.



Example

To reference the name of the current user in the body of a procedure or trigger, use the syntax:

```
%user.user_name
```

Creating External Procedures

You can create external procedures to run an executable on a local or remote system.



Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
  ( [ parameter_name
    { parameter_type | ARRAY OF parameter_type | ROW OF database_name.table_name },... ] )
  EXECUTABLE 'executable_name'
  HOST 'host_name'
  USER user_id
  GROUP group_id
  [ ARGUMENTS expression,... ] [;]
```

If a procedure may already exist with the same name as the one you want to create, use the optional OR REPLACE keywords. If the procedure exists, it is replaced by the one you are creating. If the procedure does not exist, a new one is created.

The procedure name must be unique within the ObjectServer and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

Use the parameter declaration to specify the parameters that can be passed into the external procedure. You must include parentheses following the procedure name even if the procedure has no parameters.

The parameter names must be unique within the procedure and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.



Tip: External procedure parameters are read-only. They allow you to pass variable values into an external procedure. You cannot return values from an external procedure.

The *parameter_type* defines the type of data the parameter can pass into the procedure. The data type can be any valid ObjectServer data type as described in Table 53 on page 136, except VARCHAR or INCR.

The executable is the path to an executable on a local or remote file system.

The host is the name of the host on which to execute the procedure executable.

The user is the effective user ID under which to execute the executable.

The group is the effective group ID under which to execute the executable.

The arguments are those passed to the executable.



Example

The following external procedure calls a program called `nco_mail`, which sends e-mail about unacknowledged critical alerts:

```
create or replace procedure send_email
(in node character(255), in severity integer, in subject character(255),
 in email character(255), in summary character(255), in hostname character(255))
executable '$OMNIHOME/utils/nco_mail'
host 'localhost'
user 0
group 0
arguments '\\' + node + '\\', severity, '\\' + subject + '\\',
          '\\' + email + '\\', '\\' + summary + '\\';
```

This example also shows how to pass text strings to an executable. Strings must be enclosed in quotes, and the quotes must be escaped with backslashes. All quotes in this example are single quotes.

An example trigger that calls this procedure is shown in *Trigger to E-mail on Critical Alerts* on page 222.



Note: To execute an external procedure, you must have a process control agent daemon (`nco_pad`) running on the host where the executable command resides. For more information on process control, see Chapter 4: *Process Control* on page 223.

Executing Procedures

Once you have created a procedure, you must execute it using the EXECUTE PROCEDURE command for the actions in the procedure to occur. You can do this using `nco_sql` or in a trigger or procedure.



Syntax

```
{ EXECUTE | CALL } [ PROCEDURE ] procedure_name  
[ ( expression, ... ) | ( [ expression, expression, ... ] , ... ) ];
```

Each of the expressions passed as actual parameters must resolve to an assignable value which matches the type of the parameter specified when the procedure was created. For more information about procedure parameters, see *SQL Procedure Parameters* on page 191.



Note: If you are passing an array parameter, the square brackets around the expression list, shown in bold type in the syntax description above, are not optional.



Example

To execute the procedure described in *Creating External Procedures* on page 198, use the following call in a trigger:

```
execute send_email( critical.Node, critical.Severity, 'Netcool E-mail',  
'root@localhost', critical.Summary, 'localhost');
```

An example trigger that calls this procedure is shown in *Trigger to E-mail on Critical Alerts* on page 222.

Dropping Procedures

If you no longer need a procedure, you can remove it using the DROP PROCEDURE command.



Syntax

```
DROP PROCEDURE procedure_name;
```

You cannot drop a procedure if it is referenced by other objects, such as triggers.



Example

```
drop procedure testproc;
```

3.11 Automation: Triggers and Trigger Groups

You can use automation to detect changes in the ObjectServer and execute automated responses to these changes. This enables the ObjectServer to process alerts without requiring an operator to take action.

For example, network alerts often include the message `Link Down`, which indicates that there is a problem in the network. When the problem is solved, the ObjectServer receives another alert including the message `Link Up`. Using a correctly configured automation, the ObjectServer can automatically associate the two alerts, recognize that the `Link Up` message indicates that the problem is solved, and delete both alerts.

You can also use automation to manage deduplication, which reduces the quantity of data held in the ObjectServer by eliminating duplicate alerts.

Triggers form the basis of the ObjectServer automation subsystem. Triggers enable the ObjectServer to automatically fire (execute a trigger action) when the ObjectServer detects an incident associated with a trigger. In a trigger, you can execute SQL commands and call procedures in response to the change.

Every trigger belongs to a *trigger group*, which is a collection of related triggers. You can easily enable or disable all triggers in a given group.

Some example automations are shown in *Automation Examples* on page 220.

Types of Triggers

The syntactic elements common to all types of triggers are described in *Syntax Elements Common to All Types of Triggers* on page 203. Depending on the trigger type, there are additional syntax elements that enable you to customize the trigger action.

A *database trigger* is configured to spontaneously fire when a triggering database incident occurs. For example, you can create a trigger to perform an action each time a *reinsert* occurs (an attempt is made to insert a row into a table, but a row with the same value for the `Identifier` primary key already exists) on the `alerts.status` table. Database triggers are described in *Database Triggers* on page 207.

A *temporal trigger* is configured to fire repeatedly based on a specified frequency. For example, you can use a temporal trigger to delete all clear rows (`Severity = 0`) from the `alerts.status` table that have not been modified within a certain period of time. Temporal triggers are described in *Temporal Triggers* on page 210.

A *signal trigger* is configured to fire when a system or user defined signal is raised. For example, you can send an e-mail to an operator when the ObjectServer starts or stops because system signals are generated. A user defined signal is one you define and then raise using the `RAISE SIGNAL` command. Signal triggers are described in *Signals and Signal Triggers* on page 211.

Trigger Groups

Every trigger belongs to a trigger group, which is a collection of related triggers.

Adding a Trigger Group: The CREATE TRIGGER GROUP Command

Use the `CREATE TRIGGER GROUP` command to create a new trigger group.



Syntax

```
CREATE TRIGGER GROUP trigger_group_name;
```

The trigger group name must be unique within the ObjectServer and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.



Example

```
create trigger group update_database_triggers;
```

Modifying a Trigger Group: The ALTER TRIGGER GROUP Command

Use the `ALTER TRIGGER GROUP` command to activate or deactivate an existing trigger group.



Syntax

```
ALTER TRIGGER GROUP trigger_group_name  
SET ENABLED { TRUE | FALSE };
```

A trigger group is enabled by default.



Example

```
alter trigger group update_database_triggers set enabled false;
```

Deleting a Trigger Group: The DROP TRIGGER GROUP Command

Use the `DROP TRIGGER GROUP` command to drop an existing trigger group.



Syntax

```
DROP TRIGGER GROUP trigger_group_name;
```

You cannot drop a trigger group if it contains any triggers.



Example

```
drop trigger group update_database_triggers;
```

Syntax Elements Common to All Types of Triggers

Database, temporal, and signal triggers have many common syntax elements, described in this section.



Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string' ]
...trigger syntax depending on the type of trigger...
[ WHEN condition ]
[ DECLARE variable_declaration ]
BEGIN
    trigger_statement_list
END;
```

If a trigger may already exist with the same name as the one you want to create, use the optional OR REPLACE keywords. If the trigger exists, it is replaced by the one you are creating. If the trigger does not exist, a new one is created.

The trigger name must be unique within the ObjectServer and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126.

The group name can be any trigger group already created using the CREATE TRIGGER GROUP command.

If DEBUG is set to TRUE, debugging information is sent to the ObjectServer message log, if the message level is set to debug.

If ENABLED is set to TRUE, the trigger fires when the associated incident occurs. Otherwise, the trigger does not fire when the incident occurs.

A trigger's `PRIORITY` determines the order in which the ObjectServer fires triggers when more than one trigger is associated with the same incident. The priority can be in the range of 1 to 20. The lower the number, the higher the priority, so a trigger with a priority of 2 is fired before a trigger with a priority of 3. If more than one trigger of the same priority is fired because of the same incident, the order in which these triggers fire is undetermined.

The optional `COMMENT` keyword enables you to add a comment for the trigger.

The syntax that follows the comment and precedes the optional `WHEN` clause and the action depends on the trigger type, as described in the following sections.

The optional `WHEN` clause allows you to test for a particular condition before the `action` is executed. If the condition is not met, the `action` is not executed. Conditions are described in *Conditions* on page 162.

You can optionally declare local trigger variables for use in the body of the trigger. These variables are declared and used in the same way as procedure variables, as described in *SQL Procedure Variable Declarations* on page 192. However, trigger variables are static, so they maintain their value between executions of the trigger.

The *trigger_statement_list* is a set of statements that are executed when the trigger is fired.

Executing Commands in Trigger Actions

The *trigger_statement_list* contains a set of commands that manipulate data in the ObjectServer. You can execute the following SQL commands in a trigger:

```
ALTER SYSTEM BACKUP
ALTER SYSTEM DROP CONNECTION
ALTER SYSTEM SET
ALTER TRIGGER
ALTER TRIGGER GROUP
ALTER USER
UPDATE
INSERT
DELETE
WRITE INTO
RAISE SIGNAL
{ EXECUTE | CALL } PROCEDURE
```

The user creating the trigger must have appropriate permissions to execute the commands in the trigger body.



Warning: You cannot have circular dependencies in triggers or procedures; for example, you must not create a trigger that calls a procedure which then causes the original trigger to fire.

You can use the following additional programming constructs in a trigger:

- The SET assignment statement, described in *Set Statement* on page 194
- The IF THEN ELSE statement, described in *IF THEN ELSE Statement* on page 194
- The CASE WHEN statement, described in *CASE WHEN Statement* on page 195
- The FOR EACH ROW loop, described in *FOR EACH ROW Loop* on page 195
- The FOR loop, described in *FOR Loop* on page 196



Example

This system signal trigger logs the name of each user who connects to the ObjectServer to a file.

```
CREATE TRIGGER LogConnections
GROUP default_triggers
PRIORITY 1
ON SIGNAL connect
BEGIN
  WRITE INTO file1 VALUES ('User', %user.user_name, 'has logged on.');
```

END;

For more information on implicit user variables like %user.user_name, see *Using Implicit User Variables in Procedures and Triggers* on page 197.

Using Trigger Variables

You can use trigger variables to access information about the current and previous executions of the trigger. Use the %trigger notation to specify trigger variables. The % symbol indicates that you are referencing an implicit variable. The trigger keyword references the current trigger. For example, to reference the previous trigger row count, use the syntax:

```
%trigger.previous_rowcount
```

Table 72 lists the read-only attributes available in the WHEN clause and action section of a trigger.

Table 72: Implicit Trigger Variables (1 of 2)

Trigger Attribute	Data Type	Description
%trigger.previous_condition	BOOLEAN	Value of the condition on last execution.
%trigger.previous_rowcount	UNSIGNED	Number of rows returned by the EVALUATE clause the last time the trigger was raised.
%trigger.num_positive_rowcount	UNSIGNED	Number of consecutive fires with one or more matches in EVALUATE clause.

Table 72: Implicit Trigger Variables (2 of 2)

Trigger Attribute	Data Type	Description
<code>%trigger.num_zero_rowcount</code>	UNSIGNED	Number of consecutive fires with zero matches in EVALUATE clause.



Note: In a database trigger, the only valid trigger variable is `%trigger.previous_condition`. All other trigger variables are used in an EVALUATE clause, which is not supported for database triggers.

ALTER TRIGGER command

Use the ALTER TRIGGER command to change the settings of an existing trigger.



Syntax

```
ALTER TRIGGER trigger_name
  SET PRIORITY integer
  SET ENABLED { TRUE | FALSE }
  SET GROUP trigger_group_name
  SET DEBUG { TRUE | FALSE };
```

Use SET PRIORITY to change the priority of a trigger to a value between 1 and 20. The lower the number, the higher the priority.

Use SET ENABLED to activate or deactivate a trigger. If a trigger is ENABLED, it fires when the associated incident occurs. If a trigger is not ENABLED, it does not fire when the associated incident occurs.

Use SET GROUP to change the trigger group of the trigger to the specified group name.

Use SET DEBUG to turn debugging on or off for the trigger. If on, debugging information is sent to the ObjectServer message log, if the message level is set to debug.

You can change more than one setting in a single ALTER TRIGGER command.



Example

```
alter trigger mytrig set priority 1;
```

DROP TRIGGER command

Use the DROP TRIGGER command to drop an existing trigger.



Syntax

```
DROP TRIGGER trigger_name;
```

You cannot drop a trigger if it has any dependant objects.



Example

```
drop trigger mytrig;
```

Database Triggers

You can create database triggers that fire when a modification or attempted modification to an ObjectServer table occurs (or when a modification or attempted modification to a view affects a base table).

You can create triggers that fire if any of the following database changes occur:

- An attempt is made to insert a row into a table.
- An attempt is made to update a row in a table.
- An attempt is made to delete a row from a table.
- An attempt is made to insert a row into a table, but a row with the same value for the Identifier primary key already exists. You can use a reinsert trigger to deduplicate rows in the ObjectServer.



Note: Deduplication is described in *Alert Processing in the ObjectServer* on page 124. You can create your own deduplication trigger to cause a different action to occur. An example reinsert trigger is described in *Trigger to Deduplicate the Status Table* on page 220.

Creating Database Triggers

The syntax of the command to create a database trigger is:



Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string' ]
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE | REINSERT } ON database_name.table_name
FOR EACH { ROW | STATEMENT }
```

```
[ WHEN condition ]  
[ DECLARE variable_declaration ]  
trigger_action;
```

For more information on the common elements of the CREATE TRIGGER command, see *Syntax Elements Common to All Types of Triggers* on page 203.


The BEFORE or AFTER timing keyword specifies whether the trigger is executed before or after the database modification that caused the trigger to fire occurs. For example, you can create a BEFORE trigger that evaluates the user's name before a row in the alerts.status is deleted. In the trigger, you can detect whether the user is allowed to delete from the alerts.status table, and if not, prevent the database modification from taking place. With an AFTER trigger, the database modification always takes place.

The *database_name.table_name* is the name of the database and table affected by the trigger action.

The level at which a database trigger fires is one of the following:

- FOR EACH ROW (known as a *row-level trigger*)
- FOR EACH STATEMENT (known as a *statement-level trigger*).


Row-level triggers fire once for each row returned as a result of the database modification. Statement-level triggers fire once for each database modification.

 **Note:** Only row-level triggers can be defined to fire on inserts and reinserts.

For example, a database signal is raised as a result of the following SQL statement:

```
DELETE FROM alerts.status WHERE Severity = 5;
```

When this statement is executed, the ObjectServer deletes all the rows in the alerts.status table with a severity of 5. If there are 20 rows in the table with this severity and the level is set to FOR EACH ROW, 20 rows are deleted and the trigger is raised 20 times. If the level is set to FOR EACH STATEMENT, the trigger is raised once.

 **Note:** BEFORE statement-level triggers always fire before BEFORE row-level triggers, and AFTER statement-level triggers always fire after AFTER row-level triggers, regardless of trigger priority.

NEW and OLD Implicit Variables in Row-Level Triggers

In addition to the local variables declared in the trigger, row-level triggers have access to implicit variables whose values are automatically set by the system. The **OLD** variable refers to the value of a column before the incident occurs; the **NEW** variable refers to a column affected by the incident, after it has occurred. You can use expressions to read from and assign values to row variables.

Certain operations on the **NEW** or **OLD** row variables may not be accessible or modifiable depending on the type of modification. For example, if the ObjectServer deletes a row, there is no **NEW** row to read or modify. Table 73 shows when the **NEW** and **OLD** variables are available depending on the database operation.

Table 73: Availability of Special Row Variables

Operation	Timing Mode	Is the NEW Variable Available?	Is the NEW Variable Modifiable?	Is the OLD Variable Available?	Is the OLD Variable Modifiable?
INSERT	BEFORE	Y	Y	N	N
INSERT	AFTER	Y	N	N	N
UPDATE	BEFORE	Y	Y	Y	N
UPDATE	AFTER	Y	N	N	N
DELETE	BEFORE	N	N	Y	N
DELETE	AFTER	N	N	Y	N
REINSERT	BEFORE	Y	N	Y	Y
REINSERT	AFTER	Y	N	N	N

The following database trigger uses the **NEW** variable to update the `StateChange` column when a row in the `alerts.status` table is modified to timestamp the change.



Example

```
create trigger SetStateChange
group default_triggers
priority 1
before update on alerts.status
for each row
begin
    set new.StateChange = getdate;
end;
```

Temporal Triggers

Temporal triggers fire at a specified frequency.

Create Temporal Trigger Syntax Definition

The syntax of the command to create a temporal trigger is:



Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string'
EVERY integer { HOURS | MINUTES | SECONDS }
[ EVALUATE SELECT_cmd BIND AS variable_name ]
[ WHEN condition ]
[ DECLARE variable_declaration ]
trigger_action;
```

For more information on the common elements of the CREATE TRIGGER command, see *Syntax Elements Common to All Types of Triggers* on page 203.

Within a temporal trigger, you must specify how often the trigger will fire in seconds (the default unit of time), minutes, or hours.

The optional EVALUATE clause enables you to build a temporary result set from a single SELECT statement to be processed in the *trigger_action*. The SELECT statement cannot contain an ORDER BY clause.

The following temporal trigger deletes all clear rows (`Severity = 0`) from the `alerts.status` table that have not been modified in the last two minutes.



Example

```
create trigger
DeleteClears
group my_triggers
priority 1
every 60 seconds
begin
    delete from alerts.status where Severity = 0
        and StateChange < (getdate - 120);
end;
```


Signals and Signal Triggers

Signal triggers fire when a predefined system signal is raised or when a user defined signal is raised using the `RAISE SIGNAL` command.

The types of signals are:

- System signals
- User defined signals

System signals are raised spontaneously by the ObjectServer when it detects changes to the system. You do not have to do anything to create or configure them. You can attach triggers to them to create automatic responses to incidents in the ObjectServer. System signals are described in *System Signals* on page 212.

User defined signals are defined by you. You must explicitly create, raise, and drop user defined signals, as described in *Creating a User Defined Signal* on page 217, *Raising a User Defined Signal* on page 219, and *Dropping a User Defined Signal* on page 219, respectively.

Create Signal Trigger Syntax Definition

The syntax of the command to create a signal trigger is:



Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string' ]
ON SIGNAL { system_signal_name | user_signal_name }
[ EVALUATE SELECT_cmd BIND AS variable_name ]
[ WHEN condition ]
[ DECLARE variable_declaration ]
trigger_action;
```

For more information on the common elements of the `CREATE TRIGGER` command, see *Syntax Elements Common to All Types of Triggers* on page 203.

The `ON SIGNAL` name can be the name of a system or user defined signal that fires the trigger.

The optional `EVALUATE` clause enables you to build a temporary result set from a single `SELECT` statement to be processed in the *trigger_action*. The `SELECT` statement cannot contain an `ORDER BY` clause.

When a system or user defined signal is raised, attributes that identify the cause of the signal are attached to the signal. These attributes are passed as implicit variables into the associated signal trigger, and are described next.

System Signals

System signals are raised spontaneously by the ObjectServer when it detects changes to the system. You do not have to do anything to create or configure them.

When a system signal is raised, attributes that identify the cause of the signal are set. These attributes are passed as implicit variables into the associated signal trigger. You can refer to system signal variables using the `%signal` notation in the action section of a signal trigger. The `%` symbol indicates that you are referencing an implicit variable. The `signal` keyword references the signal currently passed to the trigger. For example, to reference the time at which a system signal was raised in a signal trigger, use the syntax:

```
%signal.at
```

Table 74 describes the system signals that can be raised by the ObjectServer and the associated attributes.

Table 74: System Signals and Their Attributes (1 of 6)

Signal	Attributes	Data Type	Description
startup	<i>server</i>	string	Indicates the name of the ObjectServer that started.
	<i>node</i>	string	Indicates the machine on which the ObjectServer started.
	<i>at</i>	UTC	Indicates the time at which the ObjectServer started.
shutdown	<i>server</i>	string	Indicates the name of the ObjectServer that shut down.
	<i>node</i>	string	Indicates the machine on which the ObjectServer shut down.
	<i>at</i>	UTC	Indicates the time at which the ObjectServer shut down.

Table 74: System Signals and Their Attributes (2 of 6)

Signal	Attributes	Data Type	Description
connect	<i>process</i>	string	Indicates the type of client process connected to the ObjectServer.
	<i>description</i>	string	Contains additional information about the client that connected, where available. For example, if the client is a probe, the description contains the probe name.
	<i>username</i>	string	Indicates the name of the user connected to the ObjectServer.
	<i>node</i>	string	Indicates the name of the client machine connected to the ObjectServer.
	<i>at</i>	UTC	Indicates the time at which the client connected.
disconnect	<i>process</i>	string	Indicates the type of process that disconnected from the ObjectServer.
	<i>description</i>	string	Contains additional information about the client that disconnected, where available. For example, if the client is a probe, the description contains the probe name.
	<i>username</i>	string	Indicates the name of the user that disconnected from the ObjectServer.
	<i>node</i>	string	Indicates the name of the client machine that disconnected from the ObjectServer.
	<i>at</i>	UTC	Indicates the time at which the client disconnected.
backup_failed	<i>error</i>	string	Indicates a reason that the backup attempt failed.
	<i>at</i>	UTC	Indicates the time at which the backup attempt occurred.
	<i>path_prefix</i>	string	Indicates the directory to which the backup attempted to write.
	<i>elapsed_time</i>	real	Indicates the amount of time the backup was running before it failed.
	<i>node</i>	string	Indicates the name of the machine from which the backup was run.

Table 74: System Signals and Their Attributes (3 of 6)

Signal	Attributes	Data Type	Description
backup_succeeded	<i>at</i>	UTC	Indicates the time at which the backup occurred.
	<i>path_prefix</i>	string	Indicates the directory to which the backup was written.
	<i>elapsed_time</i>	real	Indicates the amount of time the backup took to complete.
	<i>node</i>	string	Indicates the name of the machine from which the backup was run.
license_lost	<i>server</i>	string	Indicates the name of the ObjectServer on which a license was lost.
	<i>node</i>	string	Indicates the machine on which a license was lost.
	<i>at</i>	UTC	Indicates the time at which a license was lost.
login_failed	<i>process</i>	string	Indicates the name of the process that could not connect because the login was denied.
	<i>username</i>	string	Indicates the name of the user that failed to connect because login was denied.
	<i>node</i>	string	Indicates the name of the client machine that could not connect because the login was denied.
	<i>at</i>	UTC	Indicates the time at which the client failed to connect because the login was denied.
security_timeout	<i>process</i>	string	Indicates the name of the process that failed to connect because login credentials could not be validated.
	<i>username</i>	string	Indicates the name of the user that failed to connect because login credentials could not be validated.
	<i>node</i>	string	Indicates the name of the client machine that failed to connect because login credentials could not be validated.
	<i>at</i>	UTC	Indicates the time at which the client failed to connect because login credentials could not be validated.

Table 74: System Signals and Their Attributes (4 of 6)

Signal	Attributes	Data Type	Description
create_object	<i>objecttype</i>	string	Indicates the object type, which is one of the following: CREATE DATABASE CREATE TABLE CREATE TRIGGER GROUP CREATE TRIGGER CREATE PROCEDURE CREATE RESTRICTION FILTER CREATE USER SIGNAL CREATE FILE CREATE USER CREATE GROUP CREATE ROLE
	<i>parentname</i>	string	Indicates the name of the parent object. For triggers, this is the trigger group name. For tables, this is the database name. Other objects do not have a parent object.
	<i>name</i>	string	Indicates the name of the object. For example, the value for the <code>alerts.status</code> table is <code>status</code> .
	<i>username</i>	string	Indicates the name of the user that executed the command.
	<i>server</i>	string	Indicates the name of the ObjectServer to which the object was added.
	<i>node</i>	string	Indicates the machine from which the request was made.
	<i>at</i>	UTC	Indicates the time at which the object was added.

Table 74: System Signals and Their Attributes (5 of 6)

Signal	Attributes	Data Type	Description
alter_object	<i>objecttype</i>	string	Indicates the object type, which is one of the following: ALTER TABLE ALTER TRIGGER GROUP ALTER TRIGGER ALTER FILE ALTER USER ALTER GROUP ALTER ROLE
	<i>parentname</i>	string	Indicates the name of the parent object. For triggers, this is the trigger group name. For tables, this is the database name. Other objects do not have a parent object.
	<i>name</i>	string	Indicates the name of the object. For example, the value for the <code>alerts.status</code> table is <code>status</code> .
	<i>username</i>	string	Indicates the name of the user that executed the command.
	<i>server</i>	string	Indicates the name of the ObjectServer in which the object was altered.
	<i>node</i>	string	Indicates the machine from which the request was made.
	<i>at</i>	UTC	Indicates the time at which the object was altered.

Table 74: System Signals and Their Attributes (6 of 6)

Signal	Attributes	Data Type	Description
drop_object	<i>objecttype</i>	string	Indicates the object type, which is one of the following: DROP DATABASE DROP TABLE DROP TRIGGER GROUP DROP TRIGGER DROP PROCEDURE DROP RESTRICTION FILTER DROP USER SIGNAL DROP FILE DROP USER DROP GROUP DROP ROLE
	<i>parentname</i>	string	Indicates the name of the parent object. For triggers, this is the trigger group name. For tables, this is the database name. Other objects do not have a parent object.
	<i>name</i>	string	Indicates the name of the object. For example, the value for the <code>alerts.status</code> table is <code>status</code> .
	<i>username</i>	string	Indicates the name of the user that executed the command.
	<i>server</i>	string	Indicates the name of the ObjectServer from which the object was dropped.
	<i>node</i>	string	Indicates the machine from which the request was made.
	<i>at</i>	UTC	Indicates the time at which the object was dropped.



Tip: You can query the `catalog.primitive_signals` and `catalog.primitive_signal_parameters` tables to view information about system signals. For example, to view the attributes of each system signal, use the following SQL command:
`SELECT * FROM catalog.primitive_signal_parameters ORDER BY SignalName, OrdinalPosition;`

Creating a User Defined Signal

User defined signals are explicitly created, raised, and dropped.

Use the `CREATE SIGNAL` command to create a user defined signal. When you create a signal, you define a list of data-typed attributes, as follows:



Syntax

```
CREATE [ OR REPLACE ] SIGNAL signal_name
  [ (signal_attribute_name data_type, ...) ]
  [ COMMENT 'comment_string' ]
```

The signal name must be unique within the ObjectServer and comply with the naming conventions described in *ObjectServer Naming Conventions* on page 126. You cannot create a user defined signal with the same name as a system signal.

When you define attributes, specify the attribute name and any valid ObjectServer data type listed in Table 53 on page 136, except VARCHAR or INCR.

You can add a comment following the optional COMMENT keyword.



Example

To create a signal called `illegal_delete` with two character string attributes, `user_name` and `row_summary`, use the command:

```
CREATE SIGNAL illegal_delete( user_name char(40), row_summary char(255) );
```

You could then create a trigger, such as the following pre-insert database trigger, to trap deletes that occur outside of standard office hours and raise this signal.

```
create trigger DETECT_AN_ILLEGAL_DELETE
group default_triggers
priority 1
before delete on alerts.status
for each row
begin
  if( ( hourofday() > 17) and (minuteofhour() > 30) ) or (hourofday() < 9) ) then
    raise signal ILLEGAL_DELETE %user.user_name, old.Summary;
    cancel;
  end if;
end;
```

For more information on database triggers, see *Database Triggers* on page 207.

The following user defined signal trigger, triggered by the preceding database trigger, executes an external procedure to send mail notification of the attempted delete.

```
create trigger AFTER_HOURS_DELETE_WARNING
group default_triggers
priority 1
on signal ILLEGAL_DELETE
begin
```



```
execute MAIL_THE_BOSS( 'User ' + '%signal.user_name ' + 'attempted to remove the
row ' + %signal.row_summary + ' at ' +to_char(getdate) )
end;
```

Raising a User Defined Signal

Use the `RAISE SIGNAL` command to raise a user defined signal.



Syntax

```
RAISE SIGNAL signal_name expression,...;
```

The expressions must resolve to a value compatible with the data type of the associated attribute as defined using the `CREATE SIGNAL` command.



Example

To raise the user defined signal described in the preceding section:

```
RAISE SIGNAL illegal_delete %user.user_name, old.Summary;
```

Dropping a User Defined Signal

Use the `DROP SIGNAL` command to drop a user defined signal.



Syntax

```
DROP SIGNAL signal_name;
```

You cannot drop a signal if a trigger references it.

3.12 Automation Examples

This section contains examples of some commonly performed automations. These and other standard automations are created by the `automation.sql` script during database initialization.



Tip: Many of the automations created by the `automation.sql` script are disabled by default.

Trigger to Deduplicate the Status Table

This database trigger intercepts an attempted reinsert on the `alerts.status` table and increments the tally to show that a new row of this kind has arrived at the ObjectServer. It also sets the `LastOccurrence` field.

```
create or replace trigger deduplication
group default_triggers
priority 1
comment 'Deduplication processing for ALERTS.STATUS'
before reinsert on alerts.status
for each row
begin
set old.Tally = old.Tally + 1;
set old.LastOccurrence = new.LastOccurrence;
set old.StateChange = getdate();
set old.InternalLast = getdate();
set old.Summary = new.Summary;
set old.AlertKey = new.AlertKey;
if ((old.Severity = 0) and (new.Severity > 0))
then set old.Severity = new.Severity;
end if;
end;
```

Trigger to Deduplicate the Details Table

This database trigger intercepts an attempted reinsert on the `alerts.details` table.

```
create or replace trigger deduplicate_details
group default_triggers
priority 1
comment 'Deduplicate rows on alerts.details'
before reinsert on alerts.details
for each row
begin
cancel; -- Do nothing. Allow the row to be discarded
end;
```

Trigger to Clean the Details Table

This temporal trigger periodically clears detail entries in the `alerts.details` table when no corresponding entry exists in the `alerts.status` table.

```
create or replace trigger clean_details_table
group default_triggers
priority 1
comment 'Housekeeping cleanup of ALERTS.DETAILS'
every 60 seconds
begin
delete from alerts.details
      where Identifier not in (select Identifier from alerts.status);
end;
```

Trigger to Set the Alerts Table StateChange Column

When a row in the `alerts.status` table is modified, this database trigger updates the `StateChange` column to timestamp the change.

```
create or replace trigger state_change
group default_triggers
priority 1
comment 'State change processing for ALERTS.STATUS'
before update on alerts.status
for each row
begin
set new.StateChange = getdate();
end;
```

Trigger to Delete Clears

This temporal trigger deletes all clear rows (`Severity = 0`) from the `alerts.status` table that have not been modified within the last two minutes.

```
create or replace trigger delete_clears
group default_triggers
priority 1
comment 'Delete cleared alerts over 2 minutes old every 60 seconds'
every 60 seconds
begin
delete from alerts.status where Severity = 0 and StateChange < (getdate() - 120);
end;
```

Trigger to E-mail on Critical Alerts

This temporal trigger sends e-mail, by calling an external procedure, if any critical alerts are not acknowledged within 30 minutes.

```
create or replace trigger mail_on_critical
group default_triggers
priority 1
comment 'Send e-mail about critical alerts that are
unacknowledged after 30 minutes - works on UNIX systems
only unless an equivalent Windows mailer is available'
every 10 seconds
evaluate
select Node, Severity, Summary, Identifier from alerts.status where
  Severity = 5 and Grade < 2 and Acknowledged = 0 and LastOccurrence <= (getdate - (60*30))
bind as criticals
begin
for each row critical in criticals
begin
  execute send_email(critical.Node, critical.Severity, 'Netcool E-mail',
    'root@localhost', critical.Summary, 'localhost');
  update alerts.status via (critical.Identifier) set Grade=2;
end;
end;
```

The `send_email` external procedure is declared as follows and calls the `nco_mail` utility:

```
create or replace procedure send_email
(in node character(255), in severity integer, in subject character(255),
 in email character(255), in summary character(255), in hostname character(255))
executable '$OMNIHOME/utils/nco_mail' host 'hostname' user 0 group 0
arguments '\\' + node + '\\', severity, '\\' + subject + '\\',
 '\\' + email + '\\', '\\' + summary + '\\';
```

This example also shows how to pass text strings to an executable. Strings must be enclosed in quotes, and the quotes must be escaped with backslashes. All quotes in this example are single quotes.

Chapter 4: Process Control

This chapter describes the components, configuration, and management utilities associated with the Netcool/OMNIbus process control system. It includes the following sections:

- *Introduction* on page 224
- *Process Control Components* on page 226
- *Creating a Process Control System Configuration* on page 228
- *Defining Processes, Services, and Hosts* on page 231
- *Example Process Agent Configuration File* on page 238
- *Process Control Agent Daemon Command Line Options* on page 240
- *Process Control Management* on page 244

4.1 Introduction

The Netcool/OMNIBus process control system performs two primary tasks:

- Execution of external procedures specified in automations
- Management of local and remote processes



Note: On Windows systems, process control only executes external procedures specified in automations. You cannot use process control to manage processes on Windows.

Execution of External Procedures in Automations

Process control is responsible for the execution of external procedures specified in automations. An automation does not execute programs by itself. It sends a request to a local process control agent, which forwards the request to the process control agent running on the specified host. The remote process control agent then executes the requested program.



Note: Process control agents on Windows machines can only connect to process control agents on other Windows machines. Process control agents on UNIX machines can only connect to process control agents on other UNIX machines. External procedures cannot pass between these different environments.

Configuring and Managing Netcool/OMNIBus Components on UNIX

On UNIX, process control allows you to configure remote processes to simplify the management of Netcool/OMNIBus components such as ObjectServers, probes, and gateways. It consists of the following:

- Process control agents, which are programs installed on each host with the responsibility for managing processes
- A set of command line utilities to provide an interface to process management

Process control agents cooperate automatically and understand their own configuration. Process control agents start remote processes and are capable of keeping those processes running. You can define processes that are dependent on the start of other process, and those that have timed threshold dependencies. If a managed host is restarted, the process control agent restarts local components automatically.

Process Agents on UNIX

Any participating Netcool/OMNIbus UNIX host must be configured with the process control agent daemon. Once the process control agent is configured and running, the host connects into the process control system. The process control agents can then communicate with each other and execute programs upon request.

Process Agents on Windows

In a Windows network, each process agent is connected to two others to form a ring. One process agent at a time can join or leave the network without breaking the ring. All the process agents can join at the same time, not necessarily through a common node.

Joining a Process Agent into a Windows Network

A process agent joins a Windows network by connecting to an existing member. If it tries to connect and fails, it retries after the number of seconds specified in the `MaxConnect` property in its properties file (`agent-name_PA.props`). If a process agent loses one of its connections in the network, it waits for the number of seconds specified in the `MinConnect` property in this file for a reconnection from another process agent. If this does not arrive, it tries to reconnect itself to the node to which it was last connected at the interval specified in the `MaxConnect` property.

When a process agent receives a request to join the existing network, it allows a proportion of the `MinConnect` period for the requesting agent to confirm that it has joined. This proportion is specified as a percentage between 20% and 80% in the `AcceptRatio` property.

Removing a Process Agent From a Windows Network

To remove a process agent from a Windows network, stop the service from the Control Panel. Before it leaves the network, the agent notifies the other members and waits for acknowledgement for the time specified in the `MaxRingSpeed` property. If it does not receive acknowledgement, it leaves the network anyway. In this case, the network is not broken even if that was the intention in removing the agent.

The appropriate values for the properties `MinConnect`, `MaxRingSpeed`, and `AcceptRatio` are a function of network speed. They should be larger for slower networks to allow less stringent recovery criteria.



Warning: Do not power cycle or reboot more than one Windows host in a process agent network at a time. Doing this can break the network in two. You cannot restore a divided network automatically.

4.2 Process Control Components

Process control consists of:

- Process control agents and associated configuration files
- Processes
- Services, in which processes are organized and run
- Utilities to help you manage the process control agent, processes, and services

These components are described in the following sections.

Process Control Agents

Process control agents are programs installed on each host to manage processes. Any participating host must have a process control agent daemon and an associated configuration file installed.

There can be any number of process control agents on any number of hosts. Process control agents can manage any number of processes.

Processes

Processes are programs that are executed by a process control agent on the same machine. Processes must be defined in a *service*, described on page 227.

Process Control Awareness

A *PA aware* process is one that is part of the process control configuration and is aware of process control. All process control features, such as process dependencies, can be utilized. For example, ObjectServers, probes, and gateways are PA aware. A process that is not PA aware can be managed by process control, but cannot use all process control features. For example, the Netcool/OMNIbus desktop is not PA aware. For information about how to set the process type to PA aware, see *Defining Processes* on page 231.

Dependent Processes

Sometimes the order in which applications are started is important. You can use process control to configure processes to be dependent on each other *if they are in the same service*. For example, a process can be configured to start only after another process has started and completed various startup tasks.

A PA aware process communicates with the process control agent. Once the process has reached the point in its startup when it recognizes itself to be running, it sends a message to the process control agent. When the process control agent receives this message, it starts dependent processes.

Services

Processes must be defined in services. You can group related processes in a service to make them easier to manage. Once a service is correctly configured, it can be managed by process control.

A service is made up of one or more processes that are executed by process control agents. A service can be configured to start up automatically when the process control agent starts. Alternatively, it can be started manually.

A service can either be a master or a non-master. When started automatically by process control, master services are started before non-master services.

Process Control Utilities

Table 75 describes the command line utilities you can use to manage process control.

Table 75: Process Control Command Line Utilities

Utility	Description	See...
nco_pa_status	Retrieves and displays the status of services and processes being controlled by the process control agent.	page 244
nco_pa_start	Starts a service or process located anywhere in the configuration.	page 245
nco_pa_stop	Stops a service or process located anywhere in the configuration.	page 246
nco_pa_shutdown	Shuts down a process control agent.	page 247
nco_pa_addentry	Adds a service or process entry while a process control agent is running.	page 248

4.3 Creating a Process Control System Configuration

This section describes how to create a new process control configuration on UNIX.

Before You Configure Process Control

Before you begin configuring process control:

- Ensure that the process control software is installed. See the Netcool/OMNIbus Installation and Deployment Guide for additional information.
- Determine which Netcool/OMNIbus components are installed and where they are located. Ensure you have taken into account all components and any failover or backup systems. Netcool/OMNIbus desktops are not managed by process control.

Once you have determined the process control configuration requirements, follow the instructions in the following sections.

Creating User Groups

By default, the process control system uses UNIX user names and passwords to grant access. You can specify other supported authorization modes using the `-authenticate` command line option, described in *Process Control Agent Daemon Command Line Options* on page 240.

The process control daemon controls who can log in to it. Any user who needs access to the process control system must be a member of a UNIX user group that you identify as an administrative group for this purpose.

You can use an existing UNIX user group or create a new one, and add process control users to this group. If you run NIS, NIS+, or some other global information service, this configuration must be performed by the administrator of that service. See the documentation provided with your operating system for information about user groups.

When you run the process control daemon, identify the administrative group with the `-admingroup` command line option. If you do not specify a group name, process control checks to see if the user is a member of the default group `ncoadmin`.

Configuring Process Control Agents

For each process control agent you must:

- Add the server name in the Server Editor
- Start the process control agent

Adding Server Names in the Server Editor

You must assign a unique server name to each process control agent for each host machine. For each process control agent:

1. Add an entry for each process control agent using the Server Editor, as described in the Netcool/OMNIbus Installation and Deployment Guide.

The name must end with `_PA` to identify the server as a process control agent in the Server Editor. For example, if you are installing the process control agent on a host named `sfosys1`, the process control agent could be named `SFOSYS1_PA`. By default, the first process control agent installed in a configuration is named `NCO_PA`.

2. Distribute updated interfaces files to all host machines in the configuration. See the Netcool/OMNIbus Installation and Deployment Guide for information about how to generate interfaces files for all platforms.

This enables all process control agents on all host machines to communicate with each other.

Manually Starting Process Control Agents

To manually start a process control agent, enter the following command on the host's command line:

```
$OMNIHOME/bin/nco_pad -name process_agent
```

where *process_agent* is the name of the process control agent.

Automatically Starting Process Control Agents on Reboot

On UNIX platforms, startup scripts are available to enable the process agent to start automatically when the system boots. They are located in the following directory:

```
$OMNIHOME/install/startup
```

This directory contains one of the following install scripts, depending on the operating system platform:

```
aix5install  
hpux11install  
solaris2install  
linux2x86install
```

Installing the Process Agent Startup Scripts

To install the process agent startup scripts:

1. Run the installation script as the `root` user. For example, to install the scripts on Solaris platforms, run `solaris2install`.

The following is displayed:

```
Name of the Process Agent Daemon [NCO_PA]
```

2. Press **Enter** to accept the default process agent server name `NCO_PA` or enter another server name.

The following is displayed:

```
Should pa_name run in secure mode (y/n)? [y]
```

3. Press **Enter** to include the `-secure` command line option when starting the process agent. Secure mode controls the authentication of connection requests with a user name and password.

The following is displayed:

```
Enter value for environment variable NETCOOL_LICENSE_FILE [27000@localhost]:
```

4. Press **Enter** to accept the default value for licensing environment variable or enter another value.

Each install script copies or links the required configuration files into the system startup directory. On some systems (for example Solaris and HP-UX), the ability to stop the processes at system shutdown is also provided.

You can modify the startup scripts before you install them, if required. Refer to the specific operating system documentation for information about modifying startup scripts.

4.4 Defining Processes, Services, and Hosts

To run under process control, processes, services, and hosts must be added to the following configuration file:

```
$OMNIHOME/etc/nco_pa.conf
```

The process control agent reads this file when it is started to establish configuration settings. The file is made up of definitions, each of which contains attributes and associated values, for each process, service, and host.

Edit this file directly to add or modify definitions. Maintain the configuration files on all of your hosts to ensure that the host configuration information stays synchronized across all of the agents in the configuration.



Note: To prevent unauthorized users from gaining access, operating system security must be set appropriately for files such as configuration files that may contain user names and passwords.

Defining Processes

You must define the list of processes in the configuration file. An example process definition in the `nco_pa.conf` configuration file is shown below:

```
nco_process 'ObjectServer'
{
    Command '$OMNIHOME/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA' run as 0
    Host = 'sfosys1'
    Managed = True
    RestartMsg = '${NAME} running as ${EUID} has been restored on ${HOST}.'
    AlertMsg = '${NAME} running as ${EUID} has died on ${HOST}.'
    RetryCount = 0
    ProcessType = PaPA_AWARE
}
```

Process Definition Description

The following table contains a description of the process definition information contained in the configuration file.

Table 76: Process Definition Description (1 of 2)

Configuration Information	Description
nco_process 'ObjectServer'	Defines the name of the process. This example is for an ObjectServer. Note: Process names must be unique for this process control agent.

Table 76: Process Definition Description (2 of 2)

Configuration Information	Description
Command	<p>The command string that starts the process as it would be entered on the command line. Use the full path for the command. For example, to configure an ObjectServer named NCOMS, enter:</p> <pre>'\$OMNIHOME/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA' run as 0</pre> <p>or:</p> <pre>'\$OMNIHOME/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA' run as 'root'</pre> <p>The <code>run as</code> option instructs the host machine to run the ObjectServer as the specified user. Enter either the user ID (typically 0) or the user name in quotes (typically <code>root</code>). When a user name is entered, the process agent looks up the user ID to use.</p> <p><i>If the process control agent is not running as root, this option is ignored, and the process is run as the user who is running the process control agent.</i></p> <p>For information about the ObjectServer command line options, see <i>Specifying ObjectServer Command Line Options</i> on page 19.</p>
Host	The name of the host on which the process should be executed. Process control automatically resolves the name of the process control agent when required.
Managed	Can have the value <code>True</code> (the process is restarted automatically if it exits) or <code>False</code> (the process is not restarted automatically if it exits).
RestartMsg	Contains the message to be sent to syslog if the process is restarted. For example, <code>The ObjectServer has been restarted.</code>
AlertMsg	Contains the message to be sent to syslog if the process exits. For example, <code>The ObjectServer has gone down.</code>
RetryCount	Specifies the number of restart attempts to be made if the process exits in the time specified by the process agent daemon <code>-retrytime</code> command line option. If set to 0, there is no limit to the number of restart attempts. The default is 0.
ProcessType	Can have the value <code>PaPA_AWARE</code> for PA aware processes and <code>PaNOT_PA_AWARE</code> for processes that are not PA aware. See <i>Processes</i> on page 226 for additional information.

Expansion Keywords

You can include expansion keywords in the `RestartMsg` and `AlertMsg` entries in the configuration file. Expansion keywords act as variables and contain information about the process that has restarted. The expansion keywords are shown in Table 77.

Table 77: Expansion Keywords

Expansion Keyword	Description
<code>\${NAME}</code>	The name of the process.
<code>\${HOST}</code>	The name of the host running the process.
<code>\${EUID}</code>	The effective user ID under which the process is running.
<code>\${COMMAND}</code>	The command that defines the process.

Alert and Restart Syslog Messages

When an alert or restart message is generated by `nco_pad`, it is passed to the syslog system. Netcool/OMNIBus has a Syslog Probe that can monitor these messages and convert them into ObjectServer alerts. For more information about the Syslog Probe, refer to the probe documentation available on the Micromuse Support Site.

The alert and restore messages are sent to syslog as warnings. The message is formatted as:

```
HOSTNAME : ALERT_OR_RESTART_MSG : MSG
```

The *HOSTNAME* is the name of the host that has reported the problem. *ALERT_OR_RESTORE_MSG* describes the type of message. *MSG* is the text defined in the configuration file for that process or host.

Defining Services

You can define services to group together related processes. The processes must already be defined in the list of processes (see *Defining Processes* on page 231).

An example service definition in the `nco_pa.conf` configuration file is shown below.

```
nco_service 'Omnibus'
{
    ServiceType      =      Master
    ServiceStart     =      Non-Auto
    process 'ObjectServer' NONE
    process 'Proxy' 'ObjectServer'
    process 'Probe' 'Proxy'
    process 'Probe-1' 'ObjectServer'
```

```

    process 'Sleep' 5
}

```

Service Definition Description

The following table contains a description of the service definition information contained in the configuration file.

Table 78: Service Definition Description

Configuration Information	Description
<code>nco_service 'Omnibus'</code>	Defines the name of the service (for example, Omnibus). Note: Each service name must be unique for this process control agent.
<code>ServiceType</code>	Defines whether this service should be started before all other services and handled as the master service upon which other services depend. This can be set as either <code>Master</code> or <code>Non-Master</code> .
<code>ServiceStart</code>	This can be set to <code>Auto</code> to start the service as soon as <code>nco_pa</code> has started and <code>Non-Auto</code> if the service must be started manually with the <code>nco_pa_start</code> command.
<code>process</code>	Defines a process that must be run as part of the service. You can indicate process dependencies so that a process cannot start before another is already running. See <i>Specifying Process Dependencies</i> on page 234.

Specifying Process Dependencies

The `process` attribute allows you to define the processes that should be run as part of the service. You can add dependencies on each of the processes in the service. The format of the `process` attribute is as follows:

```
process 'processname' dependency
```

In this attribute, *processname* is the name of the process defined in the list of processes and *dependency* can be a numeric value, a string value, or `NONE`.

If *dependency* is a number it indicates a time dependency, in seconds, for starting the dependent process. For example, if you enter 5, the process starts five seconds after the service has started.

If *dependency* is a string it indicates another PA aware process in the same service. For information about process types, see *Defining Processes* on page 231.

In the example Omnibus service on page 233, the `ObjectServer` process starts first because it has no dependencies. Five seconds after the `ObjectServer` starts, the `Sleep` process starts. Once the `ObjectServer` is running successfully, `Proxy` and `Probe-1` start. When the proxy server is running, the `Probe` process starts.

The *dependency* type NONE specifies no dependency.

Defining Secure Hosts

You can specify that only certain hosts can connect to process agents by adding a security definition to the `nco_pa.conf` configuration file between the service and routing definitions. Routing definitions are described on page 236.

If you do not create a security definition, any process can connect from any host.

You can create a security definition with no hosts specified, as follows:

```
nco_security
{
}
```

When no hosts are specified, only processes running on the current host or any host listed in the routing definition can connect.

Processes running on hosts not listed in the routing definition can only connect if their host is listed in the security definition.

The process agent compares the IP address of the incoming connection with the IP address of each entry in the security and routing definitions. It also checks the IP address of the local host. Only the main address of the host running the process agent daemon is automatically added to the security definition. You must add the loopback address (127 . 0 . 0 . 1) and secondary interfaces, if required.



Note: When a process connecting to the process agent is run on a host with multiple interfaces, you should add the address of the interface closest to the process agent daemon. This does not need to be the main address of that host, nor, in the case of the ObjectServer (`nco_objserv`) or the process agent daemon (`nco_pad`), does it need to be the address in the Server Editor (`nco_xigen`).

You can specify the following types of entries in the security definition:

- A host name, in which case a lookup is performed to find the corresponding IP address
- A full IP address in dotted decimal format

An IP address in dotted decimal format can contain the following wildcards:

- ? matches one character
- * matches any number of characters

The following example security definition allows connections from processes on hosts `alpha`, `192.9.200.34`, and any host on the subnet `193.42.52.0`.

```
nco_security
{
    host 'alpha'
    host '192.9.200.34'
    host '193.42.52.*'
}
```

Defining Routing Hosts, User Names, and Passwords

To specify the hosts that are participating in the process control system, you must define the process agent host names.

Each `host` field defines the name of the host (for example, `sfosys1`) and the name of the process agent to be used in the process control system (for example, `SFOSYS1_PA`).

When using process control in secure mode, the routing entry must also have a user name and password.

Routing Definition Example

An example routing definition in the `nco_pa.conf` configuration file is shown below:

```
nco_routing
{
    host 'sfosys1' 'SFOSYS1_PA' 'username' 'password'
    host 'sfosys2' 'SFOSYS2_PA' 'username' 'password'
}
```

If the process agent is using UNIX authentication (the default), the `username` must be an operating system user that is a member of the `ncoadmin` group, as described in *Creating User Groups* on page 228. A process agent daemon that is running in secure mode must be run by the `root` user.



Note: If you run the process agent in secure mode, user names and passwords are required in the routing entries. If you are not running the process agent in secure mode, user names and passwords are optional.

Password Encryption Example

You can use the `nco_pa_crypt` utility to encrypt plain text login passwords stored in the configuration file. Passwords encrypted using `nco_pa_crypt` are decrypted by the remote process control agent.

To encrypt a plain text password:

1. Enter the following:

```
$OMNIHOME/bin/nco_pa_crypt password
```

where *password* is the unencrypted form of the password. The `nco_pa_crypt` utility displays an encrypted version of the password.

2. Copy the encrypted password into the appropriate routing entry.

If the user name is specified without a password, the system prompts for a password. If the password is specified without a user name, the name of the user entering the command is used.



Note: To prevent unauthorized users from gaining access, operating system security must be set appropriately for files that contain user names and passwords.

You can also specify the user name and password with the `-username` and `-password` command line options. This overrides any entries in the `nco_pa.conf` configuration file.



Note: Even if the password is specified on the command line, it does not appear in `ps` command output.

4.5 Example Process Agent Configuration File

This section contains an example process agent configuration file.

```
#####
#NCO_PA3
#
# Process Agent Daemon Configuration File 1.1
#
# Ident: $Id: nco_pa.conf 1.3 2002/05/21 15:28:10
#
#
# List of processes
#
nco_process 'FlexLM'
{
Command '/opt/netcool/common/license/bin/nc_start_license' run as 0 Host='objser1'

    Managed      =      True
    RestartMsg    =      '${NAME} running as ${EUID} has been restored on ${HOST}.'
    AlertMsg      =      '${NAME} running as ${EUID} has died on ${HOST}.'
    RetryCount    =      0
    ProcessType   =      PaNOT_PA_AWARE
}
nco_process 'SFOSYS_ObjectServer'
{
    Command '$OMNIHOME/bin/nco_objserv -name NETOPS1 -pa OBJSER1_PA' run as 0
    Host      =      'objser1'
    Managed    =      True
    RestartMsg =      '${NAME} running as ${EUID} has been restored on ${HOST}.'
    AlertMsg   =      '${NAME} running as ${EUID} has died on ${HOST}.'
    RetryCount =      0
    ProcessType =      PaPA_AWARE
}
nco_process 'Syslog_Probe'
{
    Command '$OMNIHOME/probes/nco_p_syslog' run as 0
    Host      =      'objser1'
    Managed    =      True
    RestartMsg =      '${NAME} running as ${EUID} has been restored on ${HOST}.'
    AlertMsg   =      '${NAME} running as ${EUID} has died on ${HOST}.'
    RetryCount =      0
    ProcessType =      PaPA_AWARE
}
nco_process 'Mttrapd_Probe'
{
    Command '$OMNIHOME/probes/nco_p_mttrapd' run as 0
    Host      =      'objser1'
    Managed    =      True
    RestartMsg =      '${NAME} running as ${EUID} has been restored on ${HOST}.'
```

```
AlertMsg      =      '${NAME} running as ${EUID} has died on ${HOST}.'
RetryCount    =      0
ProcessType   =      PaPA_AWARE
}
#
# List of Services
#
nco_service 'Core'
{
    ServiceType      =      Master
    ServiceStart     =      Auto
    process 'FlexLM' NONE
# ObjectServer started after 10 second delay to allow Licence Manager to
complete startup
    process 'NETOPS1_ObjectServer' 10
# Trapd probe and then Syslog probe started after another 10 seconds
    process 'Mttrapd_Probe' 10
    process 'Syslog_Probe' NONE
}
#
# ROUTING TABLE
#
# 'user' -      (optional) only required for secure mode PAD on target host
#              'user' must be member of UNIX group 'ncoadmin'
# 'password' - (optional) only required for secure mode PAD on target host
#              use nco_pa_crypt to encrypt.
nco_routing
{
    host 'objser1' 'OBJSER1_PA'
}
=====
```

4.6 Process Control Agent Daemon Command Line Options

This section describes the command line options for the process control agent daemon, `nco_pa.d`.

Table 79: Process Control Agent Daemon Command Line Options (1 of 4)

Option	Parameter	Description
<code>-admingroup</code>	<i>string</i>	Specifies the name of the UNIX user group that has administrator privileges. Members of this group can access the process control system. The default group name is <code>ncoadmin</code> .
<code>-apicheck</code>	N/A	If specified, Sybase API checking is enabled.
<code>-authenticate</code>	<i>string</i>	<p>Specifies the authentication mode to use to verify the credentials of a user or remote process control agent daemon. The options are <code>UNIX</code>, <code>PAM</code>, <code>KERBEROS</code>, and <code>HPTCB</code>.</p> <p>The default authentication mode is <code>UNIX</code>, which means that the Posix <code>getpwnam</code> or <code>getspnam</code> function is used to verify user credentials on UNIX platforms. Depending on system setup, passwords are verified using the <code>/etc/password</code> file, <code>/etc/shadow</code> shadow password file, <code>NIS</code>, or <code>NIS+</code>.</p> <p>If <code>PAM</code> is specified as the authentication mode, Pluggable Authentication Modules are used to verify user credentials. The service name used by the gateway when the PAM interface is initialized is <code>netcool</code>. PAM authentication is available on Linux, Solaris, and HP-UX 11 platforms only.</p> <p>If <code>KERBEROS</code> is specified, Kerberos IV authentication is used to verify user credentials. This is only available on Solaris systems with a Kerberos IV authentication server installed.</p> <p>If <code>HPTCB</code> is specified as the authentication mode, the HP-UX password protection system is used. This is only available on HP trusted (secure) systems.</p>
<code>-configfile</code>	<i>string</i>	Use this file name, relative to <code>\$OMNIHOME</code> , as the configuration file rather than the default file <code>\$OMNIHOME/etc/nco_pa.conf</code> .
<code>-debug</code>	<i>integer</i>	Enables debugging. The <i>integer</i> specifies the amount of debug information written to the log. Available levels are 1 (Debug), 2 (Information), 3 (Warning), 4 (Error), and 5 (Fatal). The default is 3.
<code>-DNS</code>	<i>string</i>	Specifies a value to override the host name in DNS environments. This must be the same as the entry in the configuration file.
<code>-help</code>	N/A	Displays help information about the process control agent and exits.
<code>-killprocessgroup</code>	N/A	If specified, when the process agent daemon stops a process, it also sends a signal to kill any processes in the same operating system process group.

Table 79: Process Control Agent Daemon Command Line Options (2 of 4)

Option	Parameter	Description
-logfile	<i>string</i>	Specifies an alternate log file. The default log file is: <code>\$OMNIHOME/log/pa_name.log</code> where <i>pa_name</i> is the name of the process control agent specified with <code>-name</code> .
-logsize	<i>integer</i>	Maximum log file size in KBytes. The default is 1024 KBytes, and the minimum size is 16 KBytes.
-msgpoolsize	<i>integer</i>	Specifies the number of messages available to the process control agent.
-name	<i>string</i>	Specifies the name of the server for this process control agent. If not specified, the default process control agent name is <code>NCO_PA</code> .
-newlog	N/A	Starts a new log file. Without this option, process control appends to the end of any existing log file. With this option, the log is cleared first.
-noauto	N/A	If specified, the process control agent does not start any services automatically, even if they are set to start automatically in the <code>nco_pa.conf</code> file.
-noconfig	N/A	If specified, the process control agent does not read the <code>nco_pa.conf</code> configuration file. This forces process control to start with no configuration information.
-nodaemon	N/A	By default, process control forks into the background to run as a daemon process. When <code>-nodaemon</code> is specified, the process runs in the foreground.
-password	<i>string</i>	Specifies the password used to log into a remote process control agent. If the <code>-user</code> option is not also specified, the user name used to make the connection is the user executing the command.
-pidfile	<i>string</i>	Specifies the path, relative to <code>\$OMNIHOME</code> , to the file in which the process control daemon PID is stored. Each process agent daemon must have its own PID file. The default is <code>\$OMNIHOME/var/nco_pa.pid</code> . This makes it possible to run more than one process agent daemon on the same machine.
-pidmsgpool	<i>integer</i>	Specifies the size of the signal-handling message pool.
-redirectfile	<i>string</i>	Specifies a file to which the <code>stderr</code> and <code>stdout</code> messages of processes started by the process agent are directed.

Table 79: Process Control Agent Daemon Command Line Options (3 of 4)

Option	Parameter	Description
<code>-retrytime</code>	<i>integer</i>	<p>Specifies the number of seconds that a process started by process control must run to be considered a successful start. The default <code>retrytime</code> is 5.</p> <p>The process control agent attempts to restart a process if the process exits. If the process exits after <code>retrytime</code> seconds, the process agent attempts to restart the process immediately. If the process exits before <code>retrytime</code> seconds, the process agent attempts to restart the process at the exponential rate of 2, 4, 8, 16, 32, ..., 256 seconds. The process agent resets the timing interval after eight attempts to start the process.</p> <p>If the process fails to run for more than <code>retrytime</code> seconds, the <code>RetryCount</code> (specified in the process definition) for that process is also decremented. If the process runs successfully for at least <code>retrytime</code> seconds, the <code>RetryCount</code> is set back to its original value. If the <code>RetryCount</code> is 0, there is no limit to the number of restart attempts.</p>
<code>-roguetimeout</code>	<i>integer</i>	Specifies the time in seconds to wait for the process to shut down. The default is 30 and the minimum is 5 seconds.
<code>-secure</code>	N/A	<p>All clients need to authenticate themselves with a valid user name and password, specified with the <code>-user</code> and <code>-password</code> command line options. If specified, when the process control agent connects to another process control agent, login information is automatically encrypted in transmission.</p> <p><i>Do not use this option when the process control agent is connecting to a process control agent prior to Netcool/OMNIBus version 3.5.</i></p>
<code>-sslcertificate</code>	<i>string</i>	<p>Specifies the path to the SSL certificate. The default is <code>\$OMNIHOME/etc/servername.crt</code>.</p> <p>For more information on setting up a system using SSL communications, see the Netcool/OMNIBus Installation and Deployment Guide.</p>
<code>-sslpassword</code>	<i>string</i>	<p>Specifies the SSL private key password. The default is ' '.</p> <p>For more information on setting up a system using SSL communications, see the Netcool/OMNIBus Installation and Deployment Guide.</p>
<code>-stacksize</code>	<i>integer</i>	Specifies the size of the thread stack.
<code>-ticketdir</code>	<i>string</i>	Directory for Kerberos tickets if <code>-authenticate</code> is set to <code>KERBEROS</code> .
<code>-traceevtq</code>	N/A	Enables tracing of event queue activity.
<code>-tracemsgq</code>	N/A	Enables tracing of message queue activity.
<code>-tracemtx</code>	N/A	Enables the tracing of mutex locks.

Table 79: Process Control Agent Daemon Command Line Options (4 of 4)

Option	Parameter	Description
-tracenet	N/A	Enables net library tracing.
-user	<i>string</i>	Specifies the user name used to log into another process control agent. This must be specified if connecting to a process control agent running in secure mode (using the <code>-secure</code> option).
-version	N/A	Displays version information about the process control agent and exits.
-walkhosttab	N/A	Forces the process control agent to search the hosts table to verify all aliases.



Note: A new instance of the process control agent cannot manage processes that were started by another instance and are still running. When the process control agent is stopped and restarted, it has no knowledge of such processes, and therefore starts new instances of them. The previous instances are left running.

4.7 Process Control Management

The process control system provides some utilities to manage and change the Netcool/OMNIBus configuration. This section describes the following management utilities:

- `nco_pa_status`
- `nco_pa_start`
- `nco_pa_stop`
- `nco_pa_shutdown`
- `nco_pa_addentry`



Note: Each utility prompts for your password.

Displaying Service Status - `nco_pa_status`

The `nco_pa_status` utility retrieves the status of services in the configuration. For each service, `nco_pa_status` returns a list of defined processes, the status of each process, and the UNIX process identifier. To display the service status, enter the command:

```
$OMNIHOME/bin/nco_pa_status -server string
```

where *string* is the process agent name. The following is example output:

```
-----
Service Name      Process Name      Hostname  User   Status  PID
-----
Master Service   ObjectServer     SFOSYS1  root   RUNNING 16751
                  Proxy            SFOSYS1  root   RUNNING 16752
                  Sleep            SFOSYS1  root   RUNNING 16753
                  Probe            SFOSYS1  root   RUNNING 16754
-----
```

Table 80 describes each of the status levels:

Table 80: Service Status Descriptions (1 of 2)

Status Level	Description
RUNNING	The process is running.
STARTING	A start request has been issued.

Table 80: Service Status Descriptions (2 of 2)

Status Level	Description
PENDING	The process is waiting for a dependency to start. This status can also indicate that the process has failed to start properly (whether or not it has any dependencies).
WAITING	The process is waiting for a time dependency to complete.
DEAD	The process is not running.
ERROR	It was not possible to retrieve a status from the process agent.

If a process agent is instructed to run a process by a process agent running on a separate machine, the remote process agent does not retain a record of the process. If the remote process agent stops, the process continues to run. When the remote process agent restarts, it has no record of the process, and therefore the process status for this orphan process is listed as DEAD.

You can manually restart the process using the `nco_pa_start` utility.

Command Line Options

Command line options for `nco_pa_start` are described in Table 81.

Table 81: `nco_pa_status` Command Line Options

Option	Parameter	Description
<code>-help</code>	N/A	Displays help on the command line options and exits.
<code>-nosecure</code>	N/A	You must use this option to connect to process control agents for releases prior to Netcool/OMNIbus version 3.5.
<code>-password</code>	<i>string</i>	The password to use for the process agent.
<code>-server</code>	<i>string</i>	Name of process control agent to contact.
<code>-user</code>	<i>string</i>	The user name for the process agent. The default is the user running the command.
<code>-version</code>	N/A	Displays software version information and exits.

Starting a Service or Process - `nco_pa_start`

The `nco_pa_start` utility starts a service or process at any location in the process control configuration. You can only specify a single service or process. If the service or process has already been started, the command is ignored.

Command Line Options

Command line options for `nco_pa_start` are described in Table 82.

Table 82: `nco_pa_start` Command Line Options

Option	Parameter	Description
<code>-help</code>	N/A	Displays help about the command line options and exits.
<code>-nosecure</code>	N/A	You must use this option to connect to process control agents for releases prior to Netcool/OMNIBus version 3.5.
<code>-password</code>	<i>string</i>	The password to use for the process agent.
<code>-process</code>	<i>string</i>	Name of the process to start.
<code>-server</code>	<i>string</i>	Name of process control agent to contact.
<code>-service</code>	<i>string</i>	Name of the service to start.
<code>-user</code>	<i>string</i>	The user name for the process agent. The default is the user running the command.
<code>-version</code>	N/A	Displays software version information and exits.

Stopping a Service or Process - `nco_pa_stop`

The `nco_pa_stop` utility stops a service or process at any location in the process control configuration. You can only specify a single service or process. If the service or process has already been stopped, the command is ignored.

Command Line Options

Command line options for `nco_pa_stop` are described in Table 83.

Table 83: `nco_pa_stop` Command Line Options (1 of 2)

Option	Parameter	Description
<code>-force</code>	N/A	If specified, no warning is output if the process or service is not running.
<code>-help</code>	N/A	Displays help about the command line options and exits.
<code>-nosecure</code>	N/A	You must use this option to connect to process control agents for releases prior to Netcool/OMNIBus version 3.5.
<code>-password</code>	<i>string</i>	The password to use for the process agent.

Table 83: nco_pa_stop Command Line Options (2 of 2)

Option	Parameter	Description
-process	<i>string</i>	Name of the process to stop.
-server	<i>string</i>	Name of process control agent to contact.
-service	<i>string</i>	Name of the service to stop.
-user	<i>string</i>	The user name for the process agent. The default is the user running the command.
-version	N/A	Displays software version information and exits.

Shutting Down a Process Control Agent - nco_pa_shutdown

The `nco_pa_shutdown` utility shuts down a process control agent and optionally stops associated services and processes.

Command Line Options

Command line options for `nco_pa_shutdown` are described in Table 84.

Table 84: nco_pa_shutdown Command Line Options

Option	Parameter	Description
-help	N/A	Displays help about the command line options and exits.
-nosecure	N/A	You must use this option to connect to process control agents for releases prior to Netcool/OMNIbus version 3.5.
-option	<i>string</i>	Specifies how the shutdown is completed. Can be <code>STOP</code> to shut down all managed processes, or <code>LEAVE</code> to leave the managed processes running after the shutdown. If <code>-option</code> is not specified on the command line, the utility displays a menu with the shutdown options and prompts the user for the type of shutdown to perform.
-password	<i>string</i>	The password to use for the process agent.
-server	<i>string</i>	Name of process control agent to shut down.
-user	<i>string</i>	The user name for the process agent. The default is the user running the command.
-version	N/A	Displays software version information and exits.

Adding a New Service or Process to a Running PAD - nco_pa_addentry

The `nco_pa_addentry` utility enables you to add a new service or process to a running process control agent.



Note: The new service or process is not added to the process agent configuration file.

Command Line Options

Command line options for `nco_pa_addentry` are described in Table 85.

Table 85: `nco_pa_addentry` Command Line Options (1 of 2)

Option	Parameter	Description
<code>-alert_msg</code>	<i>string</i>	Specifies the message to send to the syslog if the process exits.
<code>-auto</code> <code>-nonauto</code>	N/A	If <code>auto</code> , the service or process is started as soon as the process control agent is started. By default, the service must be started manually with the <code>nco_pa_start</code> command (see page 245).
<code>-command</code>	<i>string</i>	Specifies the process command line. For example: <code>\$OMNIHOME/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA</code>
<code>-delay</code>	<i>string</i>	Specifies the time delay in seconds before the specified process is started.
<code>-depend</code>	<i>string</i>	Specifies a process on which the specified process depends.
<code>-help</code>	N/A	Displays help on the command line options and exits.
<code>-host</code>	<i>string</i>	Specifies the host on which to run the process.
<code>-managed</code> <code>-unmanaged</code>	N/A	If <code>managed</code> (the default), the process is restarted automatically if it exits.
<code>-master</code> <code>-nonmaster</code>	N/A	If <code>master</code> (the default), the service type is set to <code>master</code> .
<code>-nosecure</code>	N/A	You must use this option to connect to process control agents for releases prior to Netcool/OMNIBus version 3.5.
<code>-pa_aware</code> <code>-not_pa_aware</code>	N/A	If <code>pa_aware</code> , the <code>ProcessType</code> is set to <code>PaPA_AWARE</code> . By default, the process is not PA aware.
<code>-parentservice</code>	<i>string</i>	Specifies the service to which to add the process.

Table 85: nco_pa_addentry Command Line Options (2 of 2)

Option	Parameter	Description
-password	<i>string</i>	Specifies the password to use when connecting to the process control agent.
-process	<i>string</i>	Specifies the name of the process to add.
-restart_msg	<i>string</i>	Specifies the message to send to syslog if the process is restarted.
-retrycount	<i>integer</i>	Specifies the number of restart attempts to be made if the process exits in the time specified by the process agent daemon <code>-retrytime</code> command line option. If set to 0, there is no limit to the number of restart attempts. The default is 0.
-runas	<i>integer</i>	Specifies the user ID to run the process as.
-server	<i>string</i>	Specifies the name of the process control agent. The default is NCO_PA.
-service	<i>string</i>	Specifies the name of the service to add.
-user	<i>string</i>	Specifies the user name to use when connecting to the process control agent. The default is the user running the command.
-version	N/A	Displays software version information and exits.

Appendix A: ObjectServer Tables

This appendix contains ObjectServer database table information. It contains the following sections:

- *Alerts Tables* on page 252
- *Service Tables* on page 261
- *System Catalog Tables* on page 262
- *Statistics Tables* on page 274
- *Client Tool Support Tables* on page 278
- *Desktop Tools Tables* on page 283
- *Desktop ObjectServer Tables* on page 288
- *Security Tables for Backward Compatibility* on page 290

A.1 Alerts Tables

Alert information is forwarded to the ObjectServer from external programs such as probes, monitors, and gateways, stored and managed in database tables, and displayed in the event list. For information about how alerts are processed, see *Alert Processing in the ObjectServer* on page 124.

alerts.status Table

The `alerts.status` table contains status information about problems that have been detected by probes.

Table A1: Columns in the alerts.status Table (1 of 7)

Column Name	Data Type	Mandatory	Description
Identifier	varchar (255)	Yes	Controls ObjectServer deduplication. Deduplication is described in <i>Introduction to Deduplication</i> on page 124.
Serial	incr	Yes	The Netcool/OMNIBus serial number for the row.
Node	varchar (64)	Yes	Identifies the managed entity from which the alarm originated. This could be a host or device name, service name, or other entity. For IP network devices or hosts, the <code>Node</code> column contains the resolved name of the device or host. In cases where the name cannot be resolved, the <code>Node</code> column should contain the IP address of the host or device.
NodeAlias	varchar (64)	No	The alias for the node. For network devices or hosts, this should be the logical (layer-3) address of the entity. For IP devices or hosts, this should be the IP address.
Manager	varchar (64)	Yes	The descriptive name of the probe that collected and forwarded the alarm to the ObjectServer. This can also be used to indicate the host on which the probe is running.
Agent	varchar (64)	No	The descriptive name of the sub-manager that generated the alert.
AlertGroup	varchar (64)	No	The descriptive name of the type of failure indicated by the alert (for example, <code>Interface Status</code> or <code>CPU Utilization</code>).

Table A1: Columns in the alerts.status Table (2 of 7)

Column Name	Data Type	Mandatory	Description
AlertKey	varchar (255)	Yes	The descriptive key that indicates the managed object instance referenced by the alert (for example, the disk partition indicated by a file system full alert or the switch port indicated by a utilization alert).
Severity	integer	Yes	Indicates the alert severity level, which provides an indication of how the perceived capability of the managed object has been affected. The color of the alert in the event list is controlled by the severity value: 0 - Clear 1 - Indeterminate 2 - Warning 3 - Minor 4 - Major 5 - Critical
Summary	varchar (255)	Yes	The text summary of the cause of the alert.
StateChange	time	Yes	An automatically maintained ObjectServer timestamp of the last insert or update of the alert from any source.
FirstOccurrence	time	Yes	The time in seconds (from midnight Jan 1, 1970) when this alert was created or when polling started at the probe.
LastOccurrence	time	Yes	The time when this alert was last updated at the probe.
InternalLast	time	Yes	The time when this alert was last updated at the ObjectServer.
Poll	integer	No	The frequency of polling for this alert in seconds.

Table A1: Columns in the alerts.status Table (3 of 7)

Column Name	Data Type	Mandatory	Description
Type	integer	No	The type of alert: 0 - Type not set 1 - Problem 2 - Resolution 3 - Netcool/Visionary problem 4 - Netcool/Visionary resolution 7 - Netcool/ISMs new alarm 8 - Netcool/ISMs old alarm 11 - More Severe 12 - Less Severe 13 - Information
Tally	integer	Yes	Automatically maintained count of the number of inserts and updates of the alert from any source. This count is affected by deduplication, described in <i>Introduction to Deduplication</i> on page 124.
Class	integer	Yes	The alert class used to identify the probe or vendor from which the alert was generated. Controls the applicability of context-sensitive event list tools.
Grade	integer	No	Indicates the state of escalation for the alert: 0 - Not Escalated 1 - Escalated
Location	varchar (64)	No	Indicates the physical location of the device, host, or service for which the alert was generated.
OwnerUID	integer	Yes	The user identifier of the user who is assigned to handle this alert. The default is 65534, which is the identifier for the nobody user.
OwnerGID	integer	No	The group identifier of the group that is assigned to handle this alert. The default is 0, which is the identifier for the public group.

Table A1: Columns in the alerts.status Table (4 of 7)

Column Name	Data Type	Mandatory	Description
Acknowledged	integer	Yes	Indicates whether the alert has been acknowledged: 0 - No 1 - Yes Alerts can be acknowledged manually by a network operator or automatically by a correlation or workflow process.
Flash	integer	No	Enables the option to make the event list flash.
EventID	varchar(64)	No	The event ID (for example, SNMPTRAP-link down). Multiple events can have the same event ID. This is populated by the probe rules file and used by Netcool/Precision.
ExpireTime	integer	Yes	The number of seconds until the alert is cleared automatically. Used by the Expire automation.
ProcessReq	integer	No	Indicates whether the alert should be processed by Netcool/Precision. This is populated by the probe rules file and used by Netcool/Precision.
SuppressEscl	integer	Yes	Used to suppress or escalate the alert: 0 - Normal 1 - Escalated 2 - Escalated-Level 2 3 - Escalated-Level 3 4 - Suppressed 5 - Hidden 6 - Maintenance The suppression level is manually selected by operators from the event list.
Customer	varchar(64)	No	The name of the customer affected by this alert.
Service	varchar(64)	No	The name of the service affected by this alert.
PhysicalSlot	integer	No	The slot number indicated by the alert.

Table A1: Columns in the alerts.status Table (5 of 7)

Column Name	Data Type	Mandatory	Description
PhysicalPort	integer	No	The port number indicated by the alert.
PhysicalCard	varchar (64)	No	The card name or description indicated by the alert.
TaskList	integer	Yes	Indicates whether a user has added the alert to the Task List: 0 - No 1 - Yes Operators can add alerts to the Task List from the event list.
NmosSerial	varchar (64)	No	The serial number of a suppressed alert. Populated by Netcool/Precision.
NmosObjInst	integer	No	Populated by Netcool/Precision during alert processing.
NmosCauseType	integer	No	The alert state, populated by Netcool/Precision as an integer value: 0 - Unknown 1 - Root cause 2 - Symptom
LocalNodeAlias	varchar (64)	Yes	The alias of the network entity indicated by the alert. For network devices or hosts, this is the logical (layer-3) address of the entity, or another logical address that enables direct communication with the device. For use in managed object instance identification.
LocalPriObj	varchar (255)	No	The primary object referenced by the alert. For use in managed object instance identification.
LocalSecObj	varchar (255)	No	The secondary object referenced by the alert. For use in managed object instance identification.
LocalRootObj	varchar (255)	Yes	An object that is equivalent to the primary object referenced in the alarm. For use in managed object instance identification.
RemoteNodeAlias	varchar (64)	Yes	The network address of the remote network entity. For use in managed object instance identification.

Table A1: Columns in the alerts.status Table (6 of 7)

Column Name	Data Type	Mandatory	Description
RemotePriObj	varchar (255)	No	The primary object of a remote network entity referenced by an alarm. For use in managed object instance identification.
RemoteSecObj	varchar (255)	No	The secondary object of a remote network entity referenced by an alarm. For use in managed object instance identification.
RemoteRootObj	varchar (255)	Yes	An object that is equivalent to the remote entity's primary object referenced in the alarm. For use in managed object instance identification.
X733EventType	integer	No	Indicates the alert type: 0 - Not defined 1 - Communications 2 - Quality of Service 3 - Processing error 4 - Equipment 5 - Environmental 6 - Integrity violation 7 - Operational violation 8 - Physical violation 9 - Security service violation 10 - Time domain violation
X733ProbableCause	integer	No	Indicates the probable cause of the alert.
X733SpecificProb	varchar (64)	No	Identifies additional information for the probable cause of the alert. Used by probe rules files to specify a set of identifiers for use in managed object instance identification.
X733CorrNotif	varchar (255)	No	A listing of all notifications with which this notification is correlated.
ServerName	varchar (64)	Yes	The name of the originating ObjectServer. Used by gateways to control propagation of alerts between ObjectServers.

Table A1: Columns in the alerts.status Table (7 of 7)

Column Name	Data Type	Mandatory	Description
ServerSerial	integer	Yes	The serial number of the alert on the originating ObjectServer (if it did not originate on this ObjectServer). Used by gateways to control the propagation of alerts between ObjectServers.
URL	varchar(1024)	No	Optional URL which provides a link to additional information in the vendor's device or ENMS.
MasterSerial	integer	No	Identifies the master ObjectServer if this alert is being processed in a desktop ObjectServer environment. This column is added when you run <code>nco_dbinit</code> with the <code>-desktopserver</code> option. Note: <code>MasterSerial</code> must be the last column in the <code>alerts.status</code> table if you are using a desktop ObjectServer environment. For information about the desktop ObjectServer environment, see the Netcool/OMNIBus Installation and Deployment Guide.



Note: You can only display columns of type CHAR, VARCHAR, INCR, INTEGER, and TIME in the event list. Do not add columns of any other type to the `alerts.status` table.

alerts.details Table

The `alerts.details` table contains the detail attributes of the alerts in the system.

Table A2: Columns in the alerts.details Table (1 of 2)

Column Name	Data Type	Description
KeyField	varchar(255)	Internal sequencing string for uniqueness.
Identifier	varchar(255)	Identifier to relate details to entries in the <code>alerts.status</code> table.
AttrVal	integer	Boolean; when false (0), just the <code>Detail</code> column is valid. Otherwise, the <code>Name</code> and <code>Detail</code> columns are both valid.
Sequence	integer	Sequence number, used for ordering entries in the event list <i>Event Information</i> window.

Table A2: Columns in the alerts.details Table (2 of 2)

Column Name	Data Type	Description
Name	varchar (255)	Name of attribute stored in Detail column.
Detail	varchar (255)	Attribute value.

alerts.journal Table

The alerts.journal table provides a history of work performed on alerts.

Table A3: Columns in the alerts.journal Table (1 of 2)

Column Name	Data Type	Description
KeyField	varchar (255)	Primary key for table.
Serial	integer	Serial number of alert that this journal entry is related to.
UID	integer	User identifier of user who made this entry.
Chrono	time	Time and date that this entry was made.
Text1	varchar (255)	First block of text for journal entry.
Text2	varchar (255)	Second block of text for journal entry.
Text3	varchar (255)	Third block of text for journal entry.
Text4	varchar (255)	Fourth block of text for journal entry.
Text5	varchar (255)	Fifth block of text for journal entry.
Text6	varchar (255)	Sixth block of text for journal entry.
Text7	varchar (255)	Seventh block of text for journal entry.
Text8	varchar (255)	Eighth block of text for journal entry.
Text9	varchar (255)	Ninth block of text for journal entry.
Text10	varchar (255)	Tenth block of text for journal entry.

Table A3: Columns in the alerts:journal Table (2 of 2)

Column Name	Data Type	Description
Text11	varchar (255)	Eleventh block of text for journal entry.
Text12	varchar (255)	Twelfth block of text for journal entry.
Text13	varchar (255)	Thirteenth block of text for journal entry.
Text14	varchar (255)	Fourteenth block of text for journal entry.
Text15	varchar (255)	Fifteenth block of text for journal entry.
Text16	varchar (255)	Sixteenth block of text for journal entry.

A.2 Service Tables

The service table contains information about Netcool/ISMs.

service.status Table

The `service.status` table is used to control the additional features required to support Netcool/ISMs.

Table A4: Columns in the `service.status` Table

Column Name	Data Type	Description
Name	<code>varchar(255)</code>	Name of the service.
CurrentState	<code>integer</code>	Indicates the state of the service: 0 - Good 1 - Bad 2 - Marginal 3 - Unknown
StateChange	<code>time</code>	Indicates the last time the service state changed.
LastGoodAt	<code>time</code>	Indicates the last time the service was Good (0).
LastBadAt	<code>time</code>	Indicates the last time the service was Bad (1).
LastMarginalAt	<code>time</code>	Indicates the last time the service was Marginal (2).
LastReportAt	<code>time</code>	Time of the last service status report.

A.3 System Catalog Tables

The `catalog` database contains the system tables created and maintained by the ObjectServer. System tables contain metadata about ObjectServer objects. You can view the information in system tables using the `SELECT` and `DESCRIBE` commands, but you cannot modify these tables.

catalog.databases Table

The `catalog.databases` table stores information about each database, including the number of objects in the database and the type of database (system or user).

Table A5: Columns in the `catalog.databases` Table

Column Name	Data Type	Description
DatabaseName	<code>varchar(40)</code>	Name of the database.
NumTables	<code>unsigned</code>	Number of base tables and views in the database.
IsSystem	<code>boolean</code>	TRUE if this is a system database.

catalog.tables Table

The `catalog.tables` table stores information about all types of tables, including system and user tables, views, and transition tables.

Table A6: Columns in the `catalog.tables` Table (1 of 2)

Column Name	Data Type	Description
TableName	<code>varchar(40)</code>	Name of the table.
DatabaseName	<code>varchar(40)</code>	Name of the parent database.
Status	<code>integer</code>	Current status of the table: 0 - Valid 1 - Invalid 2 - Compile failed
NumDependents	<code>unsigned</code>	Number of dependants.
TableID	<code>integer</code>	Table identifier.

Table A6: Columns in the catalog.tables Table (2 of 2)

Column Name	Data Type	Description
TableKind	integer	Type of table: 0 - Base table 1 - Transition table 2 - View
StorageKind	integer	Type of storage: 1 - Persistent 2 - Virtual 4 - Transient

catalog.base_tables Table

The catalog.base_tables table stores information about user and system tables.

Table A7: Columns in the catalog.base_tables Table

Column Name	Data Type	Description
TableName	varchar(40)	Name of the table.
DatabaseName	varchar(40)	Name of the parent database.
StoreName	varchar(40)	Name of the parent store.
NumColumns	integer	Number of columns in the table.
CreationTime	time	Time the table was created.
StorageKind	integer	Type of storage: 1 - Persistent 2 - Virtual
IsSystem	boolean	TRUE if this is a system table.
IsNoModify	boolean	TRUE if the table cannot currently be modified.

catalog.views Table

The `catalog.views` table stores information about views. The `CreationText` column contains the SQL used to create the view.

Table A8: Columns in the `catalog.views` Table

Column Name	Data Type	Description
<code>ViewName</code>	<code>varchar(40)</code>	Name of the view.
<code>DatabaseName</code>	<code>varchar(40)</code>	Name of the parent database.
<code>CreationText</code>	<code>varchar(16384)</code>	The <code>CREATE VIEW</code> text used to create the view.
<code>StorageKind</code>	<code>integer</code>	Type of storage: 1 - Persistent 4 - Transient
<code>IsRecovered</code>	<code>boolean</code>	TRUE if this is a successfully recovered view after restart.
<code>IsDmlEnabled</code>	<code>boolean</code>	TRUE if all of the table's primary keys are in the view, and therefore DML actions can be performed on the view.
<code>IsAggregate</code>	<code>boolean</code>	TRUE if this is created from an aggregate <code>SELECT</code> statement.

catalog.files Table

The `catalog.files` table stores information about ObjectServer files, including the path to the operating system file associated with each ObjectServer file.

Table A9: Columns in the `catalog.files` Table (1 of 2)

Column Name	Data Type	Description
<code>FileName</code>	<code>varchar(40)</code>	Name of the ObjectServer file.
<code>FilePath</code>	<code>varchar(1028)</code>	Full path to the file on the file system.
<code>MaximumFiles</code>	<code>unsigned</code>	Maximum number of files.
<code>MaximumSize</code>	<code>unsigned</code>	Maximum file size.
<code>IsEnabled</code>	<code>boolean</code>	TRUE if information is being logged to this file.

Table A9: Columns in the catalog.files Table (2 of 2)

Column Name	Data Type	Description
Status	integer	Current status of the file: 0 - Valid 1 - Invalid 2 - Compile failed

catalog.restrictions Table

The `catalog.restrictions` table stores information about restriction filters. The `ConditionText` column contains the SQL condition.

Table A10: Columns in the catalog.restrictions Table

Column Name	Data Type	Description
RestrictionName	varchar(40)	Name of the restriction filter.
TableName	varchar(40)	Name of the table on which the restriction filter has been created.
DatabaseName	varchar(40)	Name of the parent database.
ConditionText	varchar(16384)	The condition text for the restriction filter.
CreationText	varchar(16384)	The <code>CREATE RESTRICTION</code> text used to create the restriction filter.

catalog.columns Table

The `catalog.columns` table stores information about table columns, including their data types.

Table A11: Columns in the catalog.columns Table (1 of 2)

Column Name	Data Type	Description
ColumnName	varchar(40)	Name of the column.
TableName	varchar(40)	Name of the table.
DatabaseName	varchar(40)	Name of the parent database.
DataType	integer	Column data type. The data types and their corresponding identifiers are listed in Table 53 on page 136.

Table A11: Columns in the catalog.columns Table (2 of 2)

Column Name	Data Type	Description
Length	unsigned	Number of characters in the column.
IsPrimaryKey	boolean	TRUE if the column is a primary key.
OrdinalPosition	unsigned	Position in the column list.
IsHidden	boolean	TRUE if this is a hidden column.
IsNoModify	boolean	TRUE if the column cannot currently be modified.
IsNoDefault	boolean	TRUE if the value of this column must be specified in the initial INSERT command.
IsSystem	boolean	TRUE if this is a system column.

catalog.primitive_signals Table

The `catalog.primitive_signals` table stores information about user and system signals.

Table A12: Columns in the catalog.primitive_signals Table

Column Name	Data Type	Description
SignalName	<code>varchar(40)</code>	Name of the signal.
IsSystem	boolean	TRUE if this is a system signal.
CommentBlock	<code>varchar(1024)</code>	Comment string specified in the CREATE SIGNAL command.

catalog.primitive_signal_parameters Table

The `catalog.primitive_signal_parameters` table stores information about the parameters to system and user defined signals.

Table A13: Columns in the catalog.primitive_signal_parameters Table (1 of 2)

Column Name	Data Type	Description
ParameterName	<code>varchar(40)</code>	Name of the parameter.
SignalName	<code>varchar(40)</code>	Name of signal with this parameter.

Table A13: Columns in the catalog.primitive_signal_parameters Table (2 of 2)

Column Name	Data Type	Description
DataType	integer	Parameter data type. The data types and their corresponding identifiers are listed in Table 53 on page 136.
Length	unsigned	Number of characters in the parameter.
OrdinalPosition	integer	Position in the parameter list.

catalog.trigger_groups Table

The `catalog.trigger_groups` table stores information about trigger groups, including whether or not the trigger group is enabled.

Table A14: Columns in the catalog.trigger_groups Table

Column Name	Data Type	Description
GroupName	varchar (40)	Name of the trigger group.
IsEnabled	boolean	TRUE if the trigger group is currently enabled.

catalog.triggers Table

The `catalog.triggers` table stores information about triggers, including the type of trigger, the trigger priority, and what trigger group it is in.

Table A15: Columns in the catalog.triggers Table (1 of 2)

Column Name	Type	Description
TriggerName	varchar (40)	Name of the trigger.
GroupName	varchar (40)	Trigger group name.
TriggerKind	integer	Type of trigger: 0 - Database 1 - Signal 2 - Temporal
DebugEnabled	boolean	TRUE if debugging is enabled for the trigger.
IsEnabled	boolean	TRUE if the trigger is enabled.

Table A15: Columns in the catalog.triggers Table (2 of 2)

Column Name	Type	Description
TriggerPriority	integer	Trigger priority: 1 is the highest, 20 is the lowest priority.
CommentBlock	varchar(1024)	Comment string specified in the CREATE TRIGGER command.
EvaluateBlock	varchar(2048)	Evaluation clause specified in the CREATE TRIGGER command.
BindName	varchar(40)	Bind name specified in the evaluation clause of the CREATE TRIGGER command.
ConditionBlock	varchar(1024)	When condition specified in the CREATE TRIGGER command.
DeclareBlock	varchar(1024)	Variable declaration specified in the CREATE TRIGGER command.
CodeBlock	varchar(8192)	The body of the trigger.

catalog.database_triggers Table

The catalog.database_triggers table stores information about database triggers, including the type of database operation that causes the trigger to fire.

Table A16: Columns in the catalog.database_triggers Table (1 of 2)

Column Name	Data Type	Description
TriggerName	varchar(40)	Name of the trigger.
EventOrder	integer	Order of events: 0 - Before 1 - After
EventOp	integer	Event operation: 0 - Insert 1 - Reinsert 2 - Update 3 - Delete
EventLevel	integer	Trigger level: 0 - Row-level trigger 1 - Statement-level trigger

Table A16: Columns in the catalog.database_triggers Table (2 of 2)

Column Name	Data Type	Description
DatabaseName	varchar (40)	Name of the database.
TableName	varchar (40)	Name of the table.

catalog.signal_triggers Table

The `catalog.signal_triggers` table stores information about signal triggers, including the name of the signal that causes the trigger to fire.

Table A17: Columns in the catalog.signal_triggers Table

Column Name	Data Type	Description
TriggerName	varchar (40)	Name of the trigger.
SignalName	varchar (40)	Name of the signal.

catalog.temporal_triggers Table

The `catalog.temporal_triggers` table stores information about temporal triggers, including how often they fire.

Table A18: Columns in the catalog.temporal_triggers Table

Column Name	Data Type	Description
TriggerName	varchar (40)	Name of the trigger.
Frequency	integer	Trigger frequency in seconds.

catalog.procedures Table

The `catalog.procedures` table stores information about procedures, including whether the procedure is an SQL procedure or an external procedure.

Table A19: Columns in the catalog.procedures Table (1 of 2)

Column Name	Data Type	Description
ProcedureName	varchar (40)	Name of the procedure.

Table A19: Columns in the catalog.procedures Table (2 of 2)

Column Name	Data Type	Description
Kind	unsigned	Procedure type: 0 - SQL 1 - External

catalog.sql_procedures Table

The `catalog.sql_procedures` table stores information about SQL procedures, including the code for the procedure.

Table A20: Columns in the catalog.sql_procedures Table

Column Name	Data Type	Description
ProcedureName	varchar(40)	Name of the procedure.
DeclareBlock	varchar(16384)	Variable declaration specified in the CREATE PROCEDURE command.
CodeBlock	varchar(32768)	The body of the procedure.

catalog.external_procedures Table

The `catalog.external_procedures` table stores information about external procedures, including the name of the procedure executable and the host on which it runs.

Table A21: Columns in the catalog.external_procedures Table

Column Name	Data Type	Description
ProcedureName	varchar(40)	Name of the procedure.
ExecutableName	varchar(1024)	Name of the executable.
HostName	varchar(1024)	Name of the host.
UserId	varchar(1024)	User identifier.
GroupId	varchar(1024)	Group identifier.
ArgumentsSpec	varchar(32768)	Arguments specified in the CREATE PROCEDURE text.

catalog.procedure_parameters Table

The `catalog.procedure_parameters` table stores information about procedure parameters, including parameter types.

Table A22: Columns in the `catalog.procedure_parameters` Table

Column Name	Data Type	Description
ParameterName	varchar (40)	Name of the parameter.
ProcedureName	varchar (40)	Name of the procedure.
ParameterKind	integer	Parameter type: 0 - Base 1 - Row 2 - Array
DataType	integer	Data type of the parameter.
OrdinalPosition	integer	Position in the argument list.
Length	integer	Number of characters in the parameter.
TableName	varchar (40)	If it is a row parameter, this is the parent table of that row. Otherwise this is an empty string.
DatabaseName	varchar (40)	If it is a row parameter, this is the parent database of the parent table of the row. Otherwise this is an empty string.
ParameterMode	integer	Parameter mode: 1 - In 2 - Out 3 - In/Out

catalog.connections Table

The `catalog.connections` table contains information about connections to the ObjectServer.

Table A23: Columns in the `catalog.connections` Table (1 of 2)

Column Name	Data Type	Description
ConnectionID	integer	Connection identifier.

Table A23: Columns in the catalog.connections Table (2 of 2)

Column Name	Data Type	Description
LogName	varchar (40)	Name of the log file for the connected application.
HostName	varchar (40)	Name of the connected host.
AppName	varchar (40)	Name of the connected application.
AppDescription	varchar (40)	Description of the connected application.
IsRealTime	boolean	TRUE if the client does not use IDUC. Desktops and gateways use IDUC and are not real-time connections.
ConnectTime	time	Amount of time the client is connected.

catalog.properties Table

The `catalog.properties` table contains information about ObjectServer properties.

Table A24: Columns in the catalog.properties Table

Column Name	Data Type	Description
PropName	varchar (40)	Name of the property.
PropGroup	varchar (40)	Group of the property, such as <code>Auto</code> or <code>Store</code> . Not all properties belong to a group.
Description	varchar (255)	Description of the property.
Type	integer	Data type of the property.
Value	varchar (255)	Current value of the property.
IsModifiable	boolean	TRUE if the property is modifiable.
IsImmediate	boolean	TRUE if when the property is changed the effect is immediate. Otherwise, the ObjectServer must be restarted.
IsAdvanced	boolean	TRUE if the property is advanced. Advanced properties should not be changed without the assistance of Micromuse Support.

catalog.security_permissions Table

The `catalog.security_permissions` table contains permission information for ObjectServer objects. This table is used only by Netcool/OMNIBus Administrator.

Table A25: Columns in the `catalog.security_permissions` Table

Column Name	Data Type	Description
ApplicationID	integer	Application identifier.
Object	varchar(40)	Name of the object.
ObjectType	integer	Type of object, for example, a table.
ActionID	integer64	Identifier for the permission action. Used only by Netcool/OMNIBus Administrator.
Permission	varchar(40)	Type of permission.

catalog.profiles Table

The `catalog.profiles` table contains timing information for the execution of SQL commands from client connections. See *Statistics Tables* on page 274 for more information.

A.4 Statistics Tables

Statistics tables contain timing information.

The `catalog.profiles` table contains timing information for the execution of SQL commands from client connections.

The `master.stats` table stores timing information about the `alerts.status`, `alerts.details`, and `alerts.journal` tables.

The `catalog.trigger_stats` table stores timing information about triggers.

catalog.profiles Table

The `catalog.profiles` table contains timing information for the execution of SQL commands from client connections. SQL profiling is enabled using the `Profile` property or `-profile` command line option.

SQL profile statistics are also logged to the file `$OMNIHOME/log/servername_profiler_report.logn`, at the interval specified in the `ProfileStatsInterval` property or `-profilestatsinterval` command line option.

Table A26: Columns in the `catalog.profiles` Table (1 of 2)

Column Name	Data Type	Description
ConnectionID	integer	Connection identifier.
UID	integer	User identifier.
AppName	varchar(40)	Name of the connected application.
HostName	varchar(40)	Name of the connected host.
ProfiledFrom	time	Time at which profiling began.
LastSQLTime	real	Duration, in seconds, of the last SQL command.
MinSQLTime	real	Shortest execution time, in seconds, for this client.
MaxSQLTime	real	Longest execution time, in seconds, for this client.
PeriodSQLTime	real	Amount of time, in seconds, the application has spent executing SQL since the last profile report.
TotalSQLTime	real	Total time, in seconds, for the execution of all SQL commands for this client.
LastTimingAt	time	Last time an SQL profile was taken for this client.

Table A26: Columns in the catalog.profiles Table (2 of 2)

Column Name	Data Type	Description
NumSubmits	integer	Number of submissions for this client. A single submission can contain multiple SQL commands, executed with the <code>go</code> command.

master.stats Table

The `master.stats` table stores timing information about the `alerts.status`, `alerts.details`, and `alerts.journal` tables.

This timing information is gathered if the `stats_triggers` trigger group is enabled. The `stats_triggers` trigger group is disabled by default in the `automation.sql` file.

Table A27: Columns in the master.stats Table (1 of 2)

Column Name	Data Type	Description
StatTime	time	The time that the statistics are collected.
NumClients	integer	The total number of clients (for example, desktops) connected to the ObjectServer.
NumRealtime	integer	The number of real-time clients connected to the ObjectServer. Desktops and gateways use IDUC and are real-time connections.
NumProbes	integer	The number of probes connected to the ObjectServer.
NumGateways	integer	The number of gateways connected to the ObjectServer.
NumMonitors	integer	The number of monitors connected to the ObjectServer.
NumProxys	integer	The number of proxy servers connected to the ObjectServer.
EventCount	integer	The current number of entries in the <code>alerts.status</code> table.
JournalCount	integer	The current number of entries in the <code>alerts.journal</code> table.
DetailCount	integer	The current number of entries in the <code>alerts.details</code> table.
StatusInserts	integer	The total number of inserts into the <code>alerts.status</code> table.
StatusNewInserts	integer	The number of new inserts into the <code>alerts.status</code> table.

Table A27: Columns in the master.stats Table (2 of 2)

Column Name	Data Type	Description
StatusDedups	integer	The number of reinserts into the alerts.status table.
JournalInserts	integer	The number of inserts into the alerts.journal table.
DetailsInserts	integer	The number of inserts into the alerts.details table.

catalog.trigger_stats Table

The catalog.trigger_stats table stores timing information about triggers, including the number of times the trigger has been raised and the number of times the trigger has fired. These statistics are gathered unless the automation system is disabled by setting the -autoenabled command line option to FALSE.

Trigger statistics are also logged to the file \$OMNIHOME/log/servername_trigger_stats.logn.

Table A28: Columns in the catalog.trigger_stats Table (1 of 2)

Column Name	Data Type	Description
TriggerName	varchar(40)	Name of the trigger.
PreviousCondition	boolean	Value of the condition the last time the trigger was raised.
PreviousRowcount	unsigned	Number of rows returned by the EVALUATE clause the last time the trigger was raised.
NumZeroRowcount	unsigned	Number of consecutive times the evaluation has returned zero rows.
NumPositiveRowcount	unsigned	Number of consecutive times the evaluation has returned more one or more rows.
PeriodNumRaises	unsigned	Number of times the trigger has been raised since the last report.
PeriodNumFires	unsigned	Number of times the trigger has fired since the last report.
PeriodTime	real	Amount of time trigger has been operating since the last report.
NumRaises	unsigned	Number of times the trigger has been raised.
NumFires	unsigned	Number of times the trigger has been fired.
MaxTime	real	Maximum amount of time the trigger has taken to execute.

Table A28: Columns in the catalog.trigger_stats Table (2 of 2)

Column Name	Data Type	Description
TotalTime	real	Amount of time the trigger has operated since startup.



Note: The `catalog.trigger_stats` system table is updated periodically, based on the setting for the `Auto.StatsInterval` property or `-autostatsinterval` command line option. The default is every 10 seconds.

A.5 Client Tool Support Tables

The client tool support tables are used by the desktop GUIs to display alert information.

alerts.objclass Table

The `alerts.objclass` table is used to determine which menu and icons to use for a particular class of object.

Table A29: Columns in the alerts.objclass Table

Column Name	Data Type	Description
Tag	integer	Primary key column for the table; numeric value for the class.
Name	varchar (64)	Name of the class.
Icon	varchar (255)	Path and file name of the default icon for tools.
Menu	varchar (64)	Name of the tools menu (in the <code>alerts.objmenus</code> table) for the menu associated with this class; enables tools to display the appropriate menu.

alerts.objmenus Table

The `alerts.objmenus` table is used to provide the list of menus.

Table A30: Columns in the alerts.objmenus Table

Column Name	Data Type	Description
Menu	varchar (64)	Primary key field for table; menu name (referred to in the <code>alerts.objclass</code> table).
Columns	integer	Number of columns the menu should use.

alerts.objmenuitems Table

The `alerts.objmenuitems` table is used to provide the content of menus.

Table A31: Columns in the alerts.objmenuitems Table (1 of 2)

Column Name	Data Type	Description
KeyField	varchar (255)	Primary key field for the table.

Table A31: Columns in the alerts.objmenuitems Table (2 of 2)

Column Name	Data Type	Description
Menu	varchar (64)	Menu name (referred to in the alerts.objmenus table).
Sequence	integer	Number used to determine the order of the menu item in a menu when it is displayed.
Title	varchar (64)	Displayed title for the menu item.
Command1	varchar (255)	Command to be executed when this item is selected in a tools menu (first block).
Command2	varchar (255)	Command to be executed (second block).
Command3	varchar (255)	Command to be executed (third block).
Command4	varchar (255)	Command to be executed (fourth block).
RedirectStdin	integer	Currently unused.
RedirectStdout	integer	Boolean; when true (1), standard output is redirected to the dialog in the tool that called the command. Otherwise the output is discarded.
RedirectStderr	integer	Boolean; when true (1), error output is redirected to the dialog in the tool that called the command. Otherwise the error output is discarded.

alerts.resolutions Table

The `alerts.resolutions` table is used to maintain the **Resolutions** option in the event list.

Table A32: Columns in the alerts.resolutions Table (1 of 2)

Column Name	Data Type	Description
KeyField	varchar (255)	Primary key for the table.
Tag	integer	Class value for this resolution.
Sequence	integer	Sequence number which sets ordering at display time.
Title	varchar (64)	Title of the resolution.
Resolution1	varchar (255)	First line of text for the resolution.

Table A32: Columns in the alerts.resolutions Table (2 of 2)

Column Name	Data Type	Description
Resolution2	varchar (255)	Second line of text for the resolution.
Resolution3	varchar (255)	Third line of text for the resolution.
Resolution4	varchar (255)	Fourth line of text for the resolution.

alerts.conversions Table

The `alerts.conversions` table is used to provide easy conversion from a numeric value to a string for any column.

Table A33: Columns in the alerts.conversions Table

Column Name	Data Type	Description
KeyField	varchar (255)	Primary key for the table; internal sequencing string (comprised of Colname and Value).
Colname	varchar (255)	Name of the column this conversion is appropriate for.
Value	integer	Numeric value for the conversion.
Conversion	varchar (255)	String value for the conversion.

alerts.col_visuals Table

The `alerts.col_visuals` table is used to provide the default visuals for columns when displayed in the desktop tools.

Table A34: Columns in the alerts.col_visuals Table (1 of 2)

Column Name	Data Type	Description
Colname	varchar (255)	Name of the column for the visual settings.
Title	varchar (255)	Title of the column when displayed.
DefWidth	integer	Default width of the column when displayed.
MaxWidth	integer	Maximum width of the column when displayed.

Table A34: Columns in the alerts.col_visuals Table (2 of 2)

Column Name	Data Type	Description
TitleJustify	integer	Justification for column title: 0 - left 1 - center 2 - right
DataJustify	integer	Justification for column data: 0 - left 1 - center 2 - right

alerts.colors Table

The `alerts.colors` table is used to create the colors required by the Windows desktop.

Table A35: Columns in the alerts.colors Table (1 of 2)

Column Name	Data Type	Description
Severity	integer	Severity of problem: 0 - Clear 1 - Indeterminate 2 - Warning 3 - Minor 4 - Major 5 - Critical
AckedRed	integer	Red component of the RGB color for acknowledged events. Must be in the range 0-255.
AckedGreen	integer	Green component of the RGB color for acknowledged events. Must be in the range 0-255.
AckedBlue	integer	Blue component of the RGB color for acknowledged events. Must be in the range 0-255.
UnackedRed	integer	Red component of the RGB color for unacknowledged events. Must be in the range 0-255.

Table A35: Columns in the alerts.colors Table (2 of 2)

Column Name	Data Type	Description
UnackedGreen	integer	Green component of the RGB color for unacknowledged events. Must be in the range 0-255.
UnackedBlue	integer	Blue component of the RGB color for unacknowledged events. Must be in the range 0-255.

A.6 Desktop Tools Tables

The desktop tools tables contain information used to configure event list tools.

tools.actions Table

The `tools.actions` table is used to control desktop tools.

Table A36: Columns in the `tools.actions` Table (1 of 2)

Column Name	Data Type	Description
ActionID	integer	The identifier of the tool.
Name	varchar(64)	The name of the tool.
Owner	integer	Indicates whether or not the tool has an owner.
Enabled	integer	Indicates whether or not the tool is enabled.
Description1	varchar(255)	The first line of the description.
Description2	varchar(255)	The second line of the description.
Description3	varchar(255)	The third line of the description.
Description4	varchar(255)	The fourth line of the description.
HasInternal	integer	Indicates whether or not the tool has an internal effect.
InternalEffect1	varchar(255)	The first line of the internal effect.
InternalEffect2	varchar(255)	The second line of the internal effect.
InternalEffect3	varchar(255)	The third line of the internal effect.
InternalEffect4	varchar(255)	The fourth line of the internal effect.
InternalForEach	integer	When set, starts the internal effect for each selected row.
HasExternal	integer	Indicates whether the tool has an external procedure or not.
ExternalEffect1	varchar(255)	The first line of the external procedure.

Table A36: Columns in the tools.actions Table (2 of 2)

Column Name	Data Type	Description
ExternalEffect2	varchar (255)	The second line of the external procedure.
ExternalEffect3	varchar (255)	The third line of the external procedure.
ExternalEffect4	varchar (255)	The fourth line of the external procedure.
ExternalForEach	integer	When set, starts the external procedure for each selected row.
RedirectOut	integer	When selected, output is echoed through a read-only window in the same display as the event list that ran the tool.
RedirectErr	integer	When selected, errors are echoed through a read-only window in the same display as the event list that ran the tool.
Platform	varchar (255)	Indicates the type of platform the external procedure runs on: NT (Windows platforms) UNIX (UNIX platforms) UNIX, NT (UNIX and Windows platforms)
JournalText1	varchar (255)	The first line of the journal entry.
JournalText2	varchar (255)	The second line of the journal entry.
JournalText3	varchar (255)	The third line of the journal entry.
JournalText4	varchar (255)	The fourth line of the journal entry.
JournalForEach	integer	When set, adds the journal entry for each selected row.
HasForcedJournal	integer	Forces a journal entry dialog to be opened when the tool executes.

tools.action_access Table

The `tools.action_access` table is used to control access to desktop tools.

Table A37: Columns in the `tools.action_access` Table

Column Name	Data Type	Description
ActionID	integer	The unique identifier of the tool, taken from the actions table.
GID	integer	Indicates to which group the tool is available.
ClassID	integer	Indicates to which class the tool is available.
ActionAccessID	integer	Primary key field for the table.

tools.menu Table

The `tools.menu` table is used to control desktop tool menus.

Table A38: Columns in the `tools.menu` Table

Column Name	Data Type	Description
MenuID	integer	Primary key field for the menu table.
Name	varchar(64)	Name of the menu.
Owner	integer	Indicates whether the menu has an owner.
Enabled	integer	Indicates whether the menu is enabled or disabled (grayed out).

tools.menu_items Table

The `tools.menu_items` table is used to control desktop tool menu items.

Table A39: Columns in the `tools.menu_items` Table (1 of 2)

Column Name	Data Type	Description
KeyField	varchar(32)	A key field for this menu item. Created from the <code>menu_id</code> (in the <code>tools.menu</code> table) and the <code>menu_item_id</code> .

Table A39: Columns in the tools.menu_items Table (2 of 2)

Column Name	Data Type	Description
MenuID	integer	The unique menu identifier taken from the tools.menu table.
MenuItemID	integer	The primary key identifier for this menu item.
Title	varchar (64)	The name that appears on the menu.
Description	varchar (255)	The description of the menu item.
Enabled	integer	Indicates whether the menu item is enabled or disabled (grayed out).
InvokeType	integer	Indicates the type of menu item: 0 - tool 1 - separator line 2 - submenu
InvokeID	integer	Indicates the action identifier of the action defined in InvokeType.
Position	integer	Indicates the position (order) of this item on the menu.
Accelerator	varchar (32)	Indicates the keyboard short-cut of this menu item.

tools.prompt_defs Table

The tools.prompt_defs table is used to store all prompt definitions.

Table A40: Columns in the tools.prompt_defs Table

Column Name	Data Type	Description
Name	varchar (64)	The name of the prompt.
Prompt	varchar (64)	The prompt title which appears at the top of the prompt window.
Default	varchar (64)	The default value to enter if no value is entered by the user.
Value	varchar (255)	The list of available values.
Type	integer	The prompt type. Type 7 is a link to a menu definition with the same name as the prompt.

tools.menu_defs Table

The `tools.menu_defs` table is used to control desktop tool menu items.

Table A41: Columns in the `tools.menu_defs` Table

Column Name	Data Type	Description
Name	<code>varchar(64)</code>	The name of the menu. This must match the name of the type 7 prompt definition.
DatabaseName	<code>varchar(64)</code>	The database used to build the menu items.
TableName	<code>varchar(64)</code>	The table used to build the menu items.
ShowField	<code>varchar(64)</code>	The field in the table to show as the menu pull-down list.
AssignField	<code>varchar(64)</code>	The actual field used to enter a value into the prompt.
OrderByField	<code>varchar(64)</code>	The field used to order the menu.
WhereClause	<code>varchar(255)</code>	The filter (condition) to show a subset of menu items.
Direction	<code>integer</code>	The order of the menu items: 0 - Ascending 1 - Descending

A.7 Desktop ObjectServer Tables

The `master.national` table is used in a desktop ObjectServer architecture. This table is created when you run `nco_dbinit` using the `-desktopserver` option.

The `master.servergroups` table is used to load-balance desktop connections.



Note: For detailed information about the desktop ObjectServer architecture, see the Netcool/OMNIBus Installation and Deployment Guide.

master.national Table

The `master.national` table identifies the master ObjectServer and the dual-write mode in a desktop ObjectServer architecture.

Table A42: Columns in the master.national Table

Column Name	Data Type	Description
KeyField	incr	Primary key column for table.
MasterServer	varchar (11)	Name of the master ObjectServer in a desktop ObjectServer architecture.
DualWrite	integer	Whether to enable dual-write mode. Dual-write mode enables operators to quickly see the results of tool actions (for example, acknowledge and prioritize) on their dual server desktops. This is done by sending all tool actions to both the desktop ObjectServer and the master ObjectServer. 1 - enable 0 - disable

master.servergroups Table

The `master.servergroups` table is used to load-balance desktop connections.

Table A43: Columns in the master.servergroups Table (1 of 2)

Column Name	Data Type	Description
ServerName	varchar (64)	The name of a desktop ObjectServer.
Group ID	integer	The group identifier to which each desktop ObjectServer belongs. Event list user logins are only distributed among desktop ObjectServers having the same GroupID.

Table A43: Columns in the master.servergroups Table (2 of 2)

Column Name	Data Type	Description
Weight	integer	The priority for each desktop ObjectServer. Higher values attract proportionally more connections. For example, an ObjectServer with a <code>Weight</code> of 2 attracts twice the number of connections as one with a <code>Weight</code> of 1. Load balanced connections are not redirected to ObjectServer with a <code>Weight</code> of 0.

A.8 Security Tables for Backward Compatibility

In previous versions of Netcool/OMNIbus, the `master` database contained user authentication tables to store Netcool/OMNIbus security information. This information is now stored in the `security` database and the `catalog.security_permissions` table.

The `master.names`, `master.members`, and `master.groups` tables provided user and group identification and authorization. The `master.profiles` table provide the user restriction information. These tables are only required for compatibility with previous versions of the desktop.

Appendix B: Desktop Reference

This appendix contains reference information for the Netcool/OMNIbus desktop, including:

- *Event List Command Line Options* on page 292
- *Using the Transient Event List (nco_elct)* on page 296
- *SQL Commands and Variable Expressions in Tools, Automations, and Transient Event Lists* on page 302
- *Changing the Date Format in the Event List on UNIX* on page 305

B.1 Event List Command Line Options

The `nco_event` command opens an event list. You can use the following options when starting the event list from the command line.



Tip: On Windows, the name and location of the executable file is `%OMNIHOME%\desktop\NCOEvent.exe`.

If an instance of the event list (`nco_event`) is running, the `nco_event` command opens a new event list window. If there is no running instance of `nco_event`, the `nco_event` command starts one and displays the monitor box window.

UNIX Command Line Options

The command line options for the UNIX event list are shown in Table B1.

Table B1: UNIX Event List (`nco_event`) Command Line Options (1 of 2)

Command Line Option	Description
<code>-cmap</code>	Specifies a private color map. You cannot abbreviate this command line option.
<code>-config string URL</code> or <code>-elc string URL</code>	<p>Initial configuration file to use.</p> <p><code>string</code> specifies a path to an <code>.elc</code> file.</p> <p>URL specifies an <code>.elc</code> file located on a remote server accessible using the <code>http</code> or <code>ftp</code> protocol. The URL must be of one of the following forms:</p> <ul style="list-style-type: none"> • <code>http://servername[:port]/path/filename.elc</code> • <code>http://username:password@servername[:port]/path/ filename.elc</code> • <code>ftp://servername[:port]/path/ filename.elc</code> • <code>ftp://username:password@servername[:port]/path/ filename.elc</code> <p>The optional port number <code>:port</code> only needs to be specified if the server is not using the default port.</p> <p>For information about event list configurations, see the Netcool/OMNIBus User Guide.</p>
<code>-diagnostic</code>	<p>Displays extended diagnostic messages.</p> <p><i>This option is only useful for detecting problems with the event list. Do not use this command line option unless you are advised by Micromuse Technical Support.</i></p>

Table B1: UNIX Event List (nco_event) Command Line Options (2 of 2)

Command Line Option	Description
<code>-dualwrite integer</code>	<p>In a desktop ObjectServer architecture, indicates whether to enable or disable dual-write mode. Valid options are:</p> <ul style="list-style-type: none"> • 0—Disable dual-write mode • 1—Enable dual-write mode <p>The <code>-dualwrite</code> command line option overrides the <code>DualWrite</code> field entry in the desktop ObjectServer <code>master.national</code> table.</p> <p>For information about desktop ObjectServer architectures, see the Netcool/OMNIbus Installation and Deployment Guide.</p>
<code>-help</code>	Displays help for this command and exits.
<code>-masterserver string</code>	<p>In a desktop ObjectServer architecture, indicates the master ObjectServer to use.</p> <p>The <code>-masterserver</code> command line option overrides the <code>MasterServer</code> field entry in the desktop ObjectServer <code>master.national</code> table.</p> <p>For information about desktop ObjectServer architectures, see the Netcool/OMNIbus Installation and Deployment Guide.</p>
<code>-nlw</code>	Suppresses the of display license expiry warnings.
<code>-password string</code>	Password to log in to the event list.
<code>-server string</code>	Name of the ObjectServer to which you are connecting.
<code>-uneditable</code>	Does not allow the event list configuration to be edited. When this option is used, the Filter Builder and View Builder buttons and the Save and Save As menu items are disabled.
<code>-username string</code>	User name to log in to the event list.
<code>-version</code>	Displays version information and exits.



Tip: The command line options for the event list can be abbreviated to their shortest unique abbreviation. For example, `-uneditable` can be abbreviated to `-un`.

Windows Command Line Options

The command line options for the Windows event list are shown in Table B2.

Table B2: Windows Event List (NCOEvent) Command Line Options (1 of 2)

Command Line Option	Description
<code>-config string URL</code> or <code>-elc string URL</code>	<p>Initial configuration file to use.</p> <p><code>string</code> specifies a path to an <code>.elc</code> file.</p> <p>URL specifies an <code>.elc</code> file located on a remote server accessible using the http or ftp protocol. The URL must be of one of the following forms:</p> <ul style="list-style-type: none"> <code>http://servername[:port]/path/filename.elc</code> <code>http://username:password@servername[:port]/path/filename.elc</code> <code>ftp://servername[:port]/path/filename.elc</code> <code>ftp://username:password@servername[:port]/path/filename.elc</code> <p>The optional port number <code>:port</code> only needs to be specified if the server is not using the default port.</p> <p>For information about event list configurations, see the Netcool/OMNIBus User Guide.</p>
<code>-diagnostic</code>	<p>Displays extended diagnostic messages.</p> <p><i>This option is only useful for detecting problems with the event list. Do not use this command line option unless you are advised by Micromuse Technical Support.</i></p>
<code>-dualwrite integer</code>	<p>In a desktop ObjectServer architecture, indicates whether to enable or disable dual-write mode. Valid options are:</p> <ul style="list-style-type: none"> 0—Disable dual-write mode 1—Enable dual-write mode <p>The <code>-dualwrite</code> command line option overrides the <code>DualWrite</code> field entry in the desktop ObjectServer <code>master.national</code> table.</p> <p>For information about desktop ObjectServer architectures, see the Netcool/OMNIBus Installation and Deployment Guide.</p>
<code>-help</code>	Displays help for this command and exits.
<code>-masterserver string</code>	<p>In a desktop ObjectServer architecture, indicates the master ObjectServer to use.</p> <p>The <code>-masterserver</code> command line option overrides the <code>MasterServer</code> field entry in the desktop ObjectServer <code>master.national</code> table.</p> <p>For information about desktop ObjectServer architectures, see the Netcool/OMNIBus Installation and Deployment Guide.</p>

Table B2: Windows Event List (NCOEvent) Command Line Options (2 of 2)

Command Line Option	Description
<code>-networktimeout</code> <i>integer</i>	Specifies a time in seconds after which a login attempt or connection to the ObjectServer will time out, should a network failure occur. After the specified timeout period, the event list attempts to reconnect to the ObjectServer. If the connection is unsuccessful after a second timeout period, the event list will attempt to connect to a backup ObjectServer, where available. The default is 20 seconds.
<code>-password</code> <i>string</i>	Password to log in to the event list.
<code>-server</code> <i>string</i>	Name of the ObjectServer to which you are connecting.
<code>-username</code> <i>string</i>	User name to log in to the event list.
<code>-version</code>	Displays version information and exits.

B.2 Using the Transient Event List (nco_elct)

The `nco_elct` utility enables you to open a customized, transient event list. You can use `nco_elct`:

- Directly from the command line
- In a script
- As part of an event list tool

For example, you can use `nco_elct` to open an event list and apply a filter to view all `critical` alerts from a particular ObjectServer.



Tip: On Windows, the name and location of the executable file is `%OMNIHOME%\desktop\NCOelct.exe`.

If an instance of the event list (`nco_event`) is running, the `nco_elct` utility opens a new event list window. If there is no running instance of `nco_event`, the `nco_elct` utility starts one and displays the monitor box window and an event list window using the filter specified at the command line.

UNIX Command Line Options

The command line options for the UNIX `nco_elct` utility are shown in Table B3.

Table B3: `nco_elct` Command Line Options (1 of 2)

Command Line Option	Description
<code>-cmap</code>	Specifies a private color map.
<code>-diagnostic</code>	Displays extended diagnostic messages.
<code>-dualwrite 1 0</code>	<p>In a desktop ObjectServer architecture, indicates whether to enable or disable dual-write mode. Valid options are:</p> <ul style="list-style-type: none"> • 0—Disable dual-write mode • 1—Enable dual-write mode <p>The <code>-dualwrite</code> command line option overrides the <code>DualWrite</code> field entry in the desktop ObjectServer <code>master.national</code> table.</p> <p>For information about desktop ObjectServer architectures, see the Netcool/OMNIBus Installation and Deployment Guide.</p>

Table B3: nco_elct Command Line Options (2 of 2)

Command Line Option	Description
<code>-ffile string URL</code>	Specifies the filter file to use for the transient event list. Event list filters have an <code>.elf</code> extension. Note: You can also retrieve remote <code>.elf</code> files using <code>http</code> and <code>ftp</code> protocols, as described for the <code>-config</code> command line option in Table B1 on page 292.
<code>-fmetric string</code>	Specifies a filter metric for the transient event list. This is ignored if you have specified an existing event list filter file using the <code>-ffile</code> .
<code>-fname string</code>	Specifies a name for the filter. This is ignored if you have specified an existing event list filter file using the <code>-ffile</code> option.
<code>-ftext string</code>	The SQL statement to use for filtering alerts. This is ignored if you have specified an existing event list filter file using the <code>-ffile</code> option.
<code>-help</code>	Displays help for this command and then exits.
<code>-masterserver string</code>	In a desktop ObjectServer architecture, indicates the master ObjectServer to use. The <code>-masterserver</code> command line option overrides the <code>MasterServer</code> field entry in the desktop ObjectServer <code>master.national</code> table. For information about desktop ObjectServer architectures, see the Netcool/OMNIBus Installation and Deployment Guide.
<code>-nlw</code>	Suppresses the of display license expiry warnings.
<code>-password string</code>	The password for the selected user name. The <code>nco_elct</code> utility only accepts alphanumeric characters in the password. Special characters such as <code>#</code> , <code>/</code> , <code>></code> , <code>@</code> , and <code>!</code> are not permitted.
<code>-server string</code>	Name of the ObjectServer to which you are connecting.
<code>-username string</code>	User name to log in to the event list.
<code>-version</code>	Displays version information and exits.
<code>-vfile string URL</code>	Opens a transient event list using a specified event list view. Event list views have an <code>.elv</code> file extension. Note: You can also retrieve remote <code>.elv</code> files using <code>http</code> and <code>ftp</code> protocols, as described for the <code>-config</code> command line option in Table B1 on page 292.

Windows Command Line Options

The command line options for the Windows `NCOelct.exe` utility are shown in Table B4.

Table B4: NCOelct.exe Command Line Options (1 of 2)

Command Line Option	Description
<code>-diagnostic</code>	Displays extended diagnostic messages.
<code>-dualwrite 1 0</code>	<p>In a desktop ObjectServer architecture, indicates whether to enable or disable dual-write mode. Valid options are:</p> <ul style="list-style-type: none"> • 0—Disable dual-write mode • 1—Enable dual-write mode <p>The <code>-dualwrite</code> command line option overrides the <code>DualWrite</code> field entry in the desktop ObjectServer <code>master.national</code> table.</p> <p>For information about desktop ObjectServer architectures, see the Netcool/OMNibus Installation and Deployment Guide.</p>
<code>-elf string URL</code>	<p>Specifies the filter file to use for the transient event list. Event list filters have an <code>.elf</code> file extension.</p> <p>This is a required option.</p> <p>You can use the <code>-params</code> command line option to specify filter parameters in addition to those specified in the <code>.elf</code> file. See <code>-params</code>, in this table.</p> <p>Note: You can also retrieve remote <code>.elv</code> files using <code>http</code> and <code>ftp</code> protocols, as described for the <code>-config</code> command line option in Table B1 on page 292.</p>
<code>-elv string URL</code>	<p>Specifies the event list view file to use for the transient event list. Event list views have an <code>.elv</code> file extension.</p> <p>Note: You can also retrieve remote <code>.elv</code> files using <code>http</code> and <code>ftp</code> protocols, as described for the <code>-config</code> command line option in Table B1 on page 292.</p>
<code>-help</code>	Displays help for this command and then exits.
<code>-masterserver string</code>	<p>In a desktop ObjectServer architecture, indicates the master ObjectServer to use.</p> <p>The <code>-masterserver</code> command line option overrides the <code>MasterServer</code> field entry in the desktop ObjectServer <code>master.national</code> table.</p> <p>For information about desktop ObjectServer architectures, see the Netcool/OMNibus Installation and Deployment Guide.</p>

Table B4: NCOelct.exe Command Line Options (2 of 2)

Command Line Option	Description
<code>-networktimeout</code> <i>integer</i>	Specifies a time in seconds after which a login attempt or connection to the ObjectServer will time out, should a network failure occur. After the specified timeout period, the event list attempts to reconnect to the ObjectServer. If the connection is unsuccessful after a second timeout period, the event list will attempt to connect to a backup ObjectServer, where available. The default is 20 seconds.
<code>-params</code> <i>string</i>	Use to specify a string to replace a single placeholder in the <i>filter_text</i> parameter of the filter file. This placeholder must begin with the @ symbol (for example, @FilterText). See <i>Using nco_elct Example - Command Line or Script</i> on page 299 and <i>Using nco_elct Example - In a Tool</i> on page 300.
<code>-password</code> <i>string</i>	The password for the selected user name. The <i>nco_elct</i> utility only accepts alphanumeric characters in the password. Special characters such as #, /, >, @, and ! are not permitted.
<code>-server</code> <i>string</i>	The ObjectServer to which you are connecting.
<code>-username</code> <i>string</i>	The user name to connect as.
<code>-version</code>	Displays version information and exits.

Using nco_elct Example - Command Line or Script

The following examples show *nco_elct* commands for both UNIX and Windows platforms. These examples can be run directly from the command line or from a script.



Example nco_elct Command on UNIX

The following command creates a transient event list from the NCOMS ObjectServer using the `default.elv` view file. Only events from the node `wombat` will appear in the event list.

```
$OMNIHOME/bin/nco_elct -server NCOMS -username root -password "" -vfile
"$OMNIHOME/desktop/default.elv" -ftext "( Node = 'wombat' )"
```



Example nco_elct Command on Windows

The following command creates a transient event list from the NYC ObjectServer using the `lvview.elv` view file and the filter file `tool.elf`.

```
%OMNIHOME%\desktop\NCOelct.exe -server NYC -username root -password "" -elv
"%OMNIHOME%\ini\lvview.elv" -elf "%OMNIHOME%\ini\tool.elf" -params "wombat"
```

The `tool.elf` file referenced in the above `nco_elct` command is shown below:

```
filter_name = 'ToolFilter';
filter_text = '( Node = \'@NodeName\')';
filter_metric = 'avg(Severity)';
# End of file
```

When you run the command or script, the string `wombat` that follows `-params` option replaces the `@NodeName` placeholder in the `tool.elf` file, so that only events from the node `wombat` appear in the event list.

Using `nco_elct` Example - In a Tool

The following examples show how `nco_elct` is used in the **Show Related FE Node** and **Show Related FE Node (Windows)** tools, which are shipped with Netcool/OMNIbus. For information about creating tools, see *Configuring Tools* on page 75.



Example `nco_elct` Command Used In the Show Related FE Node Tool

```
$OMNIHOME/bin/nco_elct -server "%server" -username "%username" -password "%password"
-vfile "$OMNIHOME/desktop/default.elv" -ftext "( RemoteNodeAlias != '' and
RemoteNodeAlias = '@LocalNodeAlias' ) or ( LocalNodeAlias != '' and LocalNodeAlias =
'@RemoteNodeAlias' ) or Node = '@RemoteNodeAlias' or RemoteNodeAlias = '@Node' "
```



Example `nco_elct` Command Used In the Show Related FE Node Tool (Windows)

```
"$(OMNIHOME)/desktop/NC0elct.exe" -server "%server" -username "%username" -password
"%password" -vfile "$(OMNIHOME)/ini/default.elv" -elf "$(OMNIHOME)/ini/tool.elf"
-params "( RemoteNodeAlias != '\\\\' and RemoteNodeAlias = '\\\\@LocalNodeAlias\\\\' ) or
( LocalNodeAlias != '\\\\' and LocalNodeAlias = '\\\\@RemoteNodeAlias\\\\' ) or Node =
'\\@RemoteNodeAlias\\\\' or RemoteNodeAlias = '\\@Node\\\\' "
```

The `tool.elf` file referenced in the above `nco_elct` command is shown below:

```
filter_name = 'ToolFilter';
filter_text = '@FilterText';
filter_metric = 'avg(Severity)';
# End of file
```

The `OMNIHOME` environment variable for Windows is expressed differently in a tool than the normal usage (`%OMNIHOME%`). This is because the variable is interpreted and resolved by the built-in tools parser and not by the Windows operating system.

When you run the tool from the event list, the entire string that follows `-params` option replaces the `@FilterText` placeholder in the `tool.elf` file. The `@LocalNodeAlias`, `@RemoteNodeAlias`, and `@Node` placeholders are replaced in the `-params` string by values from the currently selected event in the event list.

B.3 SQL Commands and Variable Expressions in Tools, Automations, and Transient Event Lists

You can use the following SQL commands and variable expressions to retrieve information from a running event list, the current event, or the operating system environment. You can use these expressions when creating a tool, trigger, or SQL procedure, or in parameters passed to a transient event list.

Table B5: SQL Commands and Variable Expressions in Tools, Triggers, Procedures, and the Transient Event List (1 of 3)




Command/Variable Expression	Button	Usage
<code>select_command</code> <code>insert_command</code> <code>update_command</code> <code>delete_command</code> <code>use_command</code> <code>service_command</code>		<p>Click this button to select an SQL command from the pop-up menu. After selecting a command, the statement window is displayed. The available fields on the statement window change depending on the selected command, as described below:</p> <p>Select—In the From area, select the database and table on which to execute the command. Then, choose the table columns to select.</p> <p>Insert—In the From area, select the database and table on which to execute the command. Then, select the table columns in which to insert values. For each selected column, enter the value to insert. For <code>insert</code> statements, you must include the primary key. Primary keys are indicated with an asterisk (*).</p> <p>Update—In the From area, select the database and table on which to execute the command. Then, select the table columns to update. For each selected column, enter the new value. For <code>update</code> statements, you must <i>exclude</i> the primary key. Primary keys are indicated with an asterisk (*).</p> <p>Note: For inserts and updates to the <code>alerts.status</code> table, any existing conversions appear in the drop-down lists.</p> <p>Delete—Select the table to delete.</p> <p>Use—Select the database to use.</p> <p>Service—Select a service name and a value. Values can be Good, Marginal, or Bad.</p>
<code>column_name</code> <code>@column_name</code>		<p>Click this button to select a table column name.</p> <p>When prefaced with the @ symbol, the column name is substituted with the corresponding event list row value during execution. This can be used in an SQL query or restriction filter, such as:</p> <pre>RemoteNodeAlias = '@LocalNodeAlias'</pre>
<code>conversion_name</code>		<p>Click this button to select from a list of available conversions. For information about conversions, see <i>Configuring Conversions</i> on page 96.</p>

Table B5: SQL Commands and Variable Expressions in Tools, Triggers, Procedures, and the Transient Event List (2 of 3)




Command/Variable Expression	Button	Usage
N/A		Click this button to check the validity of SQL syntax.
%internal_value		<p>Click this button to select from a list of internal values known to the current instance of the event list.</p> <p>For example, to run transient event list and specify the ObjectServer to connect to using the <code>-server</code> command line option, specify:</p> <pre>-server "%server"</pre> <p>The following internal values are available:</p> <p>%display—The current display running the application (UNIX only).</p> <p>%password—The password of the user running the application.</p> <p>%encrypted_password—The encrypted password of the user running the application (UNIX only).</p> <p>%server—The name of the ObjectServer to which the tool is currently connected.</p> <p>%desktopserver—The name of the desktop ObjectServer to which the tool is currently connected.</p> <p>%uid—The ObjectServer user identifier of the user running the application.</p> <p>%username—The ObjectServer user name of the user running the application.</p> <p>You can use internal values in tools and as a parameter to the transient event list.</p>
\$prompt.prompt_name		<p>Click this button to select the name of the prompt to use when querying the user.</p> <p>For example, to run the transient event list and prompt the user to enter their password using the <code>Password</code> prompt, specify:</p> <pre>-password \$prompt.Password</pre> <p>You can use prompts in tools and as a parameter to the transient event list.</p>

Table B5: SQL Commands and Variable Expressions in Tools, Triggers, Procedures, and the Transient Event List (3 of 3)

Command/Variable Expression	Button	Usage
<code>\$(selected_rows.column_name)</code>	N/A	<p>List of values of <i>column_name</i> for all selected alerts. For example:</p> <pre>update alerts.status set TaskList = 0 where Serial in (\$(selected_rows.serial)</pre> <p>Do not use this syntax if you select the Execute for each selected row check box. Instead, select the check box if the change is different for each alert.</p>
<code>\$(environment_variable)</code>	N/A	<p>Indicates an environment variable. For example, when you run a transient event list, you can specify the filter file using the <code>-elf</code> command line option, such as:</p> <pre>-elf "\$(OMNIHOME)/ini/tool.elf</pre> <p>To run the tool on Windows, enclose the environment variable, such as <code>\$(OMNIHOME)</code>, in double quotes. If there is space in the path name, it will not be interpreted correctly.</p>



Tip: You can click the **To Clipboard** button to copy the command in a text format to the clipboard.

B.4 Changing the Date Format in the Event List on UNIX

The date format in the event list is defined by the `NCO_TIME` environment variable. The default format is `mm/dd/yy hh:mm:ss`; for example, `11/12/03 20:13:36`. This format is determined by the POSIX `strptime` function.

To set the default format, using `csh`, enter the following command:

```
setenv NCO_TIME %m/%d/%y %H:%M:%S
```

To set the default format, using `ksh` or `sh`, enter the following command:

```
NCO_TIME=%m/%d/%y %H:%M:%S;export NCO_TIME
```

See the man page for `strptime` for details of the conversion specifiers that are available.



Note: Changing the date format in the event list does not affect the alert timestamp.

Index

Symbols

\${COMMAND} 233
 \${EUID} 233
 \${HOST} 233
 \${NAME} 233

A

alert severity colors
 creating (nco_config) 99
 alert syslog messages 233
 alerts database 135
 alerts.col_visuals table 280
 alerts.colors table 281
 alerts.conversions table 280
 alerts.details table 258
 alerts.journal table 259
 alerts.objclass table 278
 alerts.objmenuitems table 278
 alerts.objmenus table 278
 alerts.resolutions table 279
 alerts.status table 252
 AlertSecurityModel property (nco_objserv) 20
 AllowConnections property (nco_objserv) 20
 AllowISQL property (nco_objserv) 20
 AllowISQLWrite property (nco_objserv) 20
 AllowTimedRefresh property (nco_objserv) 21
 ALTER FILE 145
 ALTER GROUP 180
 ALTER ROLE 182
 ALTER SYSTEM 174
 ALTER TABLE 138
 ALTER TRIGGER 206

ALTER TRIGGER GROUP 202
 ALTER USER 178
 ApplicationFile property (nco_dbinit) 12
 arithmetic operators 151
 attributes
 system signals 212
 audit.file.max.size property (nco_config) 46
 audit.file.name property (nco_config) 46
 authorization security 55
 AuthPassword property (nco_objserv) 30
 AuthPassword property (nco_proxyserv) 35
 AuthUserName property (nco_objserv) 30
 AuthUserName property (nco_proxyserv) 35
 Auto.Debug property (nco_objserv) 21
 Auto.Enabled property (nco_objserv) 21
 Auto.StatsInterval property (nco_objserv) 21
 Auto.StatsLogfile property (nco_objserv) 21
 automation 201
 AutomationFile property (nco_dbinit) 12

B

BackupObjectServer property (nco_objserv) 22
 binary comparison operators 151
 BOOLEAN ObjectServer data type 136

C

CALL PROCEDURE 199
 catalog database 262
 catalog.base_tables table 263
 catalog.columns table 265
 catalog.connections table 271
 catalog.database_triggers table 268

- catalog.databases table 262
 - catalog.external_procedures table 270
 - catalog.files table 264
 - catalog.primitive_signal_parameters table 266
 - catalog.primitive_signals table 266
 - catalog.procedure_parameters table 271
 - catalog.procedures table 269
 - catalog.profiles table 274
 - catalog.properties table 272
 - catalog.restrictions table 265
 - catalog.security_permissions table 273
 - catalog.signal_triggers table 269
 - catalog.sql_procedures table 270
 - catalog.tables table 262
 - catalog.temporal_triggers table 269
 - catalog.trigger_groups table 267
 - catalog.trigger_stats table 276
 - catalog.triggers table 267
 - catalog.views table 264
 - CHAR ObjectServer data type 137
 - CHECK STATEMENT 176
 - checkpoint verification utility 40
 - checkpointing the ObjectServer 39
 - classes
 - creating and editing (nco_config) 103
 - deleting (nco_config) 104
 - column visuals
 - creating and editing (nco_config) 101
 - deleting (nco_config) 102
 - columns
 - adding and dropping (SQL) 138
 - adding and editing (nco_config) 109
 - dropping (nco_config) 111
 - ObjectServer 137
 - command line options
 - nco_dbinit 12
 - ObjectServer 19
 - proxy server 35
 - UNIX nco_elct 296
 - UNIX nco_event 292
 - Windows NCOelct.exe 298
 - Windows NCOEvent.exe 294
 - comparison operators 151, 154
 - compound expressions 162
 - conditions 162
 - configuration files
 - elc files 292
 - elf files 297
 - elv files 297
 - process control 231
 - ConnectionRatio property (nco_proxyserv) 35
 - Connections property (nco_objserv) 22
 - conversions
 - creating and editing (nco_config) 97
 - deleting (nco_config) 98
 - CopyPropsFile property (nco_dbinit) 12
 - CREATE DATABASE 134
 - CREATE FILE 143
 - CREATE GROUP 179
 - CREATE RESTRICTION FILTER 142
 - CREATE ROLE 181
 - CREATE SIGNAL 217
 - CREATE TABLE 135
 - CREATE TRIGGER 203
 - CREATE TRIGGER GROUP 202
 - CREATE USER 177
 - CREATE VIEW 140
 - custom database 135
- ## D
- Data Definition Language (DDL) 126
 - Data Manipulation Language (DML) 126, 164
 - data types 136
 - database triggers 201
 - altering (SQL) 206
 - creating (SQL) 207

- creating and editing (nco_config) 86
- databases
 - creating (nco_config) 107
 - creating (SQL) 134
 - dropping (nco_config) 108
 - dropping (SQL) 134
 - initialization files 11
 - system-initialized 134
- date format
 - changing in UNIX event list 305
- deduplication
 - trigger example 220
- DELETE 166
- DeleteLogFile property (nco_objserv) 22
- DeleteLogging property (nco_objserv) 22
- DeleteLogLevel property (nco_objserv) 22
- DeleteLogSize property (nco_objserv) 23
- DESCRIBE 172
- desktop ObjectServer architecture
 - event list command line options 293, 294
- DesktopFile property (nco_dbinit) 12
- DesktopServer property (nco_dbinit) 13
- DesktopServerFile property (nco_dbinit) 13
- DROP DATABASE 134
- DROP FILE 145
- DROP GROUP 181
- DROP RESTRICTION FILTER 143
- DROP ROLE 183
- DROP SIGNAL 219
- DROP TABLE 139
- DROP TRIGGER 206
- DROP TRIGGER GROUP 202
- DROP USER 179
- DROP VIEW 141
- DTMaxTopRows property (nco_objserv) 23
- dynamic choice prompt
 - creating (nco_config) 82

E

- editing
 - probe properties 48
- editor syntax coloring
 - configuring (nco_config) 54
- elc files 292
- elf files 297
- elv files 297
- encrypting
 - passwords for process control 236
 - passwords for SQL scripts 132
 - passwords for the ObjectServer 30
 - passwords for the proxy server 38
- event list
 - changing date format 305
 - configuring menus (nco_config) 71
 - configuring severity colors (nco_config) 98
 - UNIX command line options 292
 - Windows command line options 294
- example triggers 220
- EXECUTE PROCEDURE 199
- expansion keywords 233
- expressions
 - compound 162
- external procedures
 - creating (SQL) 198
 - creating and editing (nco_config) 95
 - example 96

F

- files
 - altering (SQL) 145
 - creating (SQL) 143
 - creating and editing (nco_config) 117
 - deleting (nco_config) 118
 - dropping (SQL) 145
 - truncating (nco_config) 117
 - truncating (SQL) 145
- fixed choice prompt

- creating (nco_config) 82
- float prompt
 - creating (nco_config) 81
- Force property (nco_dbinit) 13
- functions (ObjectServer) 157

G

- GRANT 183
- GRANT ROLE 187
- Granularity property (nco_objserv) 24, 32
- groups
 - adding and removing users (nco_config) 62, 65
 - altering (SQL) 180
 - assigning and removing restriction filters (nco_config) 66
 - creating (SQL) 179
 - creating and editing (nco_config) 59
 - deleting (nco_config) 62
 - dropping (SQL) 181
- GWDEDUPLICATION property (nco_objserv) 24

H

- hidden columns 137

I

- Identifier field 125
- IDUC 25, 32
- Iduc.ListeningHostname property (nco_objserv) 25
- Iduc.ListeningPort property (nco_objserv) 25
- INCR ObjectServer data type 136
- inheritance of permissions 186
- initializing
 - ObjectServer 11
- INSERT 164
- Insert, Delete, Update or Control
 - see IDUC
- INTEGER ObjectServer data type 136
- integer prompt

- creating (nco_config) 81
- INTEGER64 ObjectServer data type 137
- interfaces file
 - adding process control hosts 229
- IpC.SSLCertificate property (nco_objserv) 25
- IpC.SSLEnable property (nco_objserv) 25
- IpC.SSLPrivateKeyPassword property (nco_objserv) 26

J

- java.security.policy property (nco_config) 46

L

- LastOccurrence field 125
- license.file property (nco_config) 47
- LIKE comparison 153
- list comparison operators 154
- log.file.max.size property (nco_config) 47
- log.file.name property (nco_config) 47
- logging
 - using ObjectServer files 143
- logging out
 - of Netcool/OMNIBus Administrator 49
- logical operators 155
- look.and.feel property (nco_config) 47
- lookup prompt
 - creating (nco_config) 82

M

- master database 135
- master.national table 288
- master.servergroups table 288
- master.stats table 275
- math operators 151
- MaxConnections property (nco_proxyserv) 36
- Memstore.DataDirectory property (nco_dbinit) 13
- menus (nco_config)

- adding separators 73
- adding sub-menus 73
- adding tools 73
- changing contents order 74
- changing the order of menu items 74
- configuring for the desktop 71
- editing menu items 73
- removing menu items 75
- renaming menu items 73
- testing 75

MessageLevel property (nco_dbinit) 13

MessageLevel property (nco_objserv) 26

MessageLog property (nco_dbinit) 14

MessageLog property (nco_objserv) 26

N

nco_check_store 40

nco_config (see also Netcool/OMNIBus Administrator)

- properties and command line options 46
- property and command line processing order 48

nco_dbinit

- properties and command line options 12

nco_elct 296

nco_g_crypt 30, 38, 237

nco_jdbc.timeout property (nco_config) 47

nco_new_server

- see nco_dbinit 11

nco_objserv 17, 19

nco_pa.conf 231

nco_pa_addentry 248

nco_pa_crypt 236

nco_pa_shutdown 247

nco_pa_start 245

nco_pa_status 244

nco_pa_stop 246

nco_pad 229, 240

nco_proxyserv 35

nco_sql 18, 128

- paths in 131
- sending text files to 131

nco_sql_crypt 132

ncoadmin user group 228

NCOelct.exe 298

NCOMS.props 11, 17

Netcool/OMNIBus Administrator (nco_config)

- configuring editor syntax coloring 54
- overview 50
- selecting columns to display 53
- sorting window columns 52
- SSL connection in 50
- starting on UNIX 45
- starting on Windows 45

NetworkTimeout property (nco_proxyserv) 36

O

object permissions 184

ObjectServer

- ALTER SYSTEM 18
- changing properties (nco_config) 104
- checkpointing 39
- column properties 137
- configuration options 17
- configuring (nco_config) 68
- configuring with alter system commands 19, 174
- creating 11
- data types 136
- database initialization files 11
- database rebuild on startup 40
- deduplication 124
- file creation sequence (nco_config) 116
- file creation sequence (SQL) 145
- files (nco_config) 115
- files (SQL) 143
- initializing 11
- maintaining database table files on disk 39
- object naming conventions 68
- properties and command line options 19
- security 177

- sending SQL commands to (nco_config) 119
- sending SQL commands to (SQL) 131
- shutdown 18, 174
- SQL 126
- starting manually 17
- starting using process control 16
- stopping manually 17
- stopping using process control 16
- ObjectServer files (see also files)
 - altering (SQL) 145
 - creating (SQL) 143
 - creating and editing (nco_config) 117
 - deleting (nco_config) 118
 - dropping (SQL) 145
 - truncating (nco_config) 117
 - truncating (SQL) 145
- ObjectServerPropsFile property (nco_dbinit) 14
- omni.home property (nco_config) 48
- online help
 - selecting web browser (nco_config) 54
- operators
 - binary comparison 151
 - list comparison 154
 - logical 155
 - math 151
 - precedence of 157
 - string 151
- P**
- PA.Name property (nco_objserv) 27
- PA.Password property (nco_objserv) 27
- PA.Username property (nco_objserv) 27
- password encryption 30, 38, 132, 236
- password prompt
 - creating (nco_config) 82
- pattern matching 153
- permissions
 - granting 183
 - inheritance 186
 - object 184
 - revoking 186
 - system 183
- persist database 135
- precedence of operators 157
- primary keys 137
- probes
 - editing properties 48
- procedures
 - calling (SQL) 199
 - creating and editing external (nco_config) 95
 - creating and editing SQL (nco_config) 93
 - creating external (SQL) 198
 - creating SQL (SQL) 189
 - deleting (nco_config) 96
 - dropping (SQL) 200
 - executing (SQL) 199
- process control
 - adding a service or process 248
 - adding host names to interfaces file 229
 - alert and restart syslog messages 233
 - automatically starting the agent 229
 - configuring hosts 228
 - creating user groups 228
 - defining processes 231
 - defining routing hosts 236
 - defining secure hosts 235
 - defining services 233
 - example configuration file 238
 - executing automations 224
 - expansion keywords 233
 - installing process agent startup scripts 230
 - managing processes 224
 - manually starting the agent 229
 - PA aware processes 232
 - running 240
 - shutting down the agent 247
 - starting a service or process 245
 - starting an ObjectServer 16
 - stopping a service or process 246
 - stopping an ObjectServer 16
 - viewing service status 244

- Windows services 16
 - processes
 - defining for process control 231
 - executing using process control 226
 - Profile property (nco_objserv) 27
 - ProfileStatsInterval property (nco_objserv) 27
 - prompts
 - creating and editing (nco_config) 80
 - deleting (nco_config) 83
 - dynamic choice 82
 - fixed choice 82
 - float 81
 - integer 81
 - lookup 82
 - password 82
 - string 81
 - time 82
 - types of 81
 - properties
 - changing ObjectServer (nco_config) 104
 - changing ObjectServer (SQL) 174
 - nco_dbinit 12
 - ObjectServer 19
 - proxy server 35
 - Props.CheckNames property (nco_dbinit) 14
 - Props.CheckNames property (nco_objserv) 28
 - proxy server
 - command line options 35
 - connecting 34
 - properties 35
 - running in secure mode 38
 - starting 34
- R**
- RAISE SIGNAL 219
 - REAL ObjectServer data type 136
 - regular expressions 153
 - RemoteServer property (nco_proxyserv) 36
 - reserved words (ObjectServer) 147
 - restart syslog messages 233
 - restriction filters
 - creating (nco_config) 70
 - creating (SQL) 142
 - deleting (nco_config) 71
 - dropping (SQL) 143
 - RestrictionEnforceUpdateCheck property (nco_objserv) 28
 - RestrictPasswords property (nco_dbinit) 14
 - RestrictPasswords property (nco_objserv) 28
 - RestrictProxySQL property (nco_objserv) 28
 - REVOKE 186
 - REVOKE ROLE 188
 - roles
 - altering (SQL) 182
 - creating (SQL) 181
 - creating and editing (nco_config) 56
 - deleting (nco_config) 58
 - dropping (SQL) 183
 - granting (SQL) 187
 - revoking (SQL) 188
- S**
- Sec.AuditLevel property (nco_dbinit) 14
 - Sec.AuditLevel property (nco_objserv) 28
 - Sec.AuditLog property (nco_dbinit) 14
 - Sec.AuditLog property (nco_objserv) 29
 - secure mode
 - in the ObjectServer 30
 - in the proxy server 38
 - SecureMode property (nco_objserv) 29
 - SecureMode property (nco_proxyserv) 37
 - security
 - authorization 55
 - ObjectServer 177
 - SQL commands 177
 - security database 134
 - SecurityFile property (nco_dbinit) 14
 - separators

- adding to menus (nco_config) 73
- server property (nco_config) 48
- Server property (nco_dbinit) 14
- Server property (probes) 34
- ServerName property (nco_proxyserv) 37
- service database 135
- service.status table 261
- services
 - defining for process control 233
 - executing using process control 227
- session SQL commands 174
- SET DATABASE 175
- shutdown
 - see ALTER SYSTEM command 18, 174
- signal triggers 201, 211
 - altering (SQL) 206
 - creating (SQL) 211
 - creating and editing (nco_config) 87
 - example 205
- signals
 - system 211, 212
 - user defined 211, 217
- SQL (ObjectServer)
 - ALTER FILE 145
 - ALTER GROUP 180
 - ALTER ROLE 182
 - ALTER SYSTEM 174
 - ALTER TABLE 138
 - ALTER TRIGGER 206
 - ALTER TRIGGER GROUP 202
 - ALTER USER 178
 - CALL PROCEDURE 199
 - CHECK STATEMENT 176
 - conditions 162
 - CREATE DATABASE 134
 - CREATE FILE 143
 - CREATE GROUP 179
 - CREATE RESTRICTION FILTER 142
 - CREATE ROLE 181
 - CREATE SIGNAL 217
 - CREATE TABLE 135
 - CREATE TRIGGER 203
 - CREATE TRIGGER GROUP 202
 - CREATE USER 177
 - CREATE VIEW 140
 - DELETE 166
 - DESCRIBE 172
 - DROP DATABASE 134
 - DROP FILE 145
 - DROP GROUP 181
 - DROP RESTRICTION FILTER 143
 - DROP ROLE 183
 - DROP SIGNAL 219
 - DROP TABLE 139
 - DROP TRIGGER 206
 - DROP TRIGGER GROUP 202
 - DROP USER 179
 - DROP VIEW 141
 - EXECUTE PROCEDURE 199
 - expressions 161
 - external procedures 95, 198
 - functions 157
 - GRANT 183
 - GRANT ROLE 187
 - INSERT 164
 - keywords 147
 - operators 150
 - procedures 189
 - RAISE SIGNAL 219
 - reserved words 147
 - REVOKE 186
 - REVOKE ROLE 188
 - session commands 174
 - SET DATABASE 175
 - SQL procedures 189
 - SVC UPDATE 172
 - system commands 174
 - UPDATE 165
 - USE DATABASE 175
 - using the SQL interactive interface (iSQL) 119
 - using the SQL interactive interface (nco_sql) 128
 - WRITE INTO 171

- SQL procedures
 - creating (SQL) 189
 - creating and editing (nco_config) 93
 - example 94
 - SSLCertificate property (nco_proxyserv) 36
 - SSLEnable property (nco_proxyserv) 36
 - SSLPrivateKeyPassword property (nco_proxyserv) 36
 - starting
 - ObjectServer manually 17
 - ObjectServer using process control 16
 - proxy server 34
 - services using process control 245
 - transient event list on UNIX 296
 - transient event list on Windows 298
 - stopping
 - ObjectServer manually 17
 - ObjectServer using process control 16
 - process control 247
 - services using process control 246
 - storage structures
 - memstores 39
 - string operators 151
 - string prompt
 - creating (nco_config) 81
 - sub-menus
 - adding to menus (nco_config) 73
 - syslog messages 233
 - system permissions 183
 - system signal attributes 212
 - system signals 211, 212
 - system SQL commands 174
 - system tables 139, 262
 - SystemFile property (nco_dbinit) 15
- T**
- table columns
 - adding and editing (nco_config) 109
 - tables
 - altering (SQL) 138
 - creating (nco_config) 108
 - creating (SQL) 135
 - dropping (nco_config) 109
 - dropping (SQL) 139
 - editing columns (nco_config) 109
 - system 139, 262
 - Tally field 125
 - temporal triggers 201
 - altering (SQL) 206
 - creating (SQL) 210
 - creating and editing (nco_config) 89
 - testing
 - menus (nco_config) 75
 - text files
 - sending to nco_sql 131
 - TIME ObjectServer data type 136
 - time prompt
 - creating (nco_config) 82
 - tools
 - adding to menus (nco_config) 73
 - creating and editing (nco_config) 76
 - deleting (nco_config) 79
 - using nco_elct in 296
 - tools database 135
 - tools.action_access table 285
 - tools.actions table 283
 - tools.menu_defs table 287
 - tools.menu_items table 285
 - tools.menu table 285
 - tools.prompt_defs table 286
 - transfer database 135
 - transient event list 296
 - UNIX command line options 296
 - Windows command line options 298
 - trigger groups
 - altering (SQL) 202
 - creating (SQL) 202
 - creating and editing (nco_config) 84
 - deleting (nco_config) 85
 - dropping (SQL) 202

trigger types 201

triggers

altering (SQL) 206

creating (SQL) 203

creating and editing (nco_config) 85

database 201, 207

deduplication 220

deleting (nco_config) 90

dropping (SQL) 206

examples 220

signal 201, 205, 211

temporal 201, 210

variables 205

truncating

ObjectServer files (nco_config) 117

ObjectServer files (SQL) 145

removing from groups (SQL) 178

V

VARCHAR ObjectServer data type 137

views

creating (SQL) 140

dropping (SQL) 141

W

web browser

selecting for online help (nco_config) 54

Windows services 16

WRITE INTO 171

U

UNSIGNED ObjectServer data type 136

UNSIGNED64 ObjectServer data type 137

UPDATE 165

USE DATABASE 175

user defined signals 211, 217

creating (SQL) 217

creating and editing (nco_config) 112

deleting (nco_config) 115

dropping (SQL) 219

example (nco_config) 114

raising (SQL) 219

user groups 228

user.name property (nco_config) 48

user.password property (nco_config) 48

users

adding to groups (nco_config) 62, 65

altering (SQL) 178

creating (SQL) 177

creating and editing (nco_config) 63

deleting (nco_config) 66

dropping (SQL) 179

removing from groups (nco_config) 62, 65

Contact Information

Table 1: 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

Table 2: 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

Table 3: License Generation Team

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

