

Netcool/OMNIbus
Version 7 Release 3

Administration Guide



Netcool/OMNIbus
Version 7 Release 3

Administration Guide



Note

Before using this information and the product it supports, read the information in "Notices" on page 365.

This edition applies to version 7, release 3, modification 1 of IBM Tivoli Netcool/OMNIBus (product number 5724-S44) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 1994, 2011.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this publication vii

Intended audience	vii
What this publication contains	vii
Publications	viii
Accessibility	x
Tivoli technical training.	x
Support information.	x
Conventions used in this publication	x

Chapter 1. Configuring the ObjectServer 1

Alert processing in the ObjectServer	1
Using the ObjectServer properties and command-line options	1
ObjectServer properties and command-line options	3
Running the ObjectServer in secure mode	17
Client tool updates using IDUC	20
Specifying the IDUC update interval	20
Specifying the IDUC port.	20
Configuring the ObjectServer for multicultural support.	21
Data storage and checkpointing	22
Data storage using memstores	22
Introduction to checkpointing	23
nco_check_store checkpoint verification utility.	24
Changing the table_store memstore soft and hard limits	25
Sending alerts to the ObjectServer using nco_postmsg	26
nco_postmsg properties and command-line options	29
nco_postmsg examples and resulting INSERT statements.	32

Chapter 2. Configuring a proxy server 35

Starting the proxy server	35
Starting a proxy server by using process control	35
Starting a proxy server by using services (Windows).	36
Starting the proxy server manually	36
Proxy server properties and command-line options	36
Connecting to the proxy server	39
Running the proxy server in secure mode	39

Chapter 3. Configuring a firewall bridge server 43

A standard firewall bridge server configuration	44
A multiple firewall bridge server configuration	46
Firewall bridge server failover configuration	47
Starting the firewall bridge server	49
Starting a firewall bridge server by using process control	49
Starting a firewall bridge server by using Windows services	49
Starting the firewall bridge server manually	49

Firewall bridge server properties and command-line options.	50
Trusted hosts definition file	54
Firewall bridge server command language	55
SHOW PROPS and GET CONFIG	55
GET PROP	56
SHOW DATAFLOWS	56
SET LOG LEVEL TO	57
SHUTDOWN.	57

Chapter 4. Using Netcool/OMNIBus Administrator to configure ObjectServers. 59

Getting started with Netcool/OMNIBus Administrator	59
Considerations for multicultural support	59
Starting Netcool/OMNIBus Administrator	60
Connecting to an ObjectServer	62
Connecting to a process agent	64
Working with Tivoli Netcool/OMNIBus components	65
Secure sockets layer connections	66
Selecting ObjectServer objects to configure	67
Setting preferences in Netcool/OMNIBus Administrator	68
Exiting Netcool/OMNIBus Administrator	71
Managing authorization with users, groups, roles, and restriction filters	71
Configuring roles	71
Configuring groups.	76
Configuring users	79
Configuring restriction filters	84
Configuring menus, tools, and prompts	86
Customizing menus	86
Configuring tools	90
Configuring prompts	93
Configuring automations	97
Configuring triggers	97
Configuring procedures	107
Configuring signals	115
Configuring the visual appearance of the event list	118
Creating and editing conversions	118
Deleting conversions	118
Creating and editing event severity colors for Windows event lists	119
Creating and editing column visuals	120
Deleting column visuals	121
Creating and editing classes	121
Deleting classes	121
Configuring ObjectServer databases, files, properties, connections, and channels	122
Configuring databases	122
Viewing and changing ObjectServer properties	127
Configuring ObjectServer files.	128
Monitoring ObjectServer connections	130

Configuring channels.	130
Using the SQL interactive interface in GUI mode	131
Chapter 5. ObjectServer SQL.	135
SQL interactive interface.	135
Starting the SQL interactive interface	136
Running SQL commands in the SQL interactive interface	138
Running the SQL interactive interface in secure mode	141
Encrypting passwords in UNIX nco_sql scripts	141
Exiting the SQL interactive interface.	141
Creating, altering, and dropping ObjectServer objects.	142
Databases	142
Tables	144
Indexes	151
Views	153
Restriction filters	155
Files	156
Reserved words	158
SQL building blocks	160
Operators	160
Functions.	165
Expressions	171
Conditions	172
Querying and manipulating data using ObjectServer SQL	173
Inserting a new row of data into a table (INSERT command)	173
Updating the data in table columns (UPDATE command)	174
Deleting rows of data from a table (DELETE command)	175
Retrieving data from a table or view (SELECT command)	175
Logging information to ObjectServer files (WRITE INTO command)	179
Displaying details of columns in a table or view (DESCRIBE command)	180
Adding or updating service status data (SVC command)	180
Sending IDUC notifications to IDUC clients (IDUC FLUSH command)	181
Changing the default and current settings of the ObjectServer (ALTER SYSTEM command).	181
Setting the default database (SET DATABASE and USE DATABASE commands)	182
Verifying your SQL syntax (CHECK STATEMENT command)	183
Creating, modifying, and deleting users, groups, and roles	183
Creating a user (CREATE USER command)	184
Modifying the details of an existing user (ALTER USER command)	185
Deleting a user (DROP USER command)	186
Creating a group (CREATE GROUP command)	186
Modifying the details of an existing group (ALTER GROUP command)	186
Deleting a group (DROP GROUP command)	187
Creating a role (CREATE ROLE command)	187

Modifying the description of a role (ALTER ROLE command)	188
Using roles to assign permissions to users.	188
Deleting a role (DROP ROLE command)	194
Creating, running, and dropping procedures	195
SQL procedures	195
External procedures	203
Running procedures	205
Dropping procedures.	205
Configuring automation using triggers	205
Creating, modifying, and deleting trigger groups	206
Creating, modifying, and dropping triggers	207
Standard Tivoli Netcool/OMNIBus automations	230
Automation for service-affected events	235
Automation examples	236

Chapter 6. Configuring accelerated event notification. 239

Configuring a probe to flag events for acceleration	239
Configuring a gateway for accelerated event notification	240
Configuring the alerts.status table to receive the AEN flag.	241
Configuring channels to broadcast event data	241
Creating and editing channels.	241
Copying and pasting channels.	244
Deleting a channel.	245
Sending messages to channel recipients.	245
Disconnecting Accelerated Event Notification clients	245
Shutting down Accelerated Event Notification clients	246
Configuring triggers to support accelerated event notification	247

Chapter 7. Using process control to manage processes and external procedures 249

How process agents connect	249
Process control components	250
Process agents	250
Processes	250
Services	251
Process control utilities	251
Creating and starting a process control network system	252
Before you configure process control	252
Creating UNIX user groups for the process control system	253
Windows account requirements for the process control system	253
Configuring server communication information for process agents	254
Updating the default process control configuration file	254
Manually starting process agents	255
Automatically starting process agents on UNIX	261
Automatically starting process agents on Windows.	263

Managing your process control system configuration	263
Configuring and managing process control from the command line	263
Defining processes, services, and hosts for process control	264
Managing process control using the process control utilities	273
Using Netcool/OMNIbus Administrator to manage process control	281
Connecting to a process agent	281
Displaying and configuring status information for a process agent	283
Displaying the processes and services for a process agent	284
Configuring services for a process agent	285
Configuring processes	288
Copying and pasting a service or process between process agent hosts	293
Running an external action	294
Stopping a process agent	294
Using process control to run external procedures in automations	295

Chapter 8. Performance tuning 297

Tivoli Netcool/OMNIbus key performance indicators.	297
ObjectServer key performance indicators	298
Probe key performance indicators	301
Gateway key performance indicators	302
Best practices for performance tuning	303
Run the ObjectServer with profiling enabled	303
Collect statistical information about triggers	305
Review and revise your system architecture	307
Enable the stats_triggers trigger group	307
Review and revise your probe configuration files	308
Configure event flood detection	308
Manage the volume of information in the alerts.details table	308
Use a monitoring agent to monitor and manage Tivoli Netcool/OMNIbus resources	309
Review and amend your SQL queries, and create a selection of well-designed, efficient indexes	309
Track the performance trends at regular intervals	311
SQL query guidelines.	311
Optimization rules for SQL queries	311
Indexing guidelines	314
Example usage of indexes with SQL queries	315
Example usage of indexes with triggers or procedures	316
Best practices for creating triggers	317

Appendix A. ObjectServer tables . . . 321

Alerts tables.	321
alerts.status table	321
alerts.details table	334
alerts.journal table.	335

alerts.iduc_messages table	335
alerts.application_types table	336
master.class_membership table	336
Service tables	337
service.status table.	337
System catalog tables.	338
catalog.memstores table	338
catalog.databases table	338
catalog.tables table	338
catalog.base_tables table.	339
catalog.views table	340
catalog.files table	340
catalog.restrictions table	341
catalog.columns table.	341
catalog.primitive_signals table.	341
catalog.primitive_signal_parameters table	342
catalog.trigger_groups table	342
catalog.triggers table	342
catalog.database_triggers table.	343
catalog.signal_triggers table	344
catalog.temporal_triggers table	344
catalog.procedures table	344
catalog.sql_procedures table	344
catalog.external_procedures table.	345
catalog.procedure_parameters table	345
catalog.connections table	346
catalog.properties table	346
catalog.security_permissions table	346
catalog.profiles table	347
catalog.indexes table	348
Statistics tables	348
catalog.profiles table	348
master.stats table	349
catalog.trigger_stats table	350
catalog.channel_stats table	350
Client tool support tables	351
alerts.resolutions table	351
alerts.conversions table	352
alerts.col_visuals table	352
alerts.colors table	352
Desktop tools tables	353
tools.actions table	353
tools.action_access table	354
tools.menus table	355
tools.menu_items table	355
tools.prompt_defs table	355
tools.menu_defs table	356
Desktop ObjectServer tables	356
master.national table	356
master.servergroups table	357
Security tables for backward compatibility.	357
IDUC tables.	357
iduc_system.channel table	358
iduc_system.channel_interest table	358
iduc_system.channel_summary table	358
iduc_system.channel_summary_cols table	359
iduc_system.iduc_stats table	359
Service-affected events tables	359
precision.service_affecting_event table	359
precision.service_details table	360
precision.entity_service table	360

Appendix B. SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists	361
--	------------

Notices	365
Trademarks	367
Index	369

About this publication

Tivoli Netcool/OMNIBus is a service level management (SLM) system that delivers real-time, centralized monitoring of complex networks and IT domains.

The *IBM Tivoli Netcool/OMNIBus Administration Guide* provides detailed information about administrative tools, functions, and capabilities of Tivoli Netcool/OMNIBus. In addition, it is designed to be used as a reference guide to assist you in designing and configuring your environment.

Intended audience

This publication is intended for administrators who are responsible for configuring Tivoli Netcool/OMNIBus.

What this publication contains

This publication contains the following sections:

- Chapter 1, “Configuring the ObjectServer,” on page 1
Describes how to configure the ObjectServer, which is the central repository for data.
- Chapter 2, “Configuring a proxy server,” on page 35
Describes how to configure a proxy server to reduce the number of probe connections to an ObjectServer.
- Chapter 3, “Configuring a firewall bridge server,” on page 43
Describes how to configure a firewall bridge server to allow probes to connect to the ObjectServer from outside a secure network.
- Chapter 4, “Using Netcool/OMNIBus Administrator to configure ObjectServers,” on page 59
Describes how to use Netcool/OMNIBus Administrator to configure and manage ObjectServers.
- Chapter 5, “ObjectServer SQL,” on page 135
Describes the data structures of the ObjectServer and the syntax of ObjectServer SQL.
- Chapter 6, “Configuring accelerated event notification,” on page 239
Describes how to configure Tivoli Netcool/OMNIBus for accelerated event notification of events that could present a risk to the system.
- Chapter 7, “Using process control to manage processes and external procedures,” on page 249
Describes the components, configuration, and management utilities associated with the Tivoli Netcool/OMNIBus process control system. Also includes information about configuring and managing process control using both command utilities and Netcool/OMNIBus Administrator.
- Chapter 8, “Performance tuning,” on page 297
Describes how to monitor and fine tune Tivoli Netcool/OMNIBus performance.
- Appendix A, “ObjectServer tables,” on page 321
Contains ObjectServer database table information.

- Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361
Provides reference information about common SQL commands, variable expressions, and helper buttons that are used in tools, automations, and transient event lists.

Publications

This section lists publications in the Tivoli Netcool/OMNIBus library and related documents. The section also describes how to access Tivoli publications online and how to order Tivoli publications.

Your Tivoli Netcool/OMNIBus library

The following documents are available in the Tivoli Netcool/OMNIBus library:

- *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*, SC14-7604
Includes installation and upgrade procedures for Tivoli Netcool/OMNIBus, and describes how to configure security and component communications. The publication also includes examples of Tivoli Netcool/OMNIBus architectures and describes how to implement them.
- *IBM Tivoli Netcool/OMNIBus Administration Guide*, SC14-7605
Describes how to perform administrative tasks using the Tivoli Netcool/OMNIBus Administrator GUI, command-line tools, and process control. The publication also contains descriptions and examples of ObjectServer SQL syntax and automations.
- *IBM Tivoli Netcool/OMNIBus Web GUI Administration and User's Guide*, SC14-7606
Describes how to perform administrative and event visualization tasks using the Tivoli Netcool/OMNIBus Web GUI.
- *IBM Tivoli Netcool/OMNIBus User's Guide*, SC14-7607
Provides an overview of the desktop tools and describes the operator tasks related to event management using these tools.
- *IBM Tivoli Netcool/OMNIBus Probe and Gateway Guide*, SC14-7608
Contains introductory and reference information about probes and gateways, including probe rules file syntax and gateway commands.
- *IBM Tivoli Monitoring for Tivoli Netcool/OMNIBus Agent User's Guide*, SC14-7610
Describes how to install the health monitoring agent for Tivoli Netcool/OMNIBus and contains reference information about the agent.
- *IBM Tivoli Netcool/OMNIBus Event Integration Facility Reference*, SC14-7611
Describes how to develop event adapters that are tailored to your network environment and the specific needs of your enterprise. This publication also describes how to filter events at the source.
- *IBM Tivoli Netcool/OMNIBus Error Messages Guide*, SC14-7612
Describes system messages in Tivoli Netcool/OMNIBus and how to respond to those messages.
- *IBM Tivoli Netcool/OMNIBus Web GUI Administration API (WAAPI) User's Guide*, SC22-5403-00
Shows how to administer the Tivoli Netcool/OMNIBus Web GUI using the XML application programming interface named WAAPI.

Accessing terminology online

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available at the following Tivoli software library Web site:

<http://publib.boulder.ibm.com/tividd/glossary/tivoliglossarymst.htm>

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

<http://www.ibm.com/software/globalization/terminology>

Accessing publications online

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center Web site at:

<http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp>

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File > Print** window that allows Adobe Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at the following Web site:

<http://www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss>

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to the following Web site:
<http://www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss>
2. Select your country from the list and click **Go**. The Welcome to the IBM Publications Center page is displayed for your country.
3. On the left side of the page, click **About this site** to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully.

With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate some features of the graphical user interface.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site:

<http://www.ibm.com/software/tivoli/education>

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online

Go to the IBM Software Support site at <http://www.ibm.com/software/support/probsub.html> and follow the instructions.

IBM Support Assistant

The IBM Support Assistant (ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA provides quick access to support-related information and serviceability tools for problem determination. To install the ISA software, go to <http://www.ibm.com/software/support/isa>.

Conventions used in this publication

This publication uses several conventions for special terms and actions and operating system-dependent commands and paths.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:** and **Operating system considerations:**)
- Keywords and parameters in text

Italic

- Citations (examples: titles of publications, diskettes, and CDs)
- Words defined in text (example: a nonswitched line is called a *point-to-point* line)

- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data
- Variables and values you must provide: ... where *myname* represents....

Monospace

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This publication uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace *\$variable* with *%variable%* for environment variables, and replace each forward slash (/) with a backslash (\) in directory paths. For example, on UNIX systems, the *\$NCHOME* environment variable specifies the path of the Netcool® home directory. On Windows systems, the *%NCHOME%* environment variable specifies the path of the Netcool home directory. The names of environment variables are not always the same in the Windows and UNIX environments. For example, *%TEMP%* in Windows environments is equivalent to *\$TMPDIR* in UNIX environments.

If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Operating system-specific directory names

Where Tivoli Netcool/OMNIBus files are identified as located within an *arch* directory under *NCHOME*, *arch* is a variable that represents your operating system directory, as shown in the following table.

Table 1. Directory names for the *arch* variable

Directory name represented by <i>arch</i>	Operating system
<i>aix5</i>	AIX® systems
<i>hpux11</i>	HP-UX PA-RISC-based systems
<i>hpux11hpia</i>	HP-UX Integrity-based systems
<i>linux2x86</i>	Red Hat Linux and SUSE systems
<i>linux2s390</i>	Linux for System z®
<i>solaris2</i>	Solaris systems
<i>win32</i>	Windows systems

Chapter 1. Configuring the ObjectServer

At least one ObjectServer is required to store and manage alert information in a Tivoli Netcool/OMNIBus installation. You can configure the ObjectServer by using its properties and command-line options, and can also configure the ObjectServer to only accept secure connections. You can additionally configure Insert, Delete, Update, or Control (IDUC) support, multicultural support, and automated failover and fallback.

Alert processing in the ObjectServer

Alerts are stored as rows or entries in the ObjectServer `alerts.status` table. Each alert has an Identifier field that uniquely identifies the problem source. The identifier is generated by the probe according to the specification in the probe rules file.

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 when the problem first occurred.

If an entry with the same identifier is found, deduplication occurs. The ObjectServer acknowledges the occurrence of the duplicate entry by adding a count to the existing entry. By default, the ObjectServer updates the `LastOccurrence` field of the existing entry with the time of the new alert, and increments the `Tally` field.

Note: The V7.0, or later, Deduplication trigger takes precedence over the Update on Deduplication setting in the probe if the Identifier field is explicitly referenced by the trigger.

The ObjectServer can also respond automatically to specified alerts by using automation.

You can export alert information to other applications through a gateway, and can use a bidirectional ObjectServer gateway to provide failover support to another ObjectServer.

Using the ObjectServer properties and command-line options

The ObjectServer 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.

About the ObjectServer properties file

The default location of the properties file is `$NCHOME/omnibus/etc/servername.props`. In the ObjectServer properties file, a property and its corresponding value are separated by a colon (:). String values are surrounded by single, straight quotation marks; for example:

Name: 'NCOMS'

Each ObjectServer property has a default value. In an unedited properties file, all properties are set to the default values. At the top of the file, a list of the properties with their default values, data types, and descriptions is provided, to act as a reference. These properties are commented out with a number sign (#) at the beginning of the line. Below the list of commented-out properties, another list of properties with their default values is provided for editing purposes, if required.

Note: If you are running the ObjectServer in UTF-8 encoding on Windows, you must also save the ObjectServer properties file in UTF-8 encoding.

Specifying ObjectServer properties

You can change the settings for ObjectServer properties in one of the following ways:

- Edit the properties file and change the value of the required properties. Change the entries below the commented-out list of properties. When you make changes to the properties in the properties file, the changes do not take effect until you restart the ObjectServer.
- Run the ALTER SYSTEM SET command from the SQL interactive interface.
- From the Netcool/OMNIBus Administrator interface, use the **System** menu button and the **Properties** option to display the properties and edit their values.

When you use the ALTER SYSTEM SET command or Netcool/OMNIBus Administrator to change the ObjectServer properties, changes to some of the properties do not take effect until you restart the ObjectServer. For information about viewing the properties that require an ObjectServer restart, see the list of **Tips** that follows.

Whenever you change the ObjectServer property values by using the ALTER SYSTEM SET command or Netcool/OMNIBus Administrator, a list of properties is added at the bottom of the properties file to reflect your changes. As a result, multiple entries for a property can exist within a file. The last entry for a property takes precedence over any earlier entries.

Tips for viewing information about ObjectServer properties:

- 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 following query in the SQL interactive interface:

```
select PropName from catalog.properties where IsModifiable=FALSE;
```
- Changes to some ObjectServer properties do not take effect until you restart the ObjectServer. To retrieve a list of these properties, enter the following query:

```
select PropName from catalog.properties where IsImmediate=FALSE;
```
- You can also view information about the ObjectServer properties from the Netcool/OMNIBus Administrator interface.

Specifying ObjectServer command-line options

When running the ObjectServer by using the **nco_objserv** command, you can specify a set of command-line options. You can override both the default value and the properties file value by changing the property value from the command line.

The 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 to which you are setting the option. Not every option requires you to specify a value.

You can add command-line options to **nco_objserv** commands in the process agent configuration file.

Related concepts

“Retrieving data from a table or view (SELECT command)” on page 175

Related tasks

“Viewing and changing ObjectServer properties” on page 127

“Defining processes, services, and hosts for process control” on page 264

Related reference

“Changing the default and current settings of the ObjectServer (ALTER SYSTEM command)” on page 181

“catalog.properties table” on page 346

ObjectServer properties and command-line options

Use the ObjectServer properties or command-line options to configure settings for the ObjectServer. To avoid errors, add as many properties as possible to the properties file rather than using the command-line options.

Tip: You can encrypt string values in a properties file by using property value encryption.

The ObjectServer properties and command-line options are described in the following table.

Table 2. ObjectServer properties and command-line options

Property	Command-line option	Description
ActingPrimary TRUE FALSE See the Description column.	N/A	<p>This property is used in a failover configuration, where a primary and backup ObjectServer are connected by a bidirectional ObjectServer gateway, which is used to replicate the event data between the two ObjectServers. The property is updated by automations only, and must <i>not</i> be manually updated within the properties file.</p> <p>ActingPrimary is set to TRUE in the backup ObjectServer for the period during which the backup ObjectServer acts as the primary server. Otherwise, this property is set to FALSE in the backup ObjectServer.</p> <p>ActingPrimary is always set to TRUE in the primary ObjectServer,</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
AlertSecurityModel <i>integer</i>	-alertsecuritymodel <i>integer</i>	<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 Normal group can modify a row that is assigned to themselves or the nobody user. A member of the Administrator group can modify a row that is assigned to themselves, the nobody user, or a member of the Normal group. <p>If the AlertSecurityModel property is enabled (1), only users in the group that owns the row can modify the row. In this case, a member of the Normal or Administrator group can modify a row that is assigned to a group of which they are a member.</p> <p>A member of the System group can always modify any row.</p> <p>For more information about system and default groups, see the <i>IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide</i>.</p>
AllowConnections TRUE FALSE	-disallowconnections	Specifies whether non-root users can connect to the ObjectServer. If FALSE, no new connections to the ObjectServer are allowed. The default is TRUE.
AllowISQL TRUE FALSE	-disallowisql	<p>Specifies whether connections to the ObjectServer are allowed using the SQL interactive interface. If FALSE, no user can connect using nco_sql. The default is TRUE.</p> <p>If TRUE, this can be enabled for each user using Netcool/OMNIbus Administrator.</p>
AllowISQLWrite TRUE FALSE	-disallowisqlwrite	<p>Specifies whether modifications to the ObjectServer are allowed using the SQL interactive interface. If FALSE, no user can modify the ObjectServer using nco_sql. The default is TRUE.</p> <p>If TRUE, this can be enabled for each user using Netcool/OMNIbus Administrator.</p>
AllowTimedRefresh TRUE FALSE	-allowtimedrefresh TRUE FALSE	<p>This property determines whether the user can enable timed refresh in the Refresh tab of the Event List Preferences 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.</p> <p>If FALSE, the timed refresh check box is grayed out in the Refresh tab of the Event List Preferences window and timed refresh is disabled.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
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.
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.
ClientHeartbeatDisable TRUE FALSE	-clienthbdisable TRUE FALSE	Disables the client heartbeating system if set to TRUE. This causes a connected client to time out if the ObjectServer is busy - for example, during a gateway resynchronization or an automation. The default FALSE setting enables heartbeating, and prevents invalid and unnecessary client timeouts. If the ObjectServer is active but busy, this setting causes the ObjectServer to send a pop-up message to a connected client, with details of the type of processing in progress.
ClientHeartbeatRate <i>integer</i>	-clienthbrate <i>integer</i>	Sets the rate in seconds of a client heartbeat. This rate defines how long a client should wait for a response from the ObjectServer before timing out. The default value is 10.

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
ConfigCryptoAlg <i>string</i>	N/A	<p>Specifies the cryptographic algorithm to use for decrypting string values (including passwords) that were encrypted with the nco_aes_crypt utility and then stored in the properties file. Set the <i>string</i> value as follows:</p> <ul style="list-style-type: none"> When in FIPS 140–2 mode, use AES_FIPS. When in non-FIPS 140–2 mode, you can use either AES_FIPS or AES. Use AES only if you need to maintain compatibility with passwords that were encrypted by using the tools provided in versions earlier than Tivoli Netcool/OMNIbus V7.2.1. <p>The value that you specify must be identical to that used when you ran nco_aes_crypt with the -c setting, to encrypt the string values.</p> <p>Use this property in conjunction with the ConfigKeyFile property.</p>
ConfigKeyFile <i>string</i>	N/A	<p>Specifies the path and name of the key file that contains the key used to decrypt encrypted string values (including passwords) in the properties file.</p> <p>The key is used at run time to decrypt string values that were encrypted with the nco_aes_crypt utility. The key file that you specify must be identical to the file used to encrypt the string values when you ran nco_aes_crypt with the -k setting.</p> <p>Use this property in conjunction with the ConfigCryptoAlg property.</p>
Connections <i>integer</i>	-connections <i>integer</i>	<p>Sets the maximum number of available connections for desktop clients, probes, and gateways.</p> <p>The maximum value is 1024. The default value is 30. Up to two connections can be used by the system.</p>
DeleteLogFile <i>string</i>	-DELETES <i>string</i>	<p>The path and name of the delete log file, where all delete commands are recorded if delete logging is enabled.</p> <p>By default, deletes are logged to the file <code>\$NCHOME/omnibus/log/servername_deletes_file.logn</code>.</p>
DeleteLogging TRUE FALSE	-DELETE_LOGGING TRUE FALSE	<p>When TRUE, delete logging is enabled. The default is FALSE.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
DeleteLogLevel <i>integer</i>	-DELETES_LEVEL <i>integer</i>	<p>The log level determines how much information is sent to the delete log file. Possible settings are:</p> <ul style="list-style-type: none"> • <0: No logging • 0: Client type (application ID; for example, ctisql for nco_sql) and SQL run. This is the default log level. • 1: Time, user ID, client type, and SQL run.
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 can be larger than the specified size.</p> <p>The default log file size is 1024 KB.</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 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
GWDEDUPlication <i>integer</i>	<code>-gwdeduplication integer</code>	<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 number in 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 number in 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>	<code>-granularity integer</code>	<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>
N/A	<code>-help</code>	<p>Displays help on the command-line options and exits.</p>
Iduc.ListeningHostname <i>string</i>	<code>-listeninghostname string</code>	<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 might not be the name by which it is referred to on the network.</p> <p>For example, a computer named sfo might actually be identified on the network as sfo.bigcorp.org. To allow clients to connect to the ObjectServer on sfo, set this property to sfo.bigcorp.org.</p> <p>The default is the name of the local computer, as reported by the hostname command.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
Iduc.ListeningPort <i>integer</i>	<code>-listeningport <i>integer</i></code>	<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>
LogFilePoolSize <i>integer</i>	<code>-logfilepoolsize <i>integer</i></code>	<p>Specifies the number of log files to use if the logging system is writing to a pool of files. This property works only when the LogFileUsePool property is set to TRUE. The pool size can range from 2 to 99.</p> <p>The default is 10.</p> <p>This option is supported only on Windows operating systems.</p>
LogFileUsePool TRUE FALSE	<code>-logfileusepool</code>	<p>Specifies whether to use a pool of log files for logging messages.</p> <p>If set to TRUE, the logging system opens the number of files specified for the pool at startup, and keeps them open for the duration of its run. (You define the number of files in the pool using the LogFilePoolSize property.) When a file in the pool reaches its maximum size (as specified by the MaxLogFileSize property), the logging system writes to the next file. When all the files in the pool are at maximum size, the logging system truncates the first file in the pool and starts writing to it again. Files in the pool are named using the format <code>servername.log_ID</code>, where <i>ID</i> is a two-digit number starting from 01, to the maximum number specified for the LogFilePoolSize property. When the logging system starts to use a file pool, the system writes to the lowest-available file number, regardless of which file it was writing to when it last ran.</p> <p>The default is FALSE. When set to FALSE, the default <code>servername.log</code> file is renamed <code>servername.log_OLD</code> and a new log file is started when the maximum size is reached. If the file cannot be renamed, for example, because of a read lock on the <code>_OLD</code> file, and the LogFileUseStdErr property is set to FALSE, the logging system automatically starts using a pool of log files.</p> <p>This option is supported only on Windows operating systems.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
LogFileUseStdErr TRUE FALSE	-logfileusestderr	<p>Specifies whether to use stderr as an output stream for logging messages.</p> <p>The default is TRUE. If this setting is TRUE and the ObjectServer is run from the command line, messages are logged to the console if the ObjectServer cannot write to the default log file. If this setting is TRUE and the ObjectServer is running as a Windows service, messages are written to a file named %NCHOME%\omnibus\log\ncs_objserv*.err if the ObjectServer cannot write to the default log file.</p> <p>If set to FALSE and the ObjectServer cannot write to the default log file, messages are written to a pool of log files, as set by the LogFileUsePool property.</p> <p>Note: The LogFileUsePool property setting takes precedence over the LogFileUseStdErr setting.</p> <p>This option is supported only on Windows operating systems.</p>
MaxLogFileSize <i>integer</i>	-maxlogfilesize <i>integer</i>	<p>Specifies the maximum size the log file can grow to, in KB. The default is 1024 KB.</p> <p>When it reaches the size specified, the <i>servername.log</i> file is renamed <i>servername.log_OLD</i> and a new log file is started. When the new file reaches the maximum size, it is renamed and the process starts again.</p>
MessageLevel <i>string</i>	-messagelevel <i>string</i>	<p>Specifies the message logging level. Possible values are: debug, info, warn, error, and fatal. The default level is warn.</p> <p>Messages that are logged at each level are as follows:</p> <ul style="list-style-type: none"> 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 <p>Tip: The value of <i>string</i> can be in uppercase, lowercase, or mixed case.</p>
MessageLog <i>string</i>	-messagelog <i>string</i>	<p>Specifies the path to which messages are logged. The default is \$NCHOME/omnibus/log/NCOMS.log.</p> <p>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 %NCHOME%\omnibus\log\ncs_objserv*.err.</p> <p>Note: The ObjectServer must be running as a service; otherwise, it cannot write errors to a file.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
N/A	-name <i>string</i>	Sets the ObjectServer name, which must be unique. This is the name that is configured in the Server Editor. The default is NCOMS.
OldTimeStamp TRUE FALSE	-oldtimestamp TRUE FALSE	<p>Specifies the timestamp format to use in the log file.</p> <p>Set the value to TRUE to display the timestamp format that is used in Tivoli Netcool/OMNIBus V7.2.1, or earlier. For example: dd/MM/YYYY hh:mm:ss AM or dd/MM/YYYY hh:mm:ss PM when the locale is set to en_GB on a Solaris 9 computer.</p> <p>Set the value to FALSE to display the ISO 8601 format in the log file. For example: YYYY-MM-DDThh:mm:ss, where T separates the date and time, and hh is in 24-hour clock. The default is FALSE.</p>
PA.Name <i>string</i>	-pa <i>string</i>	Sets the process control agent name. This name must consist of 29 or fewer uppercase letters and cannot begin with an integer. 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 NCO_PA.
PA.Password <i>string</i>	-papassword <i>string</i>	<p>Specifies the password for the user connecting to a process control agent to run external procedures in automations. The default is ''.</p> <p>Note: If running external procedures, the process control daemon must be able to authenticate the user. Therefore, a valid combination, which can be authenticated by the daemon, must be specified in the <i>string</i> values for PA.Username and PA.Password properties. Otherwise, the connection to the process control agent and the running of the external procedure will fail.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
PA.Username <i>string</i>	-pausername <i>string</i>	<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 root.</p> <p>On Windows, specify the user name of a valid local account, domain account, or UPN account.</p> <p>On UNIX, any user who requires access to the process control system must be a member of a UNIX user group (default ncoadmin) that you identify as an administrative group for this purpose. Note that if using Pluggable Authentication Modules (PAM) for authentication, users do not have to be a member of a UNIX user group such as ncoadmin, to gain access to the process control system.</p>
PasswordEncryption <i>string</i>	-pwdenc <i>string</i>	<p>Defines the encryption scheme that is used to encrypt user passwords that are stored in the ObjectServer. The values are:</p> <ul style="list-style-type: none"> • DES: Data Encryption Standard encryption. Only the first eight characters of a DES-encrypted password are significant. • AES: Advanced Encryption Standard (AES128) encryption. Only the first 16 characters of an AES128-encrypted password are significant. When in FIPS 140-2 mode, the AES option is mandated by the system. <p>The default is DES for non FIPS 140-2 installations.</p> <p>The default is AES for FIPS 140-2 installations.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
PasswordFormat <i>string</i>	<code>-pwdformat</code> <i>string</i>	<p>Defines the requirements for user passwords, and works only when the RestrictPasswords property is set to TRUE.</p> <p>You must specify the <i>string</i> value of the PasswordFormat property (or <code>-pwdformat</code>) as a colon-separated set of integer values, where each value defines a password requirement. The format is: <i>min_len:alpha_num:digit_num:punct_num</i></p> <p>Where:</p> <ul style="list-style-type: none"> • <i>min_len</i> is the password length. • <i>alpha_num</i> is the minimum number of alphabetic characters. • <i>digit_num</i> is the minimum number of numeric characters. • <i>punct_num</i> is the minimum number of punctuation characters. <p>The default value of 8:1:1:0 indicates that a password must be at least 8 characters long, and contain at least 1 alphabetic character and at least one numeric character. The password need not contain any punctuation characters.</p>
ProfileStatsInterval <i>integer</i>	<code>-profilestatsinterval</code> <i>integer</i>	Specifies the interval in seconds at which the profiler writes information to the profile log file. The default is 60 seconds.
Profile TRUE FALSE	<code>-profile</code>	<p>Controls ObjectServer profiling. If TRUE, the amount of time it takes for clients to run SQL is logged to the catalog.profiles table. The default is TRUE.</p> <p>The profile statistics are also logged to the file <code>\$NCHOME/omnibus/log/servername_profiler_report.logn</code> every profile statistics interval.</p>
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	<code>-propsfile</code> <i>string</i>	Sets the ObjectServer properties file name. The default name is <code>servername.props</code> , where the <i>servername</i> is defined by the <code>-name</code> option.
RegexLibrary <i>string</i>	<code>-regexlib</code> <i>string</i>	<p>Defines which regular expression library to use for the ObjectServer. Possible values are: NETC00L and TRE.</p> <p>The default value of NETC00L is useful for single-byte character processing and is provides optimal system performance.</p> <p>To enable the use of the extended regular expression syntax on single-byte and multi-byte character languages, set the value to TRE. This setting results in decreased system performance.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
RestrictionFiltersAND TRUE FALSE	-restrictfiltersand	Controls how user restriction filters and group restriction filters are concatenated. This property is set for the system and not per user, and changes to the property setting come into effect only after you restart the ObjectServer. The values for this property are as follows: <ul style="list-style-type: none"> • TRUE: All restriction filters are combined using the AND operator. The default is TRUE. Example: user_rf AND group1 AND group2 • FALSE: All restriction filters are combined using the OR operator. Example: user_rf OR group1 OR group2
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 TRUE.
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 run. The only modifications to ObjectServer data that can be made are inserts into the alerts.status and alerts.details tables. The default is FALSE. If FALSE, connections from a proxy server can run any ObjectServer SQL commands.
Sec.AuditLevel <i>string</i>	-secauditlevel <i>string</i>	Specifies the level of security auditing performed. Possible values are debug, info, warn, and error. The default is warn. The debug and info levels generate messages for authentication successes and failures, while warn and error levels generate messages for authentication failures only.
Sec.AuditLog <i>string</i>	-secauditlog <i>string</i>	Specifies the file to which audit information is written. The default is \$NCHOME/omnibus/log/ <i>servername</i> /audit.log.

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
Sec.ExternalAuthentication <i>string</i>	<code>-authenticate <i>string</i></code>	<p>Controls which authentication module is used to authenticate users. The values are:</p> <ul style="list-style-type: none"> • none: Use this value to perform user authentication against the user names and passwords that are stored in the ObjectServer. This setting disables external authentication. • LDAP: Use this value to externally authenticate users whose credentials are stored in a Lightweight Directory Access Protocol (LDAP) repository. • PAM: Use this value to authenticate users by using the supplied Tivoli Netcool/OMNIBus PAM, or to externally authenticate users by using a third-party PAM configuration (such as the operating system, or a PAM configuration that is set up to use LDAP credentials). <p>On UNIX and Linux, the default is PAM. On Windows, the default is none.</p>
SecureMode TRUE FALSE	<code>-secure</code>	<p>Sets the security mode of the ObjectServer. If TRUE, the ObjectServer authenticates probe, gateway, and proxy server connection requests with a user name and password. Other client connection requests are always authenticated with a user name and password.</p>
Store.LocalizedSort TRUE FALSE	<code>-storelocalesort</code>	<p>Defines whether localized sorting is enabled or disabled.</p> <p>The default is FALSE, which disables localized sorting in favor of standard C library (libc) string comparisons. For example, Å will be treated as a variant of A and the two characters will sort near each other. Use this default setting for optimal system performance.</p> <p>Set the value to TRUE to enable localized sorting. For example, in a Danish locale, Å will be treated as a separate letter that sorts just after Z. Note that specifying this setting results in decreased system performance.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
Store.LocalizedSortCaseSensitive <i>string</i>	-storecasesort <i>string</i>	<p>Controls case sensitivity in localized sorting. This property is applicable only when Store.LocalizedSort is TRUE.</p> <p>The values are:</p> <ul style="list-style-type: none"> • OFF: Sorting is case-insensitive, and uppercase and lowercase letters are ordered in accordance with their tertiary sort weights. • UPPERFIRST: Uppercase characters are sorted before lowercase characters. For example, Account is sorted before account. • LOWERFIRST: Lowercase characters are sorted before uppercase characters. For example, account is sorted before Account. <p>The default is OFF.</p>
UniqueLog TRUE FALSE	-uniquelog	<p>If TRUE, the log file is uniquely named by appending the process ID of the ObjectServer to the default log file name. For example, if the NCOMS ObjectServer is running as process 1234, the log file is named NCHOME/omnibus/log/NCOMS.1234.log. The default is FALSE.</p> <p>If the MessageLog property is set to stderr or stdout, the UniqueLog property is ignored.</p>
N/A	<div>Windows</div> -utf8enabled TRUE FALSE	<p>Controls the encoding of data that is passed into, or generated by, this application on Windows.</p> <p>Set the value of -utf8enabled to TRUE to run the application in the UTF-8 encoding. The default is FALSE, which causes the default system code page to be used.</p> <p>Important: Although a UTF8Enabled property is available, an attempt to enable UTF-8 encoding by setting this property to TRUE will have no effect. To run in a UTF-8 encoding on Windows, you must always use the -utf8enabled command-line option.</p> <p>Note: When running in UTF-8 encoding, do not set the OldTimeStamp property to TRUE.</p>

Table 2. ObjectServer properties and command-line options (continued)

Property	Command-line option	Description
WTPasswordCheck <i>string</i>	<code>-wtpasswordcheck string</code>	<p>Controls how passwords are checked by the ObjectServer when received from clients.</p> <p>This property is useful when ObjectServer users are externally authenticated. Possible values for the property are:</p> <ul style="list-style-type: none"> • plain: Passwords are assumed to be in plain text. (Applicable to Netcool/Webtop 1.3 clients only.) • encrypted: Passwords are assumed to be in an encrypted format. (Applicable to Netcool/Webtop 1.3 clients only.) • default: Passwords are initially assumed to be in plain text. If the ObjectServer check fails, passwords are assumed to be in an encrypted format. (Applicable to Netcool/Webtop 1.3 clients only.) • allplain: Passwords are assumed to be in plain text. If the login fails with the plain text password, no second attempt is made. (Applicable to clients other than Netcool/Webtop 1.3.) • allencrypted: Passwords are assumed to be in an encrypted format and are decrypted before authentication. If the login fails with the decrypted password, no second attempt is made. (Applicable to clients other than Netcool/Webtop 1.3.) <p>If an encrypted password is received by the ObjectServer and the property value is set to allplain, an error message is written to the log file. This error message can be ignored.</p>
N/A	<code>-version</code>	Displays version information for the ObjectServer and exits.

Note: The **nco_objserv** command includes advanced properties that must only be used under direction from IBM Software Support. These properties are not documented.

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 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.

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

When connecting to a secure ObjectServer:

- Each probe or proxy server that makes a connection must have the **AuthUserName** and **AuthPassword** properties specified in its properties file.
- Each unidirectional gateway that uses a properties file must have values specified for the **Gate.Writer.Username**, **Gate.Writer.Password**, **Gate.Reader.Username**, and **Gate.Reader.Password** properties. Each bidirectional gateway that uses a properties file must have values specified for the **Gate.ObjectServerA.Username**, **Gate.ObjectServerA.Password**, **Gate.ObjectServerB.Username**, and **Gate.ObjectServerB.Password** properties. Each gateway that uses a configuration file must have values specified for 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**, **Gate.Writer.Username**, **Gate.Reader.Username**, **Gate.ObjectServerA.Username**, or **Gate.ObjectServerB.Username** property, or the AUTH_USER command.

Password encryption details for running in FIPS 140–2 mode and non-FIPS 140–2 mode are described in the following table.

Table 3. Password encryption in FIPS 140–2 mode and non-FIPS 140–2 mode

Mode	Action
FIPS 140–2 mode	<p>When in FIPS 140–2 mode, passwords can either be specified in plain text or in encrypted format. You can encrypt passwords by using property value encryption, as follows:</p> <ol style="list-style-type: none"> 1. If you do not yet have a key for encrypting the password, create one by running the nco_keygen utility, which is located in \$NCHOME/omnibus/bin. 2. Run the nco_aes_crypt utility to encrypt the password with the key that was generated by the nco_keygen utility. The nco_aes_crypt utility is also located in \$NCHOME/omnibus/bin. Note that you must specify AES_FIPS as the algorithm to use for encrypting the password. 3. Open the properties file to which you want to add the encrypted password and specify this encrypted output for the AuthPassword setting. <p>Note: You must also set the ConfigKeyFile property to the key file that you specified when running nco_aes_crypt, and set the ConfigCryptoAlg property to the encryption algorithm used.</p>

Table 3. Password encryption in FIPS 140–2 mode and non-FIPS 140–2 mode (continued)

Mode	Action
Non-FIPS 140–2 mode	<p>When in non-FIPS 140–2 mode, passwords can either be specified in plain text or in encrypted format. However, the client always transmits encrypted login information irrespective of the password encryption that is used in the properties file. You can encrypt passwords by using the nco_g_crypt utility or by using property value encryption, as follows:</p> <ul style="list-style-type: none"> To encrypt a password by using the nco_g_crypt utility, run the command as follows: <code>\$NCHOME/omnibus/bin/nco_g_crypt plaintext_password</code> In this command, <i>plaintext_password</i> represents the unencrypted form of the password. The nco_g_crypt utility takes the unencrypted password and displays an encrypted version. Open the properties file to which you want to add the encrypted password and specify this encrypted output for the AuthPassword setting. To encrypt a password by using property value encryption, you require a key that is generated with the nco_keygen utility. You can then run nco_aes_crypt to encrypt the password with the key. Note that you can specify either AES_FIPS or AES as the algorithm for encrypting the password. Use AES only if you need to maintain compatibility with passwords that were encrypted using the tools provided in versions earlier than Tivoli Netcool/OMNIBus V7.2.1. Open the file to which you want to add the encrypted password and specify this encrypted output for the AuthPassword setting. Note: You must also set the ConfigKeyFile property to the key file that you specified when running nco_aes_crypt, and set the ConfigCryptoAlg property to the encryption algorithm used.

A password encrypted with **nco_g_crypt** 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.

Attention: Passwords encrypted with **nco_g_crypt** can be used in the same way as unencrypted passwords to access the ObjectServer. Therefore, you must set appropriate permissions on any files containing encrypted passwords to prevent unauthorized access. Alternatively, passwords that have been encrypted with **nco_g_crypt** must be further encrypted with **nco_aes_crypt**, and permissions on the key file must be set appropriately.

For further information about using property value encryption, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*. For information on running probes and gateways in secure mode, see the *IBM Tivoli Netcool/OMNIBus Probe and Gateway Guide*.

Related reference

“Running the proxy server in secure mode” on page 39

“Proxy server properties and command-line options” on page 36

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 whenever 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.

Example entries for ObjectServers named NCOMS and DENCO are as follows:

```
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.

Configuring the ObjectServer for multicultural support

Tivoli Netcool/OMNIBus supports a variety of single-byte and multi-byte character encodings for use in different locales.

The following ObjectServer properties are relevant for multi-byte character processing:

- **Store.LocalizedSort:** Use this property to enable localized sorting. Localized sorting is disabled by default for optimal system performance.
- **Store.LocalizedSortCaseSensitive:** Use this property to control case sensitivity in localized sorting.
- **RegexpLibrary:** Use this property to enable use of the POSIX 1003.2 extended regular expression library (TRE). The standard NETCOOL regular expression library is enabled by default for optimal system performance.

You can set these properties in either of the following ways:

- Set the properties in the ObjectServer properties file `$NCHOME/omnibus/etc/servername.props`, where *servername* is the name specified for the ObjectServer during its creation.
- Change the settings from the command line when starting the ObjectServer with the **nco_objserv** command. The command-line option for the **Store.LocalizedSort** property is `-storelocalesort`. The command-line option for the **Store.LocalizedSortCaseSensitive** property is `-storecasesort`. The command-line option for the **RegexpLibrary** property is `-regexplib`.

Note: If your user name and password are being verified against an external authentication source, you must check whether this source also supports multi-byte characters. If multi-byte characters are not supported, you must specify user names and passwords using ASCII characters.

The names of the following ObjectServer objects are restricted to ASCII characters, starting with a letter or underscore, and continuing with letters, digits, and underscores:

- Memstores
- Databases
- Tables
- Columns
- Restriction filters
- Privileges
- Trigger groups
- Triggers
- Signals
- Procedures
- Parameters
- Variables
- Properties
- File objects

File object names are restricted to the specified ASCII characters, whereas file path names are restricted to characters that are supported by the operating

system. The character encoding that is used to create the files is the encoding that is used by the ObjectServer; this encoding might differ from that of the client.

For further information about multicultural support for Tivoli Netcool/OMNIBus, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*.

Related reference

“ObjectServer properties and command-line options” on page 3

Data storage and checkpointing

ObjectServer data is stored in memory for high-speed access. The ObjectServer supports data persistence by 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.

Data storage using memstores

Memstores are containers that are maintained by the ObjectServer and hold ObjectServer tables and data in memory.

The ObjectServer uses the memstores described in the following table.

Table 4. 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 alerts.status 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.

You can change the size of the table_store memstore by using the **nco_store_resize** utility.

Related reference

“Changing the table_store memstore soft and hard limits” on page 25

Introduction to checkpointing

The ObjectServer supports data persistence by 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 creation

Checkpoint files are generated for each persistent memstore.

Tip: Only persistent memstores are checkpointed.

Checkpoint files are named as follows:

- `$NCHOME/omnibus/db/servername/storename.chk0`
- `$NCHOME/omnibus/db/servername/storename.chk1`

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

- `$NCHOME/omnibus/db/servername/storename.log0`
- `$NCHOME/omnibus/db/servername/storename.log1`

The checkpoint process writes alternately to the `.chk0` and `.chk1` files. If one file is 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 run), a `.tab` file is created for each persistent memstore. This file is named `storename.tab`.

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

`$NCHOME/omnibus/db/NCOMS/MASTER_STORE.tab`

The format of these files is specific to the hardware and operating system 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.

nco_check_store checkpoint verification utility

The **nco_check_store** utility verifies that existing checkpoint files are valid. It is intended to be used by automations and can be used only to check ObjectServer stores that are not currently in use.

The **nco_check_store** utility reports the validity of the checkpoint files with the following return codes:

- 0: Success. The checkpoint files are valid.
- 1: Failure. The checkpoint files are not valid and must 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.

Note: Do not run **nco_check_store** when the ObjectServer is running.

The command-line options for **nco_check_store** are described in the following table.

Table 5. Checkpoint verification utility command-line options

Command-line option	Description
-help	Displays help on the command-line options and exits.
-memstoredatadirectory <i>string</i>	Specifies the path to the ObjectServer database. The default is \$NCHOME/omnibus/db.
-messagelevel <i>string</i>	<p>Specifies the message logging level. Possible values are: debug, info, warn, error, and fatal. The default level is error.</p> <p>Messages that are logged at each level are as follows:</p> <ul style="list-style-type: none">• 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 <p>Tip: The value of <i>string</i> can be in uppercase, lowercase, or mixed case.</p>
-messagelog <i>string</i>	Specifies the path to which messages are logged. The default is stderr.
-server <i>string</i>	Specifies the name of the ObjectServer database to verify. The default is NCOMS.
-version	Displays version information on the checkpoint verification utility and exits.

Example usage

To use **nco_check_store** from within an automation to check that backup files created by the ObjectServer are valid, set the **-memstoredata** directory command-line option to the directory that contains the backup.

Changing the table_store memstore soft and hard limits

If the alerts database becomes exceptionally large, you can use a command to change the soft and hard limits.

If the alerts database becomes exceptionally large, the following message is displayed:

```
Region soft limit exceeded
```

This means that the table_store memstore, where alert table data is stored, has reached its maximum size. You can use the ALTER MEMSTORE command to change the soft limit. For example, in **nco_sql**, enter:

```
ALTER MEMSTORE table_store SET SOFT LIMIT 500 M;
```

The soft limit cannot be set to a larger size than the hard limit, which you can change using the **nco_store_resize** utility. This utility enables you to change the hard limit for the table_store memstore for the specified ObjectServer.

Note: Before you run the **nco_store_resize** utility, you must back up the ObjectServer by using the ALTER SYSTEM BACKUP command, and then shut down the ObjectServer.

The command-line options for the **nco_store_resize** utility are described in the following table.

Table 6. Memstore resize utility command-line options

Command-line option	Description
-help	Displays help on the command-line options and exits.
-messagelevel <i>string</i>	<p>Specifies the message logging level. Possible values are: debug, info, warn, error, and fatal. The default level is error.</p> <p>Messages that are logged at each level are as follows:</p> <ul style="list-style-type: none">• 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 <p>Tip: The value of <i>string</i> can be in uppercase, lowercase, or mixed case.</p>
-messagelog <i>string</i>	Specifies the path to which messages are logged. The default is stderr.

Table 6. Memstore resize utility command-line options (continued)

Command-line option	Description
<code>-newhardlimit integer</code>	<p>Specifies the new hard limit for the memstore, in MB. The default is 500 MB. You cannot set a hard limit that is larger than the maximum memstore size. If you exceed the store size, the hard limit is truncated to the maximum permitted memstore size.</p> <p>The maximum permitted memstore size (in MB) for the associated platform is shown below:</p> <ul style="list-style-type: none"> • AIX: 2047 • HPUX: 1024 • Linux x86: 2047 • Solaris: 2047 • Windows: 700 • zLinux: 1024
<code>-server string</code>	Specifies the name of the ObjectServer for which the memstore size will be increased. The default is NCOMS.
<code>-version</code>	Displays version information on the checkpoint verification utility and exits.

Related reference

“Changing the default and current settings of the ObjectServer (ALTER SYSTEM command)” on page 181

Sending alerts to the ObjectServer using `nco_postmsg`

You can send an alert directly to an ObjectServer by using the **nco_postmsg** utility. This utility accepts name-value pairs for the alert data and constructs an SQL INSERT statement, which is used to insert a new row of data into a specified database table in the ObjectServer.

You can run **nco_postmsg** from the command line, or you can develop scripts or automations that use the **nco_postmsg** command to send alerts to the ObjectServer. The frequency of execution can vary from a few times a day when run from the command line, to a few times a second when run from a script. Multiple instances of the **nco_postmsg** utility can also run simultaneously. Some usage examples of the **nco_postmsg** utility are as follows:

- You can use **nco_postmsg** to send single alerts to the ObjectServer for diagnostic or testing purposes; for example, during system configuration, or when troubleshooting alert delivery problems.
- You can use **nco_postmsg** to send alerts that indicate the start and end of a maintenance period.
- You can use **nco_postmsg** when feedback from an external automation is needed in the ObjectServer. For example, suppose an alert is received that causes an external automation to run. The external automation fires a script that performs an action, and the success or failure of this action needs to be recorded in the ObjectServer. You can include an **nco_postmsg** command in the script to insert a new alert with the status of the action, and additionally set up automations in the ObjectServer to process the new alert.

The **nco_postmsg** utility is installed with the **Probe Support** feature of Tivoli Netcool/OMNIbus, and can therefore be deployed separately from the other Tivoli

Netcool/OMNIbus features, on one or more hosts. A properties file called `nco_postmsg.props` is available for the utility.

The **nco_postmsg** utility can establish a secure connection to the ObjectServer by using SSL in both FIPS 140-2 mode and non-FIPS 140-2 mode.

To send an alert to an ObjectServer, enter the following command:

```
$NCHOME/omnibus/bin/nco_postmsg [ -option [ value ] ... ]  
"column_name=column_value" ...
```

In this command:

- *-option* is the **nco_postmsg** command-line option and *value* is the value to which you are setting the option. Not every option requires you to specify a value.
- *column_name* is a valid column name in the database table into which you want to insert the alert, and *column_value* is the matching data value that you want to insert. Each name-value pair for *column_name=column_value* must be enclosed in double quotation marks, as shown in the syntax for the command. Additionally, *column_value* must be enclosed in single quotation marks if it is a string value.

Note: If you are sending an alert to the `alerts.status` table, a name-value pair for the Identifier field is mandatory. For guidance on setting a value for the Identifier column, see the *IBM Tivoli Netcool/OMNIbus Integration Best Practices* document, which is available at <https://www-304.ibm.com/jct01003c/software/brandcatalog/portal/opal/details?catalog.label=1TW10NC10>.

Tip: When specifying *column_value*, use a data type that is appropriate for the ObjectServer field. You can use Netcool/OMNIbus Administrator to verify the data types that are assigned to fields in database tables, or you can use the ObjectServer SQL DESCRIBE command.

Running nco_postmsg in UTF-8 encoding

You can run the **nco_postmsg** utility in UTF-8 encoding by using the `-utf8enabled` command-line option. You cannot, however, specify the name-value pairs directly from the command line, and must instead add the name-value pairs to a text file. In the text file, you must specify each name-value pair on a separate line, and enclose the value in single quotation marks if it is a string value. The text file must be in UTF-8 encoding, and can contain data for a single alert only. The `nco_postmsg.props` properties file must also be in UTF-8 encoding.

The syntax for running **nco_postmsg** in UTF-8 encoding is:

```
$NCHOME/omnibus/bin/nco_postmsg [ -option [ value ] ... ] -utf8enabled TRUE  
< file_name.txt
```

In this command:

- *-option* is any of the **nco_postmsg** command-line options (other than `-utf8enabled`), and *value* is the value to which you are setting the option.
- *file_name.txt* is the file containing the name-value pairs.

Error processing

When the **nco_postmsg** utility runs, it reads its properties file, validates any command-line options specified, and then constructs the INSERT statement by

using the name-value pairs that were specified. Alerts are sent to the ObjectServer providing the specified name-value pairs and command-line options entered are valid. Any errors that occur are categorized as follows:

Errors that cause the current alert to be written to a cache file.

If a communication error occurs because the ObjectServer is down or is taking too long to respond, the **nco_postmsg** utility saves the alert in a cache file named **nco_postmsg.cache**, and then exits. Any subsequent alerts that are generated while there is a communication failure are also written to the cache file. When the cache file reaches its maximum size, the file is renamed with an **_old** suffix and a new **nco_postmsg.cache** file is created. If a file named **nco_postmsg.cache_old** already exists, it is deleted during the renaming process, and its contents are lost.

Tip: The response time limit of the ObjectServer is set by the **Ipc.Timeout** property of **nco_postmsg**, and the maximum size of the cache file is set by the **MaxCacheFileSize** property.

The next time the **nco_postmsg** utility is run, and can successfully connect to the ObjectServer, alerts are sent to the ObjectServer in the following order:

1. Alerts in the **nco_postmsg.cache_old** file (if one exists). The alerts are sent in order of age, with the oldest being sent first. When empty, the file is deleted.
2. Alerts in the **nco_postmsg.cache** file. When empty, the file is truncated to 0 bytes.
3. The current alert, which was generated when the **nco_postmsg** utility was run.

Errors that cause the current alert to be discarded.

Alerts are discarded due to the following conditions:

- Command-line errors: An invalid properties file path was specified or invalid syntax was specified for the name-value pairs.
- Authentication errors: The authentication to the ObjectServer failed due to invalid login credentials, an invalid ObjectServer name, or SSL errors.
- Database errors: The database table write permission was not set, the database table name was invalid, or there was a syntax error in the constructed INSERT statement.
- File errors: An error occurred writing the entry to the cache file, or there were syntax errors in the properties file.

All errors are written to a log file.

Additional notes

- Users of **nco_postmsg** are subject to the ObjectServer authorization system, and so require insert permission on the database table into which the data is being written.
- If not set within the properties file, a password is required when the **nco_postmsg** utility runs from the command line.
- If the **nco_postmsg** utility is inserting data into the **alerts.status** table, it sets the value of the **FirstOccurrence** and **LastOccurrence** columns to the current UNIX time or POSIX time, unless these column values are explicitly specified on the command line or in a script.

- If a value is specified for the Class column when running **nco_postmsg**, a name-value pair is added to the INSERT statement. If no value is specified for Class, the ObjectServer assigns the default class of 0 (zero) to the alert.
- Multiple instances of the **nco_postmsg** utility can be configured to cache alerts to the same file by using the **CacheFile** property.
- In the unlikely event that a communication error occurs while alerts are being sent from a cache file to the alerts.status table, the alerts are retained in the cache file. The **nco_postmsg** utility attempts to resend the alerts the next time the utility runs. This can result in alerts being sent more than once, in which case, the Tally field will be inaccurately incremented.

Related concepts

Chapter 4, “Using Netcool/OMNIBus Administrator to configure ObjectServers,” on page 59

Related reference

“Displaying details of columns in a table or view (DESCRIBE command)” on page 180

Appendix A, “ObjectServer tables,” on page 321

nco_postmsg properties and command-line options

The **nco_postmsg** utility contains a set of properties and command-line options for sending alerts to the ObjectServer. You can run this command from the \$NCHOME/omnibus/bin directory.

The **nco_postmsg** properties file is \$NCHOME/omnibus/etc/nco_postmsg.props. In an unedited properties file, all properties are set to the default values, and are commented out with a number sign (#) at the beginning of the line. To override a default value, change its setting in the properties file and remove the number sign (#). If you change a setting on the command line, both the default value and the setting in the properties file are overridden.

The properties and command-line options for **nco_postmsg** are described in the following table.

Table 7. nco_postmsg properties and command-line options

Property	Command-line option	Description
CacheFile <i>string</i>	-cachefile <i>string</i>	Specifies the full path and name of the cache file where alerts are stored when they cannot be sent to the ObjectServer due to a communication failure. The default is \$NCHOME/omnibus/var/nco_postmsg.cache. When the cache file reaches its maximum size, as specified by the MaxCacheFileSize property, the nco_postmsg.cache file is renamed nco_postmsg.cache_old, and a new nco_postmsg.cache file is created.
CacheFileWarnThreshold <i>integer</i>	N/A	Causes a warning message to be written to the log file when the nco_postmsg.cache file size exceeds a specified percentage value. Note that this setting has no effect on the nco_postmsg.cache_old file. The default is 90%. The range is 10% - 99%.

Table 7. *nco_postmsg* properties and command-line options (continued)

Property	Command-line option	Description
ConfigCryptoAlg <i>string</i>	N/A	<p>Specifies the cryptographic algorithm to use for decrypting string values (including passwords) that were encrypted with the nco_aes_crypt utility and then stored in the properties file. Set the <i>string</i> value as follows:</p> <ul style="list-style-type: none"> When in FIPS 140–2 mode, use AES_FIPS. When in non-FIPS 140–2 mode, you can use either AES_FIPS or AES. Use AES only if you need to maintain compatibility with passwords that were encrypted by using the tools provided in versions earlier than Tivoli Netcool/OMNIbus V7.2.1. <p>The value that you specify must be identical to that used when you ran nco_aes_crypt with the -c setting, to encrypt the string values.</p> <p>Use this property in conjunction with the ConfigKeyFile property.</p> <p>The default is AES.</p>
ConfigKeyFile <i>string</i>	N/A	<p>Specifies the path and name of the key file that contains the key used to decrypt encrypted string values (including passwords) in the properties file.</p> <p>The key is used at run time to decrypt string values that were encrypted with the nco_aes_crypt utility. The key file that you specify must be identical to the file used to encrypt the string values when you ran nco_aes_crypt with the -k setting.</p> <p>Use this property in conjunction with the ConfigCryptoAlg property.</p>
N/A	-help	Displays help on the command-line options and exits.
Ipc.Timeout <i>integer</i>	-ipctimeout <i>integer</i>	<p>Sets the time, in seconds, that the nco_postmsg utility waits for a response from the ObjectServer, when attempting to send alerts. The default is 60 seconds.</p> <p>If this time is exceeded (or the ObjectServer is down), a communication error occurs and the alert is instead sent to the cache file.</p>
MaxCacheFileSize <i>integer</i>	-maxcachefilesize <i>integer</i>	<p>Sets the maximum size of the cache file, in KB. The default is 10240 KB.</p> <p>If set to 0 (zero), no cache files are created or used.</p>

Table 7. *nco_postmsg* properties and command-line options (continued)

Property	Command-line option	Description
MessageLevel <i>string</i>	<code>-messagelevel <i>string</i></code>	<p>Specifies the message logging level. Possible values are: debug, info, warn, error, and fatal. The default level is warn.</p> <p>Messages that are logged at each level are as follows:</p> <ul style="list-style-type: none"> • 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 <p>Tip: The value of <i>string</i> can be in uppercase, lowercase, or mixed case.</p>
MessageLog <i>string</i>	<code>-messagelog <i>string</i></code>	<p>Specifies where messages are logged. Messages can be logged to a log file or to stderr.</p> <p>The default is <code>\$NCHOME/omnibus/log/nco_postmsg.log</code>.</p>
Name <i>string</i>	<code>-name <i>string</i></code>	<p>Specifies a client name for the nco_postmsg utility. This name is used as the application description when nco_postmsg connects to the ObjectServer. The default is <code>nco_postmsg</code>.</p> <p>The value of the Name property also determines the name of the properties file, which takes the following format: <code>\$NCHOME/omnibus/etc/<i>name</i>.props</code>.</p> <p>Tip: You can use the Name property to distinguish between multiple nco_postmsg clients that are sending alerts to the ObjectServer, and to maintain individual properties files per client.</p>
Password <i>string</i>	<code>-password <i>string</i></code>	Specifies the password for the user connecting to the ObjectServer. The default is <code>''</code> .
Props.CheckNames TRUE FALSE	N/A	Causes the nco_postmsg utility to terminate if any specified property is invalid. The default is TRUE.
PropsFile <i>string</i>	<code>-propsfile <i>string</i></code>	Specifies the full path and name of the properties file for the nco_postmsg utility. The default name is <code>\$NCHOME/omnibus/etc/nco_postmsg.props</code> .
Server <i>string</i>	<code>-server <i>string</i></code>	Sets the name of the ObjectServer to which the nco_postmsg utility connects and sends an alert. The default is <code>NCOMS</code> .
SSLServerCommonName <i>string1,...</i>	N/A	Specifies a comma-separated list of common names to use if the nco_postmsg utility is connecting to the ObjectServer by using SSL. The default is <code>''</code> .
Table <i>string</i>	<code>-table <i>string</i></code>	Sets the name of the database table into which the INSERT statement adds the alert data. The default is <code>alerts.status</code> .
UserName <i>string</i>	<code>-username <i>string</i></code> or <code>-user <i>string</i></code>	Specifies the user name that is used to authenticate when connecting to the ObjectServer. The default is the user name used to log on to the computer.

Table 7. *nco_postmsg* properties and command-line options (continued)

Property	Command-line option	Description
N/A	<div>Windows</div> -utf8enabled TRUE FALSE	<p>Controls the encoding of data that is passed into, or generated by, this application on Windows.</p> <p>Set the value of -utf8enabled to TRUE to run the application in the UTF-8 encoding. The default is FALSE, which causes the default system code page to be used.</p> <p>Important: Although a UTF8Enabled property is available, an attempt to enable UTF-8 encoding by setting this property to TRUE will have no effect. To run in a UTF-8 encoding on Windows, you must always use the -utf8enabled command-line option.</p>
N/A	-version	Displays version information for the utility and exits.

nco_postmsg examples and resulting INSERT statements

The **nco_postmsg** utility accepts input from command-line options, scripts, or a text file, and generates an ObjectServer SQL INSERT statement that is sent to an ObjectServer. The following examples show sample **nco_postmsg** commands and their resulting INSERT statements.

Example 1

This example writes an alert to the default NCOMS ObjectServer and default alerts.status table. The user name and password for authenticating to NCOMS have not been set in the **nco_postmsg** properties file, and are instead specified from the command line.

The **nco_postmsg** command entered is:

```
nco_postmsg -user root -password "" "Identifier='example 1'" "Severity=3"
"Manager='nco_postmsg'" "Summary='An event occurred'"
```

The resulting INSERT statement that is sent to the alerts.status table in NCOMS is:

```
insert into alerts.status (Identifier,Severity,Manager,Summary,FirstOccurrence,LastOccurrence)
values ('example 1',3,'nco_postmsg','an event occurred',1255341764,1255341764);
```

Because name-value pairs were not explicitly specified for FirstOccurrence and LastOccurrence in the **nco_postmsg** command, FirstOccurrence and LastOccurrence are automatically set to the current UNIX time in the INSERT statement that is sent to the alerts.status table.

Example 2

This example writes an alert to the alerts.status table of an ObjectServer named PRESLEY. The **nco_postmsg** utility authenticates to the ObjectServer by using the user name myname and an encrypted password, which are specified in the **nco_postmsg** properties file.

In this case, it is assumed that you have used property value encryption to encrypt the password in the properties file so that the password cannot be read without a key. The properties file settings for the login credentials, including password encryption, are uncommented and completed as follows:

```
ConfigCryptoAlg: 'AES_FIPS'
ConfigKeyFile: '/dir/subdir/secret.txt'
```

...

```
Password: '@44:Kris2m3QLsy+dZYnt3/jpt18cd7c6Fmboaj+E6XrNw8=@'  
UserName: 'myname'
```

The **nco_postmsg** command entered is:

```
nco_postmsg -server PRESLEY "Identifier='example 2'" "Node='London'"
```

The resulting INSERT statement that is sent to the alerts.status table in PRESLEY is as follows:

```
insert into alerts.status (Identifier,Node,FirstOccurrence,LastOccurrence)  
values ('example 2','London',1255341764,1255341764);
```

Example 3

This example inserts an alert into the mydb.mytable table in the default NCOMS ObjectServer. In this case, it is assumed that a user name and password are specified in the **nco_postmsg** properties file.

The **nco_postmsg** command entered is:

```
nco_postmsg -table mydb.mytable "Identifier='example 3'" "Occurrence=1234567890"  
"Summary='write into my table'"
```

The resulting INSERT statement that is sent to the mydb.mytable table in NCOMS is as follows:

```
insert into mydb.mytable (Identifier,Occurrence,Summary)  
values ('example 3',1234567890,'write into my table');
```

Example 4

This example shows how to run the **nco_postmsg** utility in UTF-8 encoding.

The **nco_postmsg** utility authenticates to the ObjectServer by using the user name myname and a password, which are specified in the nco_postmsg.props properties file as follows:

```
Password: 'secret'  
UserName: 'myname'
```

This properties file must be saved in UTF-8 encoding.

A text file, my_data.txt, is created with the following name-value pairs for the alert to be sent to the ObjectServer:

```
Identifier='example 2'  
Node='London'
```

This text file must be saved in UTF-8 encoding.

To run **nco_postmsg** in UTF-8 encoding, and write an alert to the alerts.status table of an ObjectServer named PRESLEY (also presumably running in UTF-8 encoding), the command entered is:

```
nco_postmsg -server PRESLEY -utf8enabled TRUE < /file_path/my_data.txt
```

The resulting INSERT statement that is sent to the alerts.status table in PRESLEY is as follows:

```
insert into alerts.status (Identifier,Node,FirstOccurrence,LastOccurrence)  
values ('example 2','London',1255341764,1255341764);
```

Chapter 2. Configuring a proxy server

The ObjectServer receives alert information from probes. In a standard configuration, alerts are forwarded directly to the ObjectServer. You can configure a proxy server to reduce the number of probe connections to an ObjectServer.

Where a large number of probes are forwarding alert information directly to the ObjectServer, and a large number of desktop connections are also made to the same ObjectServer, there can be a negative impact on performance.

A proxy server provides a buffer to reduce the number of direct connections to the primary ObjectServer. Multiple probe connections made to the proxy server are multiplexed and forwarded through a single connection to the ObjectServer.

The following figure shows how probes communicate with the proxy server.

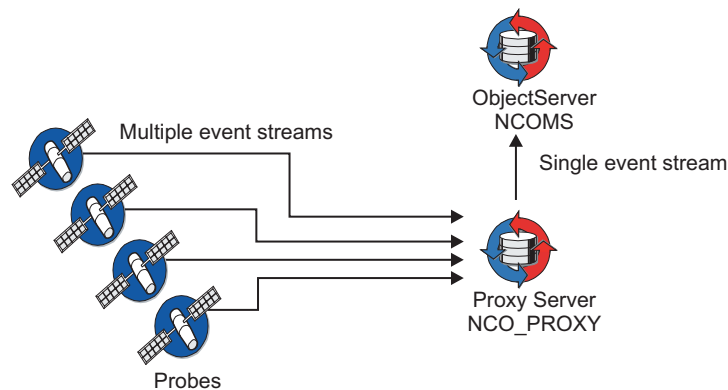


Figure 1. Example proxy server architecture

Starting the proxy server

You can start the proxy server automatically by using process control on UNIX and Windows, and also by using services on Windows. You can also start the proxy server manually from the command line. In general, use process control to start the proxy server.

Starting a proxy server by using process control

On UNIX and Windows, you can start a proxy server as a process by using process control. The proxy server must be defined as a process or part of a service.

Related concepts

Chapter 7, "Using process control to manage processes and external procedures," on page 249

Starting a proxy server by using services (Windows)

On Windows, you can optionally install and run the proxy server as a Windows service. When the service is set to automatic, the proxy server starts when the computer starts.

You must manually install and configure the proxy server to run as a service on a Windows host.

Starting the proxy server manually

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

From the command line, enter the appropriate command for your operating system:

Table 8. Starting the proxy server from the command line

Operating system	Command
UNIX	<code>\$NCHOME/omnibus/bin/nco_proxyserv [-name <i>proxyname</i>] [-server <i>servername</i>]</code>
Windows	<code>%NCHOME%\omnibus\bin\nco_proxyserv [-name <i>proxyname</i>] [-server <i>servername</i>]</code>

In both 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 default proxy server name is `NCO_PROXY`. If you do not specify the `-server` command-line option, the proxy server buffers connections for the NCOMS ObjectServer.

You can start the proxy server with additional command-line options.

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 `$NCHOME/omnibus/etc/proxyserver.props`.

In the properties file, a property and its corresponding value are separated by a colon (:). String values are surrounded by quotation marks; for example:

ServerName: "NCO_PROXY"

Tip: You can encrypt string values in a properties file by using property value encryption.

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 that 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 a proxy server, the default file is used. Use the `-propsfile` command-line option to specify the full path and file name of an alternative properties file.

The following table lists the proxy server properties and command-line options.

Table 9. Proxy server properties and command-line options

Property	Command-line option	Description
AuthPassword <i>string</i>	N/A	<p>The password that is associated with the user name used to authenticate the proxy server when it connects to an ObjectServer running in secure mode. The default is ''.</p> <p>When in FIPS 140–2 mode, the password can either be specified in plain text, or can be encrypted with the nco_aes_crypt utility. If you are encrypting passwords by using nco_aes_crypt in FIPS 140–2 mode, you must specify AES_FIPS as the encryption algorithm.</p> <p>When in non-FIPS 140–2 mode, the password can be encrypted with the nco_g_crypt or nco_aes_crypt utilities. If you are encrypting passwords by using nco_aes_crypt in non-FIPS 140–2 mode, you can specify either AES_FIPS or AES as the encryption algorithm. Use AES only if you need to maintain compatibility with passwords that were encrypted using the tools provided in versions earlier than Tivoli Netcool/OMNIBus V7.2.1.</p>
AuthUserName <i>string</i>	N/A	<p>The user name that is used to authenticate the proxy server when it connects to an ObjectServer running in secure mode. The default is root.</p>
ConfigCryptoAlg <i>string</i>	N/A	<p>Specifies the cryptographic algorithm to use for decrypting string values (including passwords) that were encrypted with the nco_aes_crypt utility and then stored in the properties file. Set the <i>string</i> value as follows:</p> <ul style="list-style-type: none"> • When in FIPS 140–2 mode, use AES_FIPS. • When in non-FIPS 140–2 mode, you can use either AES_FIPS or AES. Use AES only if you need to maintain compatibility with passwords that were encrypted by using the tools provided in versions earlier than Tivoli Netcool/OMNIBus V7.2.1. <p>The value that you specify must be identical to that used when you ran nco_aes_crypt with the -c setting, to encrypt the string values.</p> <p>Use this property in conjunction with the ConfigKeyFile property.</p>

Table 9. Proxy server properties and command-line options (continued)

Property	Command-line option	Description
ConfigKeyFile <i>string</i>	N/A	<p>Specifies the path and name of the key file that contains the key used to decrypt encrypted string values (including passwords) in the properties file.</p> <p>The key is used at run time to decrypt string values that were encrypted with the nco_aes_crypt utility. The key file that you specify must be identical to the file used to encrypt the string values when you ran nco_aes_crypt with the -k setting.</p> <p>Use this property in conjunction with the ConfigCryptoAlg property.</p>
ConnectionRatio <i>integer</i>	-ratio <i>integer</i>	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	-help	Displays the supported command-line options and exits.
N/A	-logfile <i>string</i>	Sets the name of the file to which the proxy server writes messages, including errors. By default, the file is \$NCHOME/omnibus/log/ <i>servername</i> .log, where the <i>servername</i> is defined by the -name option.
MaxConnections <i>integer</i>	-max <i>integer</i>	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>	-networktimeout <i>integer</i>	Specifies a time in seconds after which a login attempt or connection to the ObjectServer times out, if a network failure occurs. 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 attempts to connect to a backup ObjectServer, where available. The default is 20 seconds.
OldTimeStamp TRUE FALSE	-oldtimestamp TRUE FALSE	<p>Specifies the timestamp format to use in the log file.</p> <p>Set the value to TRUE to display the timestamp format that is used in Tivoli Netcool/OMNIBUS V7.2.1, or earlier. For example: dd/MM/YYYY hh:mm:ss AM or dd/MM/YYYY hh:mm:ss PM when the locale is set to en_GB on a Solaris 9 computer.</p> <p>Set the value to FALSE to display the ISO 8601 format in the log file. For example: YYYY-MM-DDThh:mm:ss, where T separates the date and time, and hh is in 24-hour clock. The default is FALSE.</p>

Table 9. Proxy server properties and command-line options (continued)

Property	Command-line option	Description
N/A	-propsfile <i>string</i>	Sets the proxy server properties file name. The default name is \$NCHOME/omnibus/etc/ <i>servername</i> .props, where the <i>servername</i> is defined by the -name option.
RemoteServer <i>string</i>	-server <i>string</i>	Sets the name of the ObjectServer to which the proxy server connects. The default is NCOMS.
SecureMode TRUE FALSE	-secure	Sets the security mode of the proxy server. If enabled, the proxy server authenticates probe connection requests with a user name and password. If disabled (the default), probes can connect to the proxy server without a user name and password.
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.

Related reference

“Running the proxy server in secure mode”

“Running the ObjectServer in secure mode” on page 17

Connecting to the proxy server

To connect probes to the proxy server, specify 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.

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 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.

Password encryption details for running in FIPS 140–2 mode and non-FIPS 140–2 mode are described in the following table.

Table 10. Password encryption in FIPS 140–2 mode and non-FIPS 140–2 mode

Mode	Action
FIPS 140–2 mode	<p>When in FIPS 140–2 mode, passwords can either be specified in plain text or in encrypted format. You can encrypt passwords by using property value encryption, as follows:</p> <ol style="list-style-type: none"> 1. If you do not yet have a key for encrypting the password, create one by running the nco_keygen utility, which is located in <code>\$NCHOME/omnibus/bin</code>. 2. Run the nco_aes_crypt utility to encrypt the password with the key that was generated by the nco_keygen utility. The nco_aes_crypt utility is also located in <code>\$NCHOME/omnibus/bin</code>. Note that you must specify AES_FIPS as the algorithm to use for encrypting the password. 3. Open the properties file to which you want to add the encrypted password and specify this encrypted output for the AuthPassword setting. Note: You must also set the ConfigKeyFile property to the key file that you specified when running nco_aes_crypt, and set the ConfigCryptoAlg property to the encryption algorithm used.
Non-FIPS 140–2 mode	<p>When in non-FIPS 140–2 mode, passwords can either be specified in plain text or in encrypted format. However, the client always transmits encrypted login information irrespective of the password encryption that is used in the properties file. You can encrypt passwords by using the nco_g_crypt utility or by using property value encryption, as follows:</p> <ul style="list-style-type: none"> • To encrypt a password by using the nco_g_crypt utility, run the command as follows: <code>\$NCHOME/omnibus/bin/nco_g_crypt plaintext_password</code> In this command, <i>plaintext_password</i> represents the unencrypted form of the password. The nco_g_crypt utility takes the unencrypted password and displays an encrypted version. Open the properties file to which you want to add the encrypted password and specify this encrypted output for the AuthPassword setting. • To encrypt a password by using property value encryption, you require a key that is generated with the nco_keygen utility. You can then run nco_aes_crypt to encrypt the password with the key. Note that you can specify either AES_FIPS or AES as the algorithm for encrypting the password. Use AES only if you need to maintain compatibility with passwords that were encrypted using the tools provided in versions earlier than Tivoli Netcool/OMNIBus V7.2.1. Open the file to which you want to add the encrypted password and specify this encrypted output for the AuthPassword setting. Note: You must also set the ConfigKeyFile property to the key file that you specified when running nco_aes_crypt, and set the ConfigCryptoAlg property to the encryption algorithm used.

If the ObjectServer is running in secure mode, the proxy server must also have the **AuthUserName** and **AuthPassword** properties in its property file to connect the ObjectServer. If the user name and password combination is incorrect, the ObjectServer issues an error message. The **AuthPassword** value can be in plain text or encrypted, as described in the preceding table.

Attention: Passwords encrypted with **nco_g_crypt** can be used in the same way as unencrypted passwords to access the ObjectServer. Therefore, you must set appropriate permissions on any files containing encrypted passwords to prevent unauthorized access. Alternatively, passwords that have been encrypted with **nco_g_crypt** must be further encrypted with **nco_aes_crypt**, and permissions on the key file must be set appropriately.

For further information about the probe properties, see the *IBM Tivoli Netcool/OMNIBus Probe and Gateway Guide*.

Chapter 3. Configuring a firewall bridge server

In a secure environment in which the ObjectServer and probes are separated by a firewall, configure a firewall bridge server so that the probes can connect to the ObjectServer from outside the secure network.

In a standard secure configuration, alerts are forwarded from probes directly to the ObjectServer. If probes are located outside the firewall, the firewall rejects the connections to the ObjectServer. By configuring a firewall bridge server, you can overcome this security restriction.

The firewall bridge consists of two servers: a Server Access Bridge and a Client Access Bridge, which run either side of the firewall. A communication channel between the two servers is initiated by the Server Access Bridge.

The firewall bridge uses this communication channel to create new data channels between the Server Access Bridge and the Client Access Bridge. Client connections and data can be sent to the ObjectServer from outside the firewall. Probes still initiate a connection but it is now made to the local Client Access Bridge, situated on the same side of the firewall. This enables the Client Access Bridge, situated outside the firewall, to provide data flow to the ObjectServer, situated inside the firewall.

The following figure shows the data-flow across a firewall between the ObjectServer and two probes located outside the firewall.

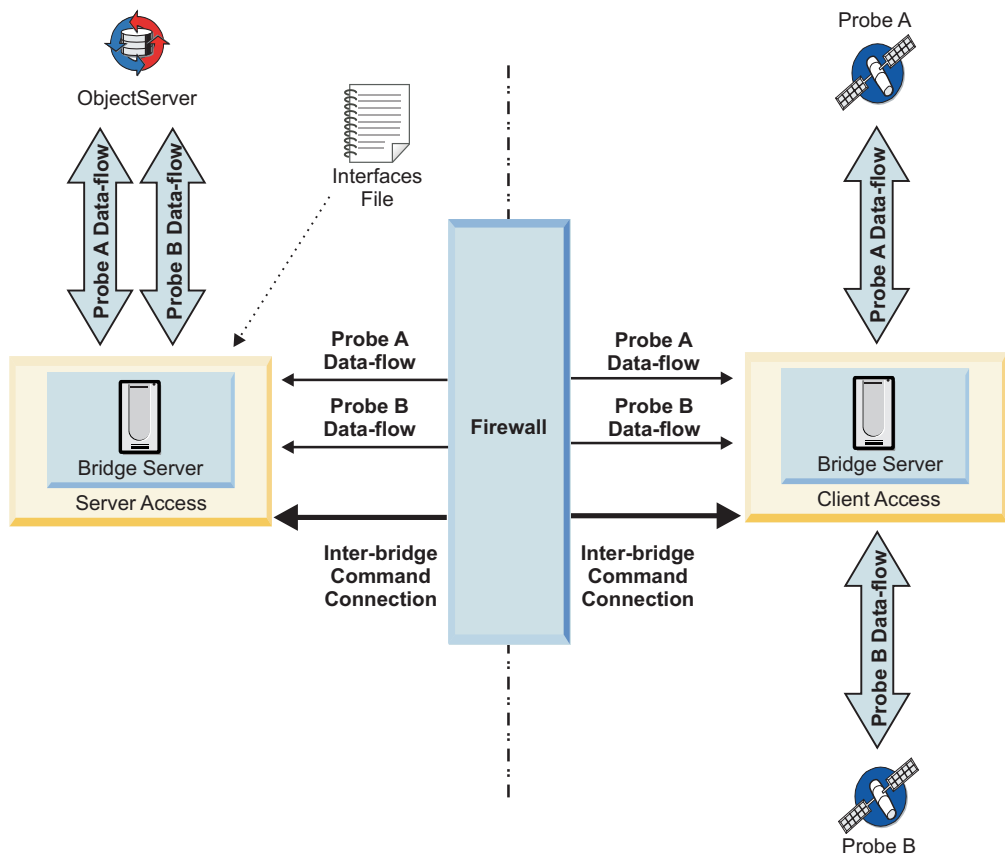


Figure 2. Example firewall bridge server architecture

A standard firewall bridge server configuration

A firewall bridge can be configured so that a probe can connect to an ObjectServer from outside the secure network.

The following figure shows the configuration setup for a standard firewall bridge:

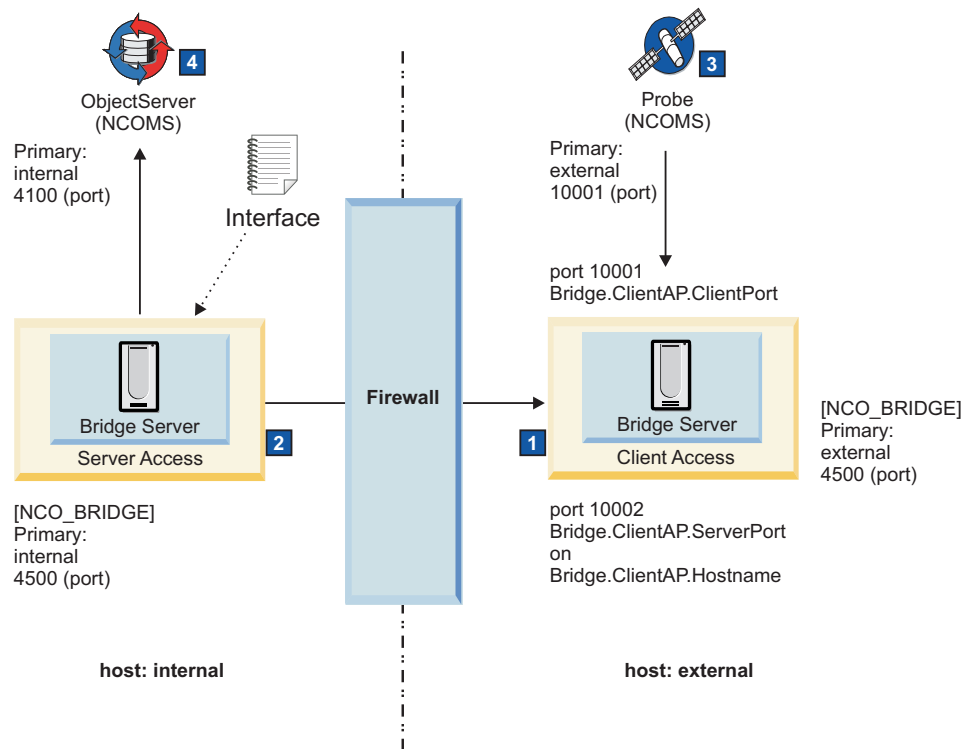


Figure 3. A standard firewall bridge server configuration

The configuration flow is as follows:

1 The Client Access Bridge

The Client Access Bridge listens for connections from the Server Access Bridge on the port and host name set in the associated **Bridge.ClientAP.ServerPort** and **Bridge.ClientAP.Hostname** properties of the Client Access Bridge property file. The Client Access Bridge also listens for connections from probes on the port specified by **BridgeClientAP.ClientPort** property.

2 The Server Access Bridge

The Server Access Bridge makes a connection to Client Access Bridge (across the Firewall) on the port and host name set in the **Bridge.ClientAP.ServerPort** and **Bridge.ClientAP.Hostname** properties of the Server Access Bridge property file.

3 The probe

The probe makes a connection to the client access port of the Client Access Bridge by connecting to the port and host name associated with the server name defined in the `omni.dat` interfaces file.

1 The Client Access Bridge

The Client Access Bridge then requests new data flow from the Server Access Bridge.

2 The Server Access Bridge

The Server Access Bridge makes a new connection to the ObjectServer (NCOMS) by using the port and host name associated with NCOMS in the `omni.dat` interfaces file.

The Server Access Bridge then makes a new data flow connection to the Client Access Bridge.

4 The ObjectServer

Data packets sent from the probe (3) are received by the Client Access Bridge (1) and are routed along the associated data flow connection to the Server Access Bridge (2). The Server Access Bridge forwards the data along the associated network connection to the ObjectServer (4), and data is returned along the same connection.

A multiple firewall bridge server configuration

Multiple firewall bridge servers can be configured so that probes can connect to an ObjectServer from across multiple firewalls.

The following figure shows the configuration setup for multiple firewall bridge servers:

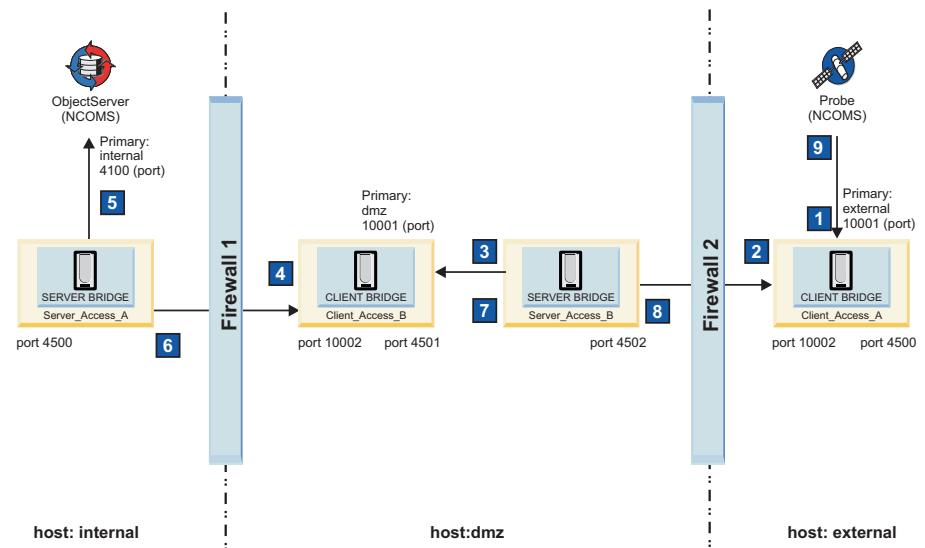


Figure 4. A multiple firewall bridge server configuration

The configuration flow is as follows:

1

The probe makes an initial connection to the Client Access Bridge server (CLIENT_ACCESS_A) on the external host. It uses the port (10001) and host name (external) associated with the NCOMS server name defined in the `omni.dat` interfaces file.

2

The Client Access Bridge server (CLIENT_ACCESS_A) then requests a new data-flow connection (across Firewall 2) from its associated Server Access Bridge server (SERVER_ACCESS_B) using the existing inter-bridge communication channel.

3

The Server Access Bridge server (SERVER_ACCESS_B) makes a new connection to the Client Access Bridge server (CLIENT_ACCESS_B). It uses the port (10001) and host name (dmz) associated with the NCOMS server name defined in the `omni.dat` interfaces file.

4

The Client Access Bridge server (CLIENT_ACCESS_B) then requests a new data-flow connection (across Firewall 1) from its associated Server Access Bridge server (SERVER_ACCESS_A) using the existing inter-bridge communication channel.

5

The Server Access Bridge server (SERVER_ACCESS_A) makes a new connection to the ObjectServer (NCOMS) on the internal host. It uses the port and host name associated with NCOMS in the `omni.dat` interfaces file. The new connection is acknowledged by the ObjectServer (NCOMS).

6 and **7**

The Server Access Bridge server (SERVER_ACCESS_A) initiates a new data-flow connection (across Firewall 1) to the Client Access Bridge server (CLIENT_ACCESS_B) and in turn a new connection is made to the Server Access Bridge server (SERVER_ACCESS_B).

8 and **9**

The Server Access Bridge server (SERVER_ACCESS_B) creates a new data-flow connection (across Firewall 2) to the Client Access Bridge server (CLIENT_ACCESS_A). This connection is acknowledged by the Client Access Bridge and in turn the incoming probe connection is accepted.

Data packets are now routed from the probe along the open connections and data-flow channels initiated by the bridge servers, and finally to the ObjectServer (NCOMS).

Firewall bridge server failover configuration

A basic firewall bridge server failover architecture comprises all the components from the basic architecture together with an additional Server Access Bridge server and Client Access Bridge server. If an initial connection to the ObjectServer fails, the probe attempts to connect to the ObjectServer using a backup Server Access Bridge server and Client Access Bridge server.

The following figure illustrates a basic firewall bridge server failover configuration:

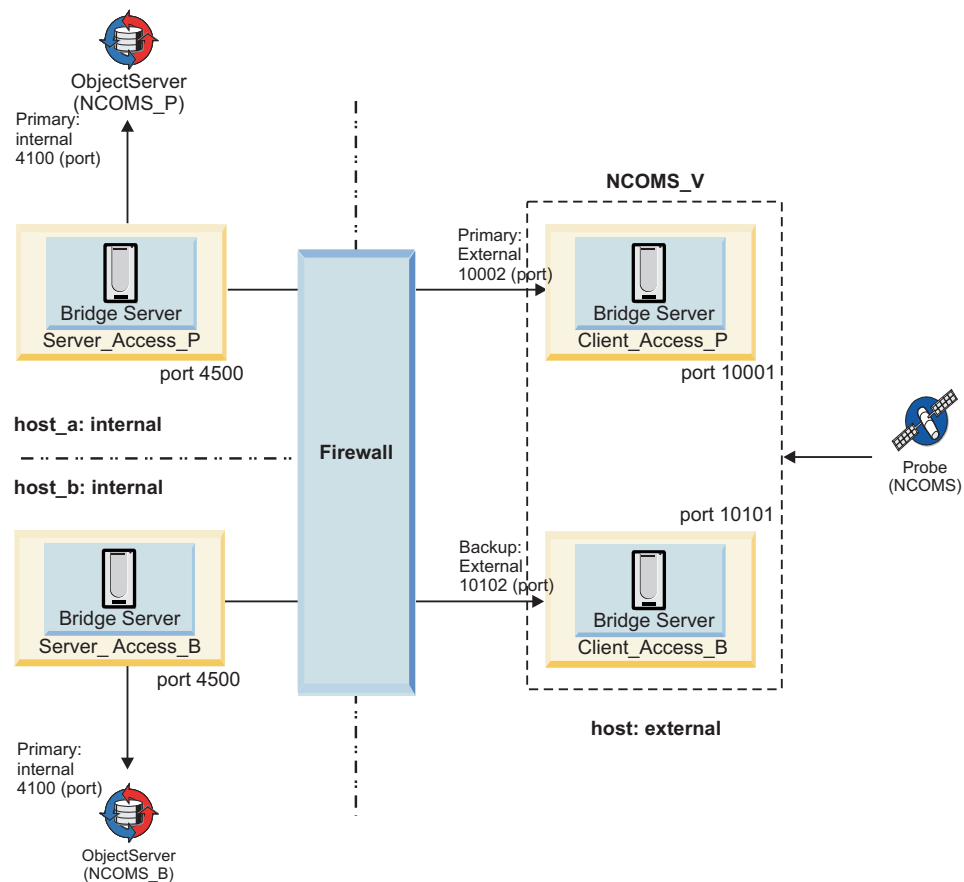


Figure 5. Basic failover configuration

In the basic firewall bridge server failover configuration, host_a and host_b are deployed inside the firewall, and a further host is deployed outside the firewall.

The Server Access Bridge server (Server_Access_P) runs on host_a (internal) and port 4500, and is configured to connect to the ObjectServer (NCOMS_P) using the host name (Primary) and port (4100) defined in the `omni.dat` interfaces file. The Server Access Bridge server connects to a corresponding Client Access Bridge server (Client_Access_P) which runs on the external host with server port external: 10002 and client port: 10001 defined.

The Server Access Bridge server (Server_Access_B) runs on host_b (internal) and port 4500, and is configured to connect to the ObjectServer (NCOMS_B) using the host name (Primary) and port (4100) defined in the `omni.dat` interfaces file. The Server Access Bridge server connects to a corresponding Client Access Bridge server (Client_Access_B) which runs on the external host with server port external: 10102 and client port: 10101 defined.

The probe makes an initial connection to port 10001 on the Client Access Bridge server (Client_Access_P) and then routes the data across the firewall to the ObjectServer (NCOMS_P) using the Server Access Bridge server (Server_Access_P).

If that connection is not available, the probe uses port 10101 on the Client Access Bridge server (Client_Access_B) to route the data across the firewall to the ObjectServer (NCOMS_B) using the Server Access Bridge server (Server_Access_B).

Starting the firewall bridge server

You can start the firewall bridge server automatically by using process control on UNIX and Windows, and also by using services on Windows. You can also start the firewall bridge server manually from the command-line interface.

Starting a firewall bridge server by using process control

On UNIX and Windows, you can start a firewall bridge server as a process by using process control. The firewall bridge server must be defined as a process or part of a service.

Related concepts

Chapter 7, “Using process control to manage processes and external procedures,” on page 249

Starting a firewall bridge server by using Windows services

On Windows, you can optionally install and run the firewall bridge server as a Windows service. When the service is set to automatic, the bridge server starts when the computer starts.

You must manually install and configure the firewall bridge server to run as a service on a Windows host.

For further information about installing and configuring a process agent as a Windows service, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

Starting the firewall bridge server manually

Use the `nco_bridgeserv` command to start the firewall bridge server manually.

From the command line, enter the appropriate command for your operating system:

Table 11. Starting the firewall bridge server from the command-line interface

Operating system	Command
UNIX	<code>\$NCHOME/omnibus/bin/nco_bridgeserv [-name <i>bridgename</i>]</code>
Windows	<code>%NCHOME%\omnibus\bin\nco_bridgeserv [-name <i>bridgename</i>]</code>

In both commands, *bridgename* is the name of the firewall bridge server. If you do not specify the `-name` command-line option, the default firewall bridge server name is `NCO_BRIDGE`.

You can start the firewall bridge server with additional command-line options.

Firewall bridge server properties and command-line options

The firewall bridge 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 `$NCHOME/omnibus/etc/bridgeserver.props`.

In the properties file, a property and its corresponding value are separated by a colon (:). String values are surrounded by quotation marks; for example:

```
ServerName: "NCO_BRIDGE"
```

Tip: You can encrypt string values in a properties file by using property value encryption.

Command-line options for the firewall bridge server use the following format:

```
nco_bridgeserv [ -option [ value ] ... ]
```

In this command, *-option* is the command-line option and *value* is the value that 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 a firewall bridge server, the default file is used. Use the `-propsfile` command-line option to specify the full path and file name of an alternative properties file.

The following table lists the firewall bridge server properties and command-line options.

Table 12. Bridge server properties and command-line options

Property	Command-line option	Description
AuthPassword <i>string</i>	<code>-authpasswd <i>string</i></code>	<p>The password that is associated with the user name used to authenticate the proxy server when it connects to an ObjectServer running in secure mode. The default is ''.</p> <p>When in FIPS 140–2 mode, the password can either be specified in plain text, or can be encrypted with the nco_aes_crypt utility. If you are encrypting passwords by using nco_aes_crypt in FIPS 140–2 mode, you must specify <code>AES_FIPS</code> as the encryption algorithm.</p> <p>When in non-FIPS 140–2 mode, the password can be encrypted with the nco_g_crypt or nco_aes_crypt utilities. If you are encrypting passwords by using nco_aes_crypt in non-FIPS 140–2 mode, you can specify either <code>AES_FIPS</code> or <code>AES</code> as the encryption algorithm. Use <code>AES</code> only if you need to maintain compatibility with passwords that were encrypted using the tools provided in versions earlier than Tivoli Netcool/OMNIBus V7.2.1.</p>
AuthUserName <i>string</i>	<code>-authusername <i>string</i></code>	<p>The user name that is used to authenticate the proxy server when it connects to an ObjectServer running in secure mode. The default is ''.</p>

Table 12. Bridge server properties and command-line options (continued)

Property	Command-line option	Description
Bridge.AllowConnections TRUE FALSE	-disallowconnections	When TRUE, a user can connect to the firewall bridge server's command port. The default is TRUE.
Bridge.ClientAP.ClientPort <i>unsigned</i>	-cpclientport	Specifies the client listening port for inbound Netcool clients (CLIENT_AP only). The default is 10001.
Bridge.ClientAP.Hostname <i>string</i>	-cpclienthname <i>string</i>	Specifies the host name of the client access point bridge server (SERVER_AP only). The default is localhost.
Bridge.ClientAP.ServerPort <i>unsigned</i>	-cpserverport	Specifies the server listening port for inbound bridge clients. The default is 10002.
Bridge.PAMEnable TRUE FALSE	-pamdisable	When TRUE, a PAM module is used to authenticate client logins. The default is TRUE.
Bridge.PAMServiceName <i>string</i>	-pamsrvname <i>string</i>	Specifies the PAM service name used for PAM authentication. The default is nco_bridgeserv.
Bridge.RetryInterval <i>unsigned</i>	-retryinterval <i>string</i>	Specifies the retry interval (in seconds) for bridge to bridge communications (SERVER_AP only). The default is 30.
Bridge.Role <i>string</i>	-bridgerole <i>string</i>	Specifies the role performed by the firewall bridge server: CLIENT_AP: The firewall bridge server runs as a client bridge. SERVER_AP: The firewall bridge server runs as a server bridge. The default is SERVER_AP.
Bridge.ServerAP.Server <i>string</i>	-spserver <i>string</i>	Specifies the name of the ObjectServer to which the firewall bridge server should forward connections (SERVER_AP only). The default is NCOMS.
Bridge.ThreadPool.MaxJobs <i>unsigned</i>	-tpmaxjobs	Specifies the maximum number of data flows handled by a thread pool worker. The default is 10.
Bridge.TrustedHostFile <i>boolean</i>	-trustedhostfile	Specifies the path to the trusted hosts security file used by the firewall bridge server. The default is \$OMNIHOME/etc/NCO_BRIDGE.thosts. Note: This only applies to the CLIENT_AP bridge server. The SERVER_AP bridge server does not accept any incoming connections.

Table 12. Bridge server properties and command-line options (continued)

Property	Command-line option	Description
ConfigCryptoAlg <i>string</i>	N/A	<p>Specifies the cryptographic algorithm to use for decrypting string values (including passwords) that were encrypted with the nco_aes_crypt utility and then stored in the properties file. Set the <i>string</i> value as follows:</p> <ul style="list-style-type: none"> When in FIPS 140–2 mode, use AES_FIPS. When in non-FIPS 140–2 mode, you can use either AES_FIPS or AES. Use AES only if you need to maintain compatibility with passwords that were encrypted by using the tools provided in versions earlier than Tivoli Netcool/OMNIbus V7.2.1. <p>The value that you specify must be identical to that used when you ran nco_aes_crypt with the -c setting, to encrypt the string values.</p> <p>Use this property in conjunction with the ConfigKeyFile property.</p>
ConfigKeyFile <i>string</i>	N/A	<p>Specifies the path and name of the key file that contains the key used to decrypt encrypted string values (including passwords) in the properties file.</p> <p>The key is used at run time to decrypt string values that were encrypted with the nco_aes_crypt utility. The key file that you specify must be identical to the file used to encrypt the string values when you ran nco_aes_crypt with the -k setting.</p> <p>Use this property in conjunction with the ConfigCryptoAlg property.</p>
Connections <i>integer</i>	-connections <i>integer</i>	<p>Sets the maximum number of available connections for clients connecting to the administration port.</p> <p>The maximum value is 1024. The default value is 30. Up to two connections can be used by the system.</p>
N/A	-help	Displays the supported command-line options and exits.
Ipcc.Timeout <i>integer</i>	-ipcc timeout <i>integer</i>	Sets the time, in seconds, that the nco_bridgeserv utility waits for a response from the ObjectServer. The default value is 60.
MaxLogFileSize <i>integer</i>	-maxlogfilesize <i>integer</i>	<p>Specifies the maximum size (in KB) the log file can grow to. The default is 1024 KB.</p> <p>When it reaches the size specified, the <i>servername.log</i> file is renamed <i>servername.log_OLD</i> and a new log file is started. When the new file reaches the maximum size, it is renamed and the process starts again.</p>

Table 12. Bridge server properties and command-line options (continued)

Property	Command-line option	Description
MessageLevel <i>string</i>	<code>-messagelevel <i>string</i></code>	<p>Specifies the message logging level. Possible values are: debug, info, warn, error, and fatal. The default level is warn.</p> <p>Messages that are logged at each level are as follows:</p> <ul style="list-style-type: none"> 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 <p>Tip: The value of <i>string</i> can be in uppercase, lowercase, or mixed case.</p>
MessageLog <i>string</i>	<code>-messagelog <i>string</i></code>	<p>Specifies the path to which messages are logged. The default is \$NCHOME/omnibus/log/NCOMS.log.</p> <p>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 %NCHOME%\omnibus\log\nco_objserv*.err.</p>
Name <i>string</i>	<code>-name <i>string</i></code>	Sets the firewall bridge server name, which must be unique. This is the name that is configured in the Server Editor. The default is NCO_BRIDGE.
Props.CheckNames TRUE FALSE	N/A	When TRUE, the firewall bridge server does not run if any specified property is invalid. The default is TRUE.
PropsFile <i>string</i>	<code>-propsfile <i>string</i></code>	Sets the firewall bridge server properties file name. The default name is <i>servername.props</i> , where the <i>servername</i> is defined by the <code>-name</code> option.
SecureMode TRUE FALSE	<code>-secure</code>	Sets the security mode of the firewall bridge server. If enabled, the firewall bridge server authenticates client connection requests with a user name and password. If disabled (the default), clients can connect to the firewall bridge server without a user name and password.
UniqueLog TRUE FALSE	<code>-uniquelog</code>	<p>If TRUE, the log file is uniquely named by appending the process ID of the Bridge Server to the default log file name. For example, if the NCO_BRIDGE firewall bridge server is running as process 1234, the log file is named NCHOME/omnibus/log/ NCO_BRIDGE.1234.log. The default is FALSE.</p> <p>If the MessageLog property is set to stderr or stdout, the UniqueLog property is ignored.</p>
N/A	<code>-version</code>	Displays version information about the bridge server and exits.

Trusted hosts definition file

The Client Access Bridge server is often deployed in a non-secure network and provides remote access to an ObjectServer located on an internal network. To prevent unauthorized access to the ObjectServer, the Client Access Bridge server uses the trusted hosts definition file to determine which hosts are allowed to access its client port.

The trusted hosts definition file is a text file which lists all the hosts that have authority to access the client port of a Client Access Bridge server. The connection will be dropped if the host attempting to connect to client port is not listed in the trusted hosts definition file. If the trusted hosts definition file contains at least one entry then all incoming connections must match that entry. If the trusted hosts definition file does not contain any entries then all incoming connections will be accepted.

The trusted hosts definition file is available from the following default location: \$OMNIHOME/etc/NC0_BRIDGE.thosts (UNIX) or %OMNIHOME%\etc\NC0_BRIDGE.thosts (Windows).

Note: The trusted hosts file is checked only for incoming connections to the Client Access Bridge server, it is not checked by the Server Access Bridge server.

Syntax

The trusted hosts definition file accepts entries in a variety of formats: example IP addresses of IPv4 or IPv6, wildcards, and human-readable host name formats. Additionally comments are supported.

```
#
# Trusted Hosts File
#
# IPv4 address - Match this IPv4 address only:
192.168.1.1

# IPv4 address/netmask - Match any IPv4 address within the 192.168.1.0 network:
192.168.1.0/255.255.255.0

# IPv4 CIDR notation - Match any IPv4 address within the 192.168.1.0 network:
192.168.1.0/24

# Hostname - Match that host name only:
darkstar.example.com

# Hostname with wildcard - Match all hosts in the ibm.com domain:
*.ibm.com

# IPv6 address - Match this IPv6 address only:
[3ffe:1900:4545:3:200:f8ff:fe21:67cf]

# IPv6 CIDR notation - Match any IPv6 address within the
3ffe:1900:4545:3:200:f8ff:fe21:0000 network:
[3ffe:1900:4545:3:200:f8ff:fe21:67cf]/120
```

Disabling access to the interactive command port on Client Access Bridge server

For additional security, any connections made to the Client Access Bridge server, using the SQL interactive interface, can be disabled by setting the **Bridge.AllowConnections** property to FALSE, or by running the Client Access Bridge server with the -disallowconnections command line argument. This may

be required if the Client Access Bridge server is running in an non-secure network outside a firewall.

Firewall bridge server command language

The firewall bridge server provides an SQL command interface for configuration and administration purposes. You can use the SQL interactive interface to connect to a firewall bridge server and run firewall bridge server commands.

Before you begin to use the firewall bridge command language, ensure you are familiar with the SQL interactive interface, how to start it, and how to run SQL commands in the SQL interactive interface.

Related concepts

“SQL interactive interface” on page 135

SHOW PROPS and GET CONFIG

Use the SHOW PROPS and GET CONFIG commands to list all the firewall bridge server properties and their associated values.

Syntax

```
SHOW PROPS;  
GET CONFIG;
```

Example

To list all the firewall bridge server properties and their associated values:

```
1> show props;  
2> go  
Property Name                                Type Property Value  
-----  
AuthPassword                                7  
AuthUserName                                7  
Bridge.AllowConnections                      4 TRUE  
Bridge.ClientAP.ClientPort                   3 10001  
Bridge.ClientAP.Hostname                     7 omnihost  
Bridge.ClientAP.ServerPort                   3 10002  
Bridge.PAMEnable                             4 TRUE  
Bridge.PAMServiceName                        7 netcool  
Bridge.Role                                  7 SERVER_AP  
Bridge.ServerAP.Server                       7 NCOMS  
Bridge.ThreadPool.MaxJobs                     3 10  
Bridge.TrustedHostFile                       7 /opt/ibm/netcool/omnibus/etc/NCO_BRIDGE.thosts  
ConfigCryptoAlg                             7 AES  
ConfigKeyFile                                7  
Connections                                  2 30  
Help                                          4 FALSE  
Ipc.QueueSize                               2 1024  
Ipc.ServerCharacterSet                       7 iso_1  
Ipc.ServerLanguage                           7 us_english  
Ipc.ServerLocale                             7 default  
Ipc.SingleThreaded                           4 FALSE  
Ipc.SSLCertificate                           7  
Ipc.SSLEnable                                4 FALSE  
Ipc.SSLPrivateKeyPassword                    7  
Ipc.StackSize                                2 131072  
Ipc.Timeout                                  2 60  
Ipc.TruncateVendorLogFile                    4 TRUE  
Ipc.VendorClientLibraryVersion               7 version string  
Ipc.VendorLogFileSize                        2 1024  
Ipc.VendorServerLibraryVersion               7 version string  
MaxLogFileSize                               2 1024  
MessageLevel                                 7 debug  
MessageLog                                   7 stdout  
Name                                          7 NCO_BRIDGE  
PAAwareID                                    2 0  
PAServerName                                 7  
Props.CheckNames                             4 TRUE
```

```

PropsFile          7    /opt/ibm/netcool/omnibus/etc/NCO_BRIDGE.props
SecureMode         4    FALSE
UniqueLog          4    FALSE
Version            4    FALSE

```

GET PROP

Use the GET PROP command to return the value of a specific firewall bridge server property.

Syntax

```
GET PROP[ERTY] <propname>;
```

Example

To display a firewall bridge server property and its associated value:

```

1> get prop 'Name';
2> go
Property Name                                Type Property Value
-----
Name                                             7    NCO_BRIDGE

```

SHOW DATAFLOWS

Use the SHOW DATAFLOWS command to list the active data-flows across a firewall bridge server.

Syntax

```
SHOW DATAFLOWS;
```

Example

To list active data flows across a firewall bridge server:

```

1> show dataflows;
2> go
TPWorkerName      DataflowID  SPSocketFD  CPSocketFD  SPBytes  CPBytes
-----
TPWorkerThread_0x74340  4033280    16          17         81      615

```

The output represents a single active data flow connection across the firewall bridge server, where each column represents the following:

TPWorkerName	The internal name of thread.
DataflowID	The internal ID of data flow.
SPSocketFD	The operating system file descriptor of the server side of the connection.
CPSocketFD	The operating system file descriptor of the client side of the connection.
SPBytes	The number of bytes transferred from the server to the client connection.
CPBytes	The number of bytes transferred from the client to the server connection.

SET LOG LEVEL TO

Use the SET LOG LEVEL TO command to specify the log level.

Syntax

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

Where *level* takes one of the following values:

debug

information

warning

error

fatal

Example

To list all the firewall bridge server properties and their associated values:

```
1> set log level to DEBUG;  
2> go
```

SHUTDOWN

Use the SHUTDOWN command to instruct the firewall bridge server to shut down.

Syntax

```
shutdown;
```

Example

To shut down the firewall bridge server:

```
shutdown;
```

Chapter 4. Using Netcool/OMNIBus Administrator to configure ObjectServers

The ObjectServer stores, manages, and processes alert and status data that is collected by external applications such as probes and gateways. You can use Netcool/OMNIBus Administrator to configure your ObjectServer objects and to configure process control.

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

- Users, groups, roles, and restriction filters
- Event list menus
- Tools and prompts
- Trigger groups and triggers
- Procedures
- User-defined signals
- Event list alert severity colors (Windows event lists only)
- Conversions
- Classes
- Column visuals
- ObjectServer databases, files, and properties
- Channels for accelerated event notification

Getting started with Netcool/OMNIBus Administrator

Netcool/OMNIBus Administrator provides a visual interface from which you can manage your ObjectServers and configure process control.

Considerations for multicultural support

Tivoli Netcool/OMNIBus supports a variety of single byte and multi-byte character encodings for use in different locales.

If user names and passwords are specified in multi-byte characters and these credentials are to be verified against external authentication sources, then these sources must also support multi-byte characters. If multi-byte characters are not supported, user names and passwords must be specified using ASCII characters.

When using Netcool/OMNIBus Administrator, you must ensure that the character set encoding of each ObjectServer being managed has a corresponding entry in the file `$NCHOME/omnibus/java/jars/csemap.dat`. This file provides a mapping between Sybase and JRE character set encoding naming conventions. If the character set encoding of an ObjectServer is missing from `csemap.dat`, you must add a mapping to this file by using the format:

Sybase_encoding Java_encoding

For example:

`ascii_7 ASCII`

For further information on multicultural support, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*.

Starting Netcool/OMNIBus Administrator

You must run the **nco_config** utility to start Netcool/OMNIBus Administrator.

To start Netcool/OMNIBus Administrator from the command line:

1. Enter the appropriate command for your operating system:

Table 13. Starting Netcool/OMNIBus Administrator

Option	Description
UNIX	<code>\$NCHOME/omnibus/bin/nco_config [-option value ...]</code>
Windows	<code>%NCHOME%\omnibus\bin\nco_config.vbs [-option value ...]</code>

In this command, *-option* is a valid command-line option and *value* is the value you are setting the option to.

2. If this is the first time you are starting Netcool/OMNIBus Administrator, or your communications settings file (`$NCHOME/etc/omni.dat` on UNIX and `%NCHOME%\ini\sql.ini` on Windows) has changed since you last started Netcool/OMNIBus Administrator, the Import Connections Wizard is run. The wizard enables you to choose which ObjectServers and process agents you want to configure using Netcool/OMNIBus Administrator.

Tip: After you have started Netcool/OMNIBus Administrator, you can select **File > Import** at any time to import new communications information from the `omni.dat` file (`sql.ini` on Windows). For more information on configuring component communications, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*.

Netcool/OMNIBus Administrator properties and command-line options

Netcool/OMNIBus Administrator contains a set of properties and command-line options for configuring the component.

The default Netcool/OMNIBus Administrator properties file is `$NCHOME/omnibus/etc/nco_config.props` (`%NCHOME%\omnibus\etc\nco_config.props` on Windows). 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 alternative properties file.

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

The properties and command-line options for **nco_config** (`nco_config.vbs` on Windows) are described in the following table.

Table 14. Netcool/OMNIBus Administrator properties and command-line options

Property	Command-line option	Description
audit.active 0 1	<code>-auditlogactive</code> 0 1	Determines whether audit logging is enabled. The default is 1; audit logging is enabled.

Table 14. Netcool/OMNIBus Administrator properties and command-line options (continued)

Property	Command-line option	Description
audit.file.max.count <i>integer</i>	-auditfilecount <i>integer</i>	The maximum number of Netcool/OMNIBus Administrator audit log files. The default is 4.
audit.file.max.size <i>integer</i>	-auditfilesize <i>integer</i>	The maximum file size (in Bytes) for the Netcool/OMNIBus Administrator audit log files. The default is 10000.
audit.file.name <i>string</i>	-auditfile <i>string</i>	The full path to the Netcool/OMNIBus Administrator audit log file. The default is \$NCHOME/omnibus/log/nco_config_audit.log.
N/A	-help	Displays help on the command-line options and exits.
java.security.policy <i>string</i>	-policyfile <i>string</i>	The full path to the Java security policy file.
log.console.active 0 1	-logtoconsole 0 1	Determines whether logging information is sent to the command shell. The default is 1; logging information is sent to the command shell.
log.directory.name <i>string</i>	-logdir <i>string</i>	The location where the Netcool/OMNIBus Administrator system log file is saved. The default is NCHOME/omnibus/log.
log.file.max.count <i>integer</i>	-logfilecount <i>integer</i>	The maximum number of Netcool/OMNIBus Administrator system log files. The default is 4.
log.file.max.size <i>integer</i>	-logfilesize <i>integer</i>	The maximum file size (in Bytes) for the Netcool/OMNIBus Administrator system log files. The default is 10000.
log.file.name <i>string</i>	-logfile <i>string</i>	The name of the Netcool/OMNIBus Administrator system log file. The default is nco_config_system.log. Tip: To change the directory where this file is stored, use the log.directory.name property or -logdir option.
messagelevel <i>string</i>	-messagelevel <i>string</i>	Specifies the message logging level for system and audit logging. Possible values are: FATAL, ERROR, WARN, INFO, and DEBUG. The default level is ERROR. 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. Note: The value of <i>string</i> must be in uppercase.

Table 14. Netcool/OMNIBus Administrator properties and command-line options (continued)

Property	Command-line option	Description
nco_jdbc.timeout <i>integer</i>	-jdbctimeout <i>integer</i>	The Java Database Connectivity (JDBC) timeout, in seconds. The default is 600.
N/A	-propsfile <i>string</i>	The full path to the Netcool/OMNIBus Administrator properties file. The default is \$NCHOME/omnibus/etc/nco_config.props (%NCHOME%\omnibus\etc\nco_config.props on Windows).
server <i>string</i>	-server <i>string</i>	The name of the ObjectServer to which you are connecting.
system.create.conversion 0 1	-createconversion 0 1	Sets up the system so that it automatically creates conversion for users that you create. The default is 1, which indicates that conversions are automatically created.
system.conversion.type <i>string</i>	-conversiontype <i>string</i>	Creates a conversion between the user ID, and the user name or full name of each newly-created user. Possible values are fullname and username. The default is fullname, which creates a conversion between the user ID and the full name.
user.name <i>string</i>	-user <i>string</i>	The Netcool/OMNIBus Administrator login user name.
user.password <i>string</i>	-password <i>string</i>	The Netcool/OMNIBus Administrator login password.
N/A	-version	Displays version information and then exits.

Property and command-line processing

Each property in the Netcool/OMNIBus Administrator properties file has a default value. In an unedited properties file, properties are listed with their default values, commented out with a number sign (#) at the beginning of the line. A property and its corresponding value are separated by a colon (:).

You can edit the property values by using a text editor. To override the default, change a setting in the properties file and remove the number sign (#).

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 that you enter to run **nco_config**, add as many properties as possible to the properties file rather than using the command-line options.

Connecting to an ObjectServer

After starting Netcool/OMNIBus Administrator, you must connect to the ObjectServer that you want to configure.

To connect to an ObjectServer:

1. From the Netcool/OMNIBus Administrator window, select the **Reports** menu button in the left pane.



Figure 6. Navigator and Reports menu buttons

2. Click **OS**. The ObjectServer Report window opens. It displays all the ObjectServers that were selected when the Import Connections Wizard was last run.
3. Right-click the ObjectServer to which you want to connect. From the pop-up menu, perform either of the following actions:
 - Click **Connect As** if you are connecting for the first time or want to enter updated connection information. You are prompted for a user name and password for the ObjectServer.

If you select the **Always use for this connection** check box, the user name and password are saved and automatically reused for connections to this ObjectServer. If you select the **Use as default** check box, the values specified for the user name and password are automatically filled in next time the connection window is displayed. If you select both check boxes, the **Always use for this connection** user name and password values take precedence. These settings last for the length of the application session.
 - Click **Connect** to use previously-specified connection information.
4. Click **OK**.

Results

The pane in the center is the ObjectServer configuration window. In this work area you can view, modify, and manage ObjectServer objects.

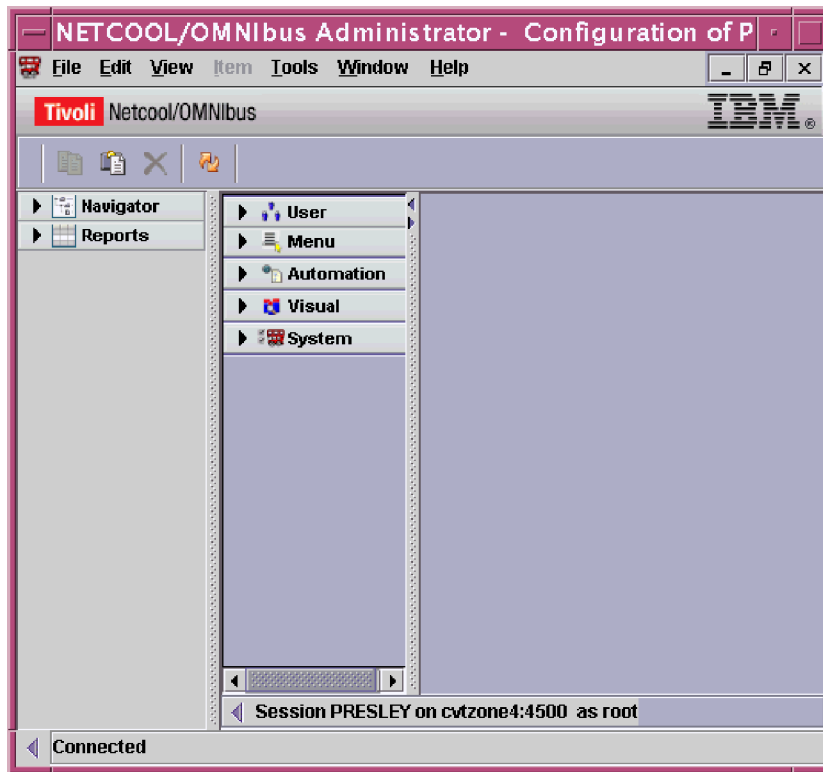


Figure 7. Netcool/OMNIBus Administrator - ObjectServer configuration window

Related tasks

“Selecting ObjectServer objects to configure” on page 67

Connecting to a process agent

Before you can connect to a process agent, you must ensure that the process agent has started.

To connect to a process agent:

1. From the Netcool/OMNIBus Administrator window, select the **Reports** menu button in the left pane.



Figure 8. Navigator and Reports menu buttons

2. Click **PA**. The Process Agent Report pane opens, and shows all the process agents that were selected when the Import Connections Wizard was last run.

Tip: After you have started Netcool/OMNIBus Administrator, you can click **File > Import** at any time to import new communications information from the `omni.dat` file (`sql.ini` on Windows). For more information on configuring component communications, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*.

3. Right-click the process agent. From the pop-up menu, perform either of the following actions:

- Click **Connect As** if you are connecting for the first time or want to enter updated connection information. The Process Agent Security window opens. Complete this window as follows:

Username

Type the user name that is used to log into the process agent.

On UNIX, any user that needs access to the process agent must be a member of a UNIX user group that you identify as an administration group for this purpose. On Windows, the user must be a valid user with a local or domain account.

Password

Type the password that is used to log into the process agent.

Always use for this connection

Select this check box to indicate that the specified user name and password should be saved for automatic reuse on subsequent connection attempts to this process agent. These settings last for the length of the application session.

Use as default

Select this check box if you want the values specified for the user name and password to be automatically filled in the next time this window is displayed. These settings last for the length of the application session.

Note: If you select both check boxes, the **Always use for this connection** setting takes precedence.

- Click **Connect** to use previously-specified connection information.
4. Click **OK**.
- The Service/Process Details window opens. This window contains information about the processes and services configured under this process agent.

Related concepts

Chapter 7, “Using process control to manage processes and external procedures,” on page 249

Working with Tivoli Netcool/OMNIBus components

From the button bar in the left-hand pane of the Netcool/OMNIBus Administrator window, you must select the components that you want to view or configure.

- Select the **Reports** menu button, and then click **Host**, **OS**, or **PA**. Select an ObjectServer to configure from the ObjectServer Report window, or a process agent to configure from the PA Report window.
- Select the **Navigators** menu button to view the components by host name, and then select the component that you want to configure.


The following figure shows the menu buttons used for selecting a component.



Figure 9. Navigator and Reports menu buttons

When you select a component, the associated window or pane is shown in the display area on the right. The available toolbar buttons and menu items in the Netcool/OMNIBus Administrator window change depending on your selection.

Secure sockets layer connections

If Netcool/OMNIBus Administrator is using an encrypted SSL connection to an ObjectServer or process agent, a lock icon  appears in the bottom, left corner of the window.

For information about using SSL with Tivoli Netcool/OMNIBus, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*.

Validating server certificates

When Tivoli Netcool/OMNIBus is set up for SSL communication, the ObjectServer and process agent present their server certificates to the Netcool/OMNIBus Administrator client, on request, to establish a connection.

If a mismatch is detected between the common name defined in the server certificate and the server name that the Netcool/OMNIBus Administrator client uses to identify and connect to the server, a Certificate Validation window opens so that you can choose whether to accept or reject the server certificate. Connections will not be established if the certificate is invalid.

The Certificate Validation window provides a reason for the validation request and presents a number of options. Complete the window as follows:

1. Select one of the options to accept or reject the certificate:
 - **Accept this certificate permanently:** Select this option to permanently accept this certificate as valid. You will no longer be prompted to accept this certificate during the current or subsequent Netcool/OMNIBus Administrator sessions.

Important: Before you accept the certificate, click **Examine Certificate** to review the contents of the certificate within the Certificate Details window. After careful examination, click **OK** to return to the Certificate Validation window.
 - **Accept this certificate temporarily for this session:** Select this option to accept the certificate for the current session only, after examining the certificate by using the **Examine Certificate** button. No more validation prompts will be generated for the duration of the session.
 - **Do not accept this certificate:** Select this option to reject the certificate and cancel the connection between the server and client.
2. Click **OK** to continue with the connection process. Click **Cancel** (or the **Close** button in the title bar) to reject the certificate irrespective of the option that you selected in step 1.

Results

If you chose to accept the certificate permanently, the common name and public key from the certificate are recorded in the following file:

```
userdir/.netcool/nco_config_settings/user_allowed_certs.properties
```

In this file path, *userdir* represents your home directory.

The `user_allowed_certs.properties` file is a system file and is not intended for modification by users. On subsequent connection attempts, this file is read and used to identify any common names that were previously accepted.

You can clear the contents of the properties file by specifying the following command-line argument:

```
mode.clear.certs "true"
```

For further information about SSL certificates, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*.

Selecting ObjectServer objects to configure

A set of menu buttons are available on the left side of the Netcool/OMNIBus Administrator window for selecting the ObjectServer object that you want to configure.

These menu buttons are labeled as follows:

- User
- Menu
- Automation
- Visual
- System

Results

The following figure shows these buttons.

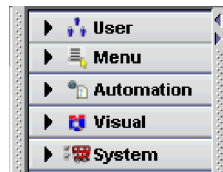


Figure 10. User, Menu, Automation, Visual, and System menu buttons

Click each menu button to obtain a set of related objects that you can configure. The available toolbar buttons and menu items in the Netcool/OMNIBus Administrator window change depending on your selection.

Working with objects

When working with an object, several ways are available for you to select an option: the menu bar, the toolbar, and a pop-up menu. For example, if you want to create a user, you must first select the **User** menu button and then click **Users** to obtain the Users pane. You can then perform any of the following actions to create the user:

- Click **Item > Add User**.
- From the toolbar, click **Add User**.
- From the **Users** pane, right-click and then click **Add User** from the pop-up menu.

If you are editing or deleting objects, you can perform an equivalent set of actions. If you are editing objects, you can additionally double-click the object to open the relevant window for editing.

Tip: You can also use SQL commands to interact with, and configure, the ObjectServer.

Setting preferences in Netcool/OMNIBus Administrator

You can set preferences in Netcool/OMNIBus Administrator by sorting results tables, setting column display appearance using views, and selecting rows to display using filters. You can also copy and paste, configure editor syntax coloring, and select a web browser for displaying online help.

Sorting results tables

Many Netcool/OMNIBus Administrator windows (including the Host Report window, PA Report window, ObjectServer Report window, SQL results, and database tables) display information as results tables.

You can click a column title to sort the rows of the table by the values of that column. Click multiple times to select either an ascending or descending sort, indicated by the up or down arrow next to the column title.

Setting column display appearance using views

You can change how columns are displayed in results tables, which include the Host Report window, PA Report window, ObjectServer Report window, SQL results, and database tables.

To hide a column, right click the column header and click **Hide Column**.

To align a column, right-click the column header and click **Align Column**. Then click **Left**, **Right**, or **Center**.

To select which columns to display from a list of all columns:

1. Right-click the column header and click **Table Columns**. The Column Settings window opens.
2. Select the columns that you want to view in the current window. You can click a column name to select or deselect it. A tick is displayed next to columns that are selected for display. Click **Close** to save your settings.

Selecting rows to display using filters

You can change which rows are displayed in results tables, which include the Host Report, PA Report, and ObjectServer Report windows, SQL results, and database tables, by creating filter conditions.

To create a filter:


1. Right click the column header for the column to which you wish to apply a filter and click **Filter On**. The Filter window opens.
2. Select the column, filter operation, and value for the condition.

On the Filter window, you can:

- Click **OK** to create the filter and close the window
- Click **Apply** to create the filter and keep the window open
- Click **Cancel** to close the window without creating a filter

To create a filter based on the values in a particular table cell:

1. Right click in the cell and select the filter icon. A sub-menu appears with matching the column name and value and a selection of either equals or not equals operators.
2. Select the appropriate condition for the filter.

When a filter is applied to a results table, the filter icon  is displayed in the status bar. For each column to which a filter is applied, the icon also appears in the column header.

To see all filter conditions that have been applied to the table, right click the column header and click **Filter Details**. The Filter Settings window displays all filter conditions that are currently applied to the table.

To remove all filters, right click the column header and click **See All Values**.

Copying and pasting

You can use the **Copy** and **Paste** functions in the **Edit** menu 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:

- Users (in the same ObjectServer only)
- Restriction filters
- Menus and menu items
- Tools
- Prompts
- Triggers
- Procedures
- User-defined signals
- Colors
- Column visuals
- ObjectServer files
- Databases
- Tables
- Processes (under process control)
- Services (under process control)

Configuring colors for syntax elements in the default SQL editors

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

To change the default syntax coloring scheme:

1. From the Netcool/OMNIbus Administrator window, click **Tools > Editor Options**. The Highlight Details window opens.
2. Complete this window as follows:

Element

Select the syntax element for which you want to view or change the color.

Color This field displays the currently-selected color for the element. 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 element.

3. Save or cancel your changes as follows:

OK Click this button to save the color details and close the window.

Cancel

Click this button to close the window without saving your changes.

Selecting a Web browser for displaying online help

In Netcool/OMNIbus Administrator, you must select a Web browser in which you want to view the online help. The online help is deployed using IBM® Eclipse Help System, on which a number of browsers are supported.

For a list of supported online help Web browsers, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

To select the Web browser for displaying online help:

1. From Netcool/OMNIbus Administrator, select **Tools > Configure Tools**. The Choose Tool window opens.
2. Select **Browser**. The External Program window opens.
3. Complete this windows as follows:

Tool name

This field displays the type of tool that you are configuring.

Executable

Type the full path and name of the executable program for the tool type. Alternatively, click the button to the right of the field to search for and select the executable program.

Arguments

Type any command-line arguments to run with this executable program.

Run time environment

Select the runtime environment.

Note: The **Run time environment** field is not available if you are configuring a Web browser.

4. Save or cancel your changes as follows:

OK Click this button to save the external program details and close the window.

Cancel

Click this button to close the window without saving your changes.

Exiting Netcool/OMNIBus Administrator

To exit Netcool/OMNIBus Administrator, click **File > Exit**.

Managing authorization with users, groups, roles, and restriction filters

Authorization is the verification of the rights to view and modify information.

Access to ObjectServer objects is controlled 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.

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

For example, creating automations requires knowledge of Tivoli 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 triggers, trigger groups, files, SQL procedures, external procedures, and signals. You can then grant that role to a group of administrators who will be creating and updating triggers.

To set up Tivoli Netcool/OMNIBus authorization, configure security objects in the following order:

1. Roles: Assign permissions to roles.
2. Groups: Assign one or more roles to each group. The assigned roles determine the actions that the group members can perform on database objects.
3. Users: Add users to groups. You must assign each user to one or more groups.

Results

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.

Configuring roles

Roles are collections of permissions that you can assign to users and groups.

Tivoli Netcool/OMNIBus provides a number of default roles, and enables you to create custom roles for association with users and groups. The default roles are described in the following table.

Table 15. Default roles

Role name	Description
CatalogUser	<p>This role includes permissions to view information about system, tools, security, and desktop database tables.</p> <p>This role provides a basis for Tivoli Netcool/OMNIbus permissions. This role does not provide sufficient permissions to use any Tivoli Netcool/OMNIbus applications.</p> <p>All groups should be assigned this role.</p>
AlertsUser	<p>This role includes the following permissions:</p> <ul style="list-style-type: none"> • View, update, and delete entries in the alerts.status table • View, insert, and delete entries in the alerts.journal table • View and delete entries in the alerts.details table <p>This role, in combination with the CatalogUser role, enables you to display and manipulate alerts, create filters and views, and run standard tools in the event list.</p>
AlertsProbe	<p>This role includes permissions to insert and update entries in the alerts.status table, and insert entries in the alerts.details table.</p> <p>This role, in combination with the CatalogUser role, provides the permissions that a probe needs to generate alerts in the ObjectServer. These permissions should be granted to any user that runs a probe application.</p>
AlertsGateway	<p>This role includes permissions to insert, update, and delete entries in the alerts.status table, alerts.details table, alerts.journal table, alerts.conversions table, alerts.col_visuals table, alerts.colors table, the desktop tools tables, and the tables in the transfer database. The transfer database is used internally by the bidirectional ObjectServer Gateway to synchronize security information between ObjectServers.</p> <p>This role also includes permissions to select, insert, update, and delete entries in the master.servergroups table, and permissions to raise the following signals: gw_counterpart_down, gw_counterpart_up, gw_resync_start, and gw_resync_finish.</p> <p>This role, in combination with the CatalogUser role, provides the permissions that a gateway needs to generate alerts in the ObjectServer. These permissions should be granted to any user that runs a gateway application.</p>
DatabaseAdmin	<p>This role includes permissions to create databases and files, and to create tables in the alerts, tools, and service databases. This role also includes permissions to modify or drop the alerts.status, alerts.details, and alerts.journal tables, and permissions to create and drop indexes in the alerts.status, alerts.details, and alerts.journal tables.</p> <p>This role, in combination with the CatalogUser role, provides permissions to create relational data structures in the ObjectServer.</p>

Table 15. Default roles (continued)

Role name	Description
AutoAdmin	<p>This role includes permissions to create trigger groups, files, SQL procedures, external procedures, and user signals. This role also includes permissions to create, modify, and drop triggers in the default trigger groups, and to modify or drop default trigger groups.</p> <p>This role, in combination with the CatalogUser role, provides permissions to create automations in the ObjectServer.</p>
ToolsAdmin	<p>This role includes permissions to delete, insert, and update all tools tables.</p> <p>This role, in combination with the CatalogUser role, provides permissions to create and modify tools that can be run from the desktop and Netcool/OMNIBus Administrator .</p>
DesktopAdmin	<p>This role includes permissions to update all desktop catalogs to insert, update, and delete colors, visuals, menus, classes, resolutions, and conversions.</p> <p>This role, in combination with the CatalogUser role, provides permissions to customize the desktop.</p>
SecurityAdmin	<p>This role, in combination with the CatalogUser role, includes permissions to manipulate users, groups, and roles by using Netcool/OMNIBus Administrator or the SQL interactive interface. This role also includes permissions to set properties and drop user connections.</p>
ISQL	<p>This role, in combination with the CatalogUser role, includes permission to view ObjectServer data by using the SQL interactive interface.</p>
ISQLWrite	<p>This role, in combination with the CatalogUser role, includes permissions to view and modify ObjectServer data by using the SQL interactive interface.</p>
SuperUser	<p>This role has all available permissions. You cannot modify the SuperUser role.</p>
Public	<p>All users are assigned this role. By default, the Public role is not assigned any permissions. You can modify, but not drop, the Public role.</p>
ChannelAdmin	<p>This role includes permissions to set up channels for accelerated event notification.</p>
ChannelUser	<p>This role includes permissions to receive and act on notifications for accelerated events that are broadcast over channels.</p>

Roles determine the types of tasks that users in the group can perform. For example, if you assign the AlertsUser role to a group, users in that group are granted the following permissions:

- View, update, and delete entries in the alerts.status table
- View, insert, and delete entries in the alerts.journal table
- View and delete entries in the alerts.details table

Related concepts

“Configuring groups” on page 76

“Configuring users” on page 79

Creating and editing roles

Use roles to assign permissions to users who are members of a particular group.

Before you begin

You must have started Netcool/OMNIBus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIBus Administrator” on page 60.

To create or edit a role:

1. From the Netcool/OMNIBus Administrator window, select the **User** menu button.
2. Click **Roles**. The Roles pane opens.
3. To add a role, click **Add Role** in the toolbar. The Role Details window opens.
4. To edit a role, select the role to edit and then click **Edit Role** in the toolbar. The Role Details window opens.
5. Define a new role as follows:

Role Name

Type a unique name for the role. If you are editing a role, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Role ID

Specify a unique numerical identifier for this role. If you are editing a role, you cannot change the role ID.

6. From the **Details** tab, enter a meaningful description for the role or update the description.
7. From the **Permissions** tab, grant permissions to the role or revoke permissions, as follows:

Add permission

Click this button to grant permissions to the role. The Permission Objects window opens. Complete this window as follows:

Object Type

Select the type of object to which you want to grant or revoke permissions.

Available Objects

The contents of this list are dependent on the object type selected. The **Available Objects** list displays the objects for which permissions have not been granted. To grant permissions to one or more objects, use the arrow keys to move the objects to the **Selected Objects** list.

To move all objects to the **Selected Objects** list, click >>. To move a single object or multiple objects to the **Selected Objects** list, select each object and then click >. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Selected Objects

This list displays the objects for which permissions have been granted. To revoke permissions for one or more objects, use the arrow keys to move the objects to the **Available Objects** list.

To move all objects to the **Available Objects** list, click <<. To move a single object or multiple objects to the **Available Objects** list, select each object and then click <. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Tip: You can add multiple object permissions by selecting each required object type in turn from the **Object Type** list, and then adding the object to the **Selected Objects** list.

OK Click this button to save the permissions configured for the role, and close the Permission Objects window.

Cancel Click this button to close the window without saving your changes.

When you return to the **Permissions** tab on the Role Details window, the permissions tree is updated with your changes. The parent node shows the main object, and the nested child nodes show associated subobjects. You can expand these subobjects to view the associated SQL permissions; for example ALTER, DELETE, and DROP. Select the check box associated with the SQL permission to assign that permission to the role. Clear the check box to revoke the permission.

Delete permission

From the permissions tree, select a parent or child node for which you want to revoke permissions, and then click this button. The node is deleted from the tree.

8. Save or cancel your changes as follows:

OK Click this button to save the role details and close the window. New roles are added to the Roles pane.

Cancel Click this button to close the window without saving your changes.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

Deleting roles

Before you begin

You must have started Netcool/OMNIBus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIBus Administrator” on page 60.

To delete a role:

1. From the Netcool/OMNIBus Administrator window, select the **User** menu button.
2. Click **Roles**. The Roles pane opens.
3. Select the role that you want to delete and click **Delete** in the toolbar. The role is deleted. If this role was assigned to a group, the role is removed from the group.

Related tasks

“Starting Netcool/OMNIBus Administrator” on page 60

Configuring groups

Use groups to organize Tivoli Netcool/OMNIBus users into units with common functional goals. All members of a group have the permissions assigned to the group roles.

Tivoli Netcool/OMNIBus provides a number of default groups, and enables you to create custom groups. The default groups are described in the following table.

Table 16. Default groups

Group name	Description
Probe	This group is assigned the CatalogUser, AlertsUser, and AlertsProbe roles.
Gateway	This group is assigned the CatalogUser, AlertsUser, and AlertsGateway roles.
ISQL	This group is assigned the ISQL role.
ISQLWrite	This group is assigned the ISQLWrite role.
Public	This group is assigned the Public role. All users are members of this group.
Normal	This group is assigned the CatalogUser, AlertsUser, and ChannelUser roles. This group cannot be deleted or renamed.
Administrator	This group is assigned the CatalogUser, AlertsUser, ToolsAdmin, DesktopAdmin, ChannelUser, and ChannelAdmin roles. This group cannot be deleted or renamed.
System	This group is assigned the CatalogUser, AlertsUser, ToolsAdmin, DesktopAdmin, AlertsProbe, AlertsGateway, DatabaseAdmin, AutoAdmin, SecurityAdmin, ISQL, ISQLWrite, SuperUser, ChannelUser, and ChannelAdmin roles. This group cannot be deleted or renamed.

Related concepts

“Configuring roles” on page 71

“Configuring users” on page 79

Creating and editing groups

Before you begin

You must have started Netcool/OMNIBus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIBus Administrator” on page 60.

To create or edit a group:

1. From the Netcool/OMNIBus Administrator window, select the **User** menu button.
2. Click **Groups**. The Groups pane opens.
3. To add a group, click **Add Group** in the toolbar. The Group Details window opens.
4. To edit a group, select the group to edit and then click **Edit Group** in the toolbar. The Group Details window opens.
5. Define the group as follows:

Group Name

Type a unique name for the group. If you are editing a group, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Group ID

Specify a unique numerical identifier for the group. If you are editing a group, you cannot change the group ID.

Description

Type a meaningful description for the group, or update the description.

6. From the **Roles** tab, specify the roles that you want to assign to the group. When you assign a role to a group of users, any user added to that group assumes the security permissions granted to the role. Complete the tab as follows:

Available Roles

This list displays the available roles that you can assign to the group. To assign one or more of these roles, use the arrow keys to move the roles to the **Applied Roles** list.

To move all roles to the **Applied Roles** list, click >>. To move a single role or multiple roles to the **Applied Roles** list, select each role and then click >. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Applied Roles

This list displays the roles that are already assigned to the group. To unassign roles, use the arrow keys to move the roles to the **Available Roles** list.

To move all roles to the **Available Roles** list, click <<. To move a single role or multiple roles to the **Available Roles** list, select each role and then click <. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Add new role

Click this button if you want to create a new role that can then be assigned to this group. The Role Details window opens. Complete this window and save your changes.

When you return to the **Roles** tab, the new role will be added to the **Available Roles** list. You can then add this role to the group.

7. From the **Restriction Filters** tab, specify any restriction filters that you want to assign to the group. You can use restriction filters to prevent the group of users from viewing or modifying certain rows in ObjectServer tables. Complete the tab as follows:

All Restriction Filters

This list displays the available restriction filters that have been set up. To assign a restriction filter to the group, select the restriction filter row and then click **Add Restriction Filter**. The selected restriction filter is transferred to the **Assigned Restriction Filters** list.

If no restriction filters have previously been set up, or if you want to create a new restriction filter and assign it to the group, click **New Restriction Filter**. The Restriction Filter Details window opens. Complete this window and save your changes.

When you return to the **Restriction Filters** tab, the new restriction filter will be added to the **All Restriction Filters** list. You can then assign this restriction filter to the group.

Assigned Restriction Filters

This list displays the restriction filters that are assigned to the group. To unassign a restriction filter, select the restriction filter row and then click **Remove Restriction Filter**. The selected restriction filter is transferred to the **All Restriction Filters** list.

8. From the **Users** tab, specify the users that you want to add as group members. Complete the tab as follows:

Non members

This list displays the available users that you can assign as members of the group. To assign one or more of these users, use the arrow keys to move the users to the **Members** list.

To move all users to the **Members** list, click >>. To move a single user or multiple users to the **Members** list, select each user and then click >. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Members

This list displays the users who are already assigned to the group. To unassign users, use the arrow keys to move the users to the **Non members** list.

To move all users to the **Non members** list, click <<. To move a single user or multiple users to the **Non members** list, select each user and then click <. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Add new user

Click this button if you want to create a new user that can then be assigned to this group. The User Details window opens. Complete this window and save your changes.

When you return to the **Users** tab, the new user will be added to the **Non members** list. You can then add this user to the group.

9. Save or cancel your changes as follows:

OK Click this button to save the group details and close the window. New groups are added to the Groups pane.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Creating and editing roles” on page 74

“Creating and editing restriction filters” on page 84

“Creating and editing users” on page 80

Deleting groups

You cannot delete the Normal, Administrator, and System groups.

Before you begin

You must have started Netcool/OMNIbus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIbus Administrator” on page 60.

To delete a group:

1. From the Netcool/OMNIbus Administrator window, select the **User** menu button.
2. Click **Groups**. The Groups pane opens.
3. Select the group that you want to delete and click **Delete** in the toolbar. The group is deleted.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

Configuring users

You can create and modify Tivoli Netcool/OMNIbus users, and organize these users into groups. You can assign roles to the user groups, to control access to ObjectServer objects.

Tivoli Netcool/OMNIbus provides a set of default user accounts. The default users are described in the following table.

Table 17. Default users

User name	Description
root	This user is created with an empty string as a password by default. You can reset the password by using Netcool/OMNIBus Administrator , or the ALTER USER ObjectServer SQL command.
nobody	This user is disabled and cannot be used to access the ObjectServer. 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.

Related concepts

“Configuring groups” on page 76

“Configuring roles” on page 71

Creating and editing users

When setting up users, you must assign them to groups with allocated roles. This determines the user permissions.

Before you begin

You must have started Netcool/OMNIBus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIBus Administrator” on page 60.

To create or edit a user:

1. From the Netcool/OMNIBus Administrator window, select the **User** menu button.
2. Click **Users**. The Users pane opens.
3. To add a user, click **Add User** in the toolbar. The User Details window opens.
4. To edit a user, select the user to edit and then click **Edit User** in the toolbar. The User Details window opens.
5. Define the user as follows:

Username

Type a unique name for the user. If the user is to be externally authenticated, for example, in a Lightweight Directory Access Protocol (LDAP) repository or by using Pluggable Authentication Modules (PAM), the name entered in this field must be identical to the name stored in the external authentication repository. If you are editing a user, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

User ID

Specify a unique numerical identifier for the user. This should be set to match the UNIX UID where possible. If you are editing a user, you cannot change the user ID.

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 2147483647.

Full Name

Type the full name of the user.

Create Conversion

Select this check box if you want to create a conversion for the user.

This causes either the user name or the full name of the user, to be displayed in Tivoli® Netcool/OMNIBus event lists. The name shown is determined by the value of the **system.conversion.type** property in the Netcool/OMNIBus Administrator properties file NCHOME/omnibus/etc/nco_config.props. (This property creates a conversion between the user ID, and the user name or full name of each newly-created user.)

6. From the **Groups** tab, specify the groups to which the user should belong. Complete the tab as follows:

Unassigned Groups

This list displays the available groups to which you can assign the user. To assign the user to one or more of these groups, use the arrow keys to move the groups to the **Assigned Groups** list.

To move all groups to the **Assigned Groups** list, click >>. To move a single group or multiple groups to the **Assigned Groups** list, select each group and then click >. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Note: If you do not add the user to a group, the user will have no permissions.

Assigned Groups

This list displays the groups to which the user is assigned. To unassign groups, use the arrow keys to move the groups to the **Unassigned Groups** list.

To move all groups to the **Unassigned Groups** list, click <<. To move a single group or multiple groups to the **Unassigned Groups** list, select each group and then click <. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

Add new group

Click this button if you want to create a new group to which the user can then be assigned. The Group Details window opens. Complete this window and save your changes.

When you return to the **Groups** tab in the User Details window, the new group will be added to the **Unassigned Groups** list. You can then assign the user to this group.

7. From the **Restriction Filters** tab, specify any restriction filters that you want to assign to the user. You can use restriction filters to prevent the user from viewing or modifying certain rows in ObjectServer tables. Complete the tab as follows:

All Restriction Filters

This list displays the available restriction filters that have been set up. To assign a restriction filter to the user, select the restriction filter row and then click **Add Restriction Filter**. The selected restriction filter is transferred to the **Assigned Restriction Filters** list.

If no restriction filters have previously been set up, or if you want to create a new restriction filter and assign it to the user, click **New Restriction Filter**. The Restriction Filter Details window opens. Complete this window and save your changes.

When you return to the **Restriction Filters** tab, the new restriction filter will be added to the **All Restriction Filters** list. You can then assign this restriction filter to the user.

Assigned Restriction Filters

This list displays the restriction filters that are assigned to the user. To unassign a restriction filter, select the restriction filter row and then click **Remove Restriction Filter**. The selected restriction filter is transferred to the **All Restriction Filters** list.

8. From the **Settings** tab, specify authentication details for the user and make the user active on the system.

Password

Type an optional password for the user. Asterisks (*) appear in place of the password characters that you type.

If the user is to be externally authenticated, the password is stored in the external repository, so leave this field blank.

Verify If you typed a password in the **Password** field, retype the password here.

Change

Click this button to reset the user password. You must then retype it in the **Password** and **Verify** fields.

This button is visible only when editing a user.

External Authentication

Select this check box to externally authenticate the user. Clear this check box to store the user name and associated password in the ObjectServer, and to perform ObjectServer authentication.

User Type

This read-only field indicates the type of user. This is set automatically based on the groups to which the user belongs.

User Enabled

Select this check box to activate the user account and allow login access to Tivoli Netcool/OMNIBus. New user accounts are activated by default.

Clear this check box to create the user account without activating it. You might choose to do this if you want to deny login access to the user until all the appropriate permissions are assigned to them.

9. Save or cancel your changes as follows:

OK Click this button to save the user details and close the window. New users are added to the Users pane.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Creating and editing groups” on page 77

“Creating and editing restriction filters” on page 84

Deleting users

Before you begin

You must have started Netcool/OMNIbus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIbus Administrator” on page 60.

To delete a user:

1. From the Netcool/OMNIbus Administrator window, select the **User** menu button.
2. Click **Users**. The Users pane opens.
3. Select the user that you want to delete and click **Delete** in the toolbar. The user is deleted.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

Viewing user connections to the ObjectServer


You can view the connection details of any user that is currently logged on to the ObjectServer.

Before you begin


You must have started Netcool/OMNIbus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIbus Administrator” on page 60.

To view user connections:

1. From the Netcool/OMNIbus Administrator window, select the **User** menu button.
2. Click **Users**. The Users pane opens, showing a row for each user that is set up on the system.

A status indicator icon  is shown to the left of the user name. The color of the icon depicts the connection status of the user as follows:

- A green color indicates that the user is connected.
- A blue color indicates that the user is enabled, but not connected.
- A gray color indicates that the user is not enabled.

From the Users pane, you can view the connection details for any selected user who is currently logged in to the ObjectServer. For example, if you want to see how many users are currently connected as root, and their connection details, you can select the row for the root user. Right-click over the selected row, and then click **See connections** from the pop-up menu. The ObjectServer Connections pane opens. A dynamic quick filter is automatically applied to the Login Name column so that only applications to which the selected user is connected are shown. The **Quick Filter** icon  is shown in the Login Name

column header to indicate that a filter has been applied to that name. You can remove this filter to show all connections for all users by clicking **Connections** under the **System** menu button.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Monitoring ObjectServer connections” on page 130

Configuring restriction filters

You can use restriction filters to prevent users from viewing or modifying certain rows in ObjectServer tables.

You can assign restrictions filters to individual users or to a collection of users in a group. After you apply a restriction filter to one or more users, the restriction filter controls the data that the users can view and modify in Tivoli Netcool/OMNIbus client applications, and the data that the users can modify in INSERT, UPDATE, and DELETE statements. You can assign only one restriction filter per ObjectServer 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.

If you are using multiple restriction filters, make sure that you have set the ObjectServer **RestrictionFiltersAND** property appropriately.

Related reference

“ObjectServer properties and command-line options” on page 3

Creating and editing restriction filters

Before you begin

You must have started Netcool/OMNIbus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIbus Administrator” on page 60.

To create or edit a restriction filter:

1. From the Netcool/OMNIbus Administrator window, select the **User** menu button.
2. Click **Restriction Filters**. The Restriction Filters pane opens.
3. To add a restriction filter, click **Add Restriction Filter** in the toolbar. The Restriction Filter Details window opens.
4. To edit a restriction filter, select the restriction filter to edit and then click **Edit Restriction Filter** in the toolbar. The Restriction Filter Details window opens.
5. Complete this window as follows:

Name Type a unique name for the restriction filter.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Database

Select the database for which you are creating the restriction filter.

Table Select the table for which you are creating the restriction filter.





Condition

Type the SQL condition (WHERE statement) for the restriction filter. For example: `Tally > 100 AND Severity >=4`

This example condition prevents a user or group to which the restriction filter is applied from seeing alerts in the event list if they occur more than 100 times and have a severity greater than or equal to 4.

You can use the SQL helper buttons to help you create the filter condition, as described in the following table.

Table 18. Buttons for creating filter conditions

Button	Description
	Click this button to select a table column name to add to the command. The column name is substituted for the corresponding event list row value when the tool runs.
	Click this button to select from a list of available conversions.
	Click this button to bring up a list of keywords that complete the entered SQL. Tip: Alternatively, you can type one or more characters and then press Ctrl+F1 to obtain a dialog box with a list of keywords that might possibly match your entry. Select the required keyword and click OK to complete your entry. If only one keyword matches your typed characters, the keyword is automatically completed for you. If you press Ctrl+F1 after typing a database-related keyword, the dialog box provides a list of possible ObjectServer databases from which you can select. If you press Ctrl+F1 after typing a database name followed by a dot (for example: <code>alerts.</code>), you can press Ctrl+F1 again to view and select from a list of tables in the database.
	Click this button to check the validity of the entered SQL syntax.

OK Click this button to save the restriction filter details and close the window. New restriction filters are added to the Restriction Filters pane.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Creating and editing conversions” on page 118

Deleting restriction filters

You cannot delete restriction filters that are currently assigned to a user or group.

Before you begin

You must have started Netcool/OMNIBus Administrator by running the **nco_config** utility. For more information, see “Starting Netcool/OMNIBus Administrator” on page 60.

To delete a restriction filter:

1. From the Netcool/OMNIBus Administrator window, select the **User** menu button.
2. Click **Restriction Filters**. The Restriction Filters pane opens.
3. Select the restriction filter that you want to delete and click **Delete** in the toolbar. The restriction filter is deleted.

Related tasks

“Starting Netcool/OMNIBus Administrator” on page 60

Configuring menus, tools, and prompts

You can configure menus, tools, and prompts for the Tivoli Netcool/OMNIBus desktop (event list and Conductor).

Each menu consists of a menu name and a list of menu items. Menu items are tools, separators, and submenus. Submenus are used within menus to group two or more menu items together.

Tools allow you to control alert management functions within Tivoli Netcool/OMNIBus. Each tool has an associated SQL statement (called an *internal effect*), an executable (called an *external effect*), or both. You can associate tools with a class or classes of alert, and can group tools in tools menus.

A tool can include a prompt window or a pop-up menu for the user to enter information.

Customizing menus

Use Netcool/OMNIBus Administrator to customize the Tivoli Netcool/OMNIBus desktop menus for the event list and Conductor.

You can customize a menu in any of the following ways:

- Add tools, submenus, and separators to a menu
- Edit menu items
- Change the order of menu items
- Remove tools, submenus, or separators from a menu

After customizing a menu, you can preview its structure to see how it will appear in the event list or Conductor.

The following table shows the menus you can customize.

Table 19. Customizable event list menus

Menu name in the Netcool/OMNIBus Administrator - Menus window	Menu name in the Tivoli Netcool/OMNIBus desktop
AlertsMenu	Alerts menu and pop-up menu on the event list, when an alert is selected
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, submenus, and separators to a menu

You can add tools, submenus, and separators as menu items in a desktop menu.

To add a tool, submenu, or separator to a menu:

1. From the Netcool/OMNIBus Administrator window, select the **Menu** menu button.
2. Click **Menus**. The Menus pane opens.

Tip: You can double-click the menu to see existing menu items and tools. You can also click the round symbol (on UNIX) or the plus (+) symbol (on Windows) that is to the left of a menu.

3. From the tree structure, select the menu to which you want to add a menu item, and then click **Add Item** in the toolbar. The Menu Item Details window opens.
4. Complete this window as follows:

Menu Item Type

If you are adding a menu item, select the type of menu item that you want to add to the selected menu. If you are editing a tool or submenu, this field is read-only.

Note: If you are adding a separator, this is the only entry required within this window.

Tool Select the tool that you want to add. If you are editing a tool, you can choose to select another tool from this drop-down list.

Note: This field is available for tools only.

If you want to create a new tool that can then be added as a menu item within the selected menu, click **Add Tool**. The Tool Details window opens. Complete this window and save your changes. When you return to the Menu Item Details window, the tool will be available for selection within the **Tool** drop-down list.

If you want to edit the settings of the selected tool, click **Edit Tool**. The Tool Details window opens. Complete this window and save your changes. You return to the Menu Item Details window.

Title If you are adding or editing a tool or a submenu, either accept the default text (if given) in this field, or type another title. Include an ampersand (&) in the title if you want to create a mnemonic. This text will be shown as the menu item name within the menu.

Note: If you are adding a tool, the **Title** field defaults to the name of the tool selected. However, if you subsequently select a different tool, the initial **Title** field entry will remain unchanged unless you manually update it.

5. Save or cancel your changes as follows:

OK Click this button to save the menu item details and close the window. The Menus pane is updated to reflect the changes made. Your changes are also reflected in the relevant event list or Conductor menu the next time the desktop is started or the event list is resynchronized with the ObjectServer.

Cancel Click this button to close the window without saving your changes.

What to do next

Tip: If you added a submenu to the selected menu, you must now add menu items to the submenu. Select the submenu within the Menus pane and then follow step 3 on page 87, step 4 on page 87, and step 5 for each menu item to be added.

Related tasks

“Creating and editing tools” on page 91

Editing menu items

You can change the title of tools and submenus that are included on the event list and Conductor menus. You can also replace one tool with another, and can edit the tool definition.

To edit a menu item:

1. From the Netcool/OMNIBus Administrator window, select the **Menu** menu button.
2. Click **Menus**. The Menus pane opens.
3. From the tree structure, select the tool or submenu that you want to edit, and then click **Edit Item** in the toolbar. The Menu Item Details window opens.
4. Edit the menu item as follows:

Menu Item Type

If you are adding a menu item, select the type of menu item that you want to add to the selected menu. If you are editing a tool or submenu, this field is read-only.

Note: If you are adding a separator, this is the only entry required within this window.

Tool Select the tool that you want to add. If you are editing a tool, you can choose to select another tool from this drop-down list.

Note: This field is available for tools only.

If you want to create a new tool that can then be added as a menu item within the selected menu, click **Add Tool**. The Tool Details window opens. Complete this window and save your changes. When you return to the Menu Item Details window, the tool will be available for selection within the **Tool** drop-down list.

If you want to edit the settings of the selected tool, click **Edit Tool**. The Tool Details window opens. Complete this window and save your changes. You return to the Menu Item Details window.

Title If you are adding or editing a tool or a submenu, either accept the default text (if given) in this field, or type another title. Include an ampersand (&) in the title if you want to create a mnemonic. This text will be shown as the menu item name within the menu.

Note: If you are adding a tool, the **Title** field defaults to the name of the tool selected. However, if you subsequently select a different tool, the initial **Title** field entry will remain unchanged unless you manually update it.

5. Save or cancel your changes as follows:

OK Click this button to save the menu item details and close the window. The Menus pane is updated to reflect the changes made. Your changes are also reflected in the relevant event list or Conductor menu the next time the desktop is started or the event list is resynchronized with the ObjectServer.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Creating and editing tools” on page 91

Changing the order of menu items

You can reorder the items in a menu to enhance user workflows.

To change the order of desktop menu items:

1. From the Netcool/OMNIBus Administrator window, select the **Menu** menu button.
2. Click **Menus**. The Menus pane opens.
3. For each menu item that you want to reposition, select the menu item and move it to its required position using any of the following methods:
 - To move an item to the top of the menu, select **Move To Top** from the **Item** menu, toolbar, or from the pop-up menu obtained when you right-click over the item.
 - To move an item up one position, select **Move Up** from the **Item** menu, toolbar, or from the pop-up menu obtained when you right-click over the item.
 - To move an item down one position, select **Move Down** from the **Item** menu, toolbar, or from the pop-up menu obtained when you right-click over the item.
 - To move an item to the bottom of the menu, select **Move To Bottom** from the **Item** menu, toolbar, or from the pop-up menu obtained when you right-click over the item.

Repositioned menu items are displayed in the new order the next time the desktop is started or the event list is resynchronized with the ObjectServer.

Removing tools, submenus, or separators from a menu

To remove a menu item from the desktop:

1. From the Netcool/OMNIbus Administrator window, select the **Menu** menu button.
2. Click **Menus**. The Menus pane opens.
3. Select the menu item that you want to remove. For submenus, you can select a single item for removal, or you can select the submenu name to delete the submenu and its contents.
4. From the toolbar, click **Delete**.

Results

The menu item is removed from the desktop the next time the desktop is started or the event list is resynchronized with the ObjectServer.

Previewing the structure of customized menus

You can preview customized menus to see how they appear in the desktop.

To preview menus:

1. From the Netcool/OMNIbus Administrator window, select the **Menu** menu button.
2. Click **Menus**. The Menus pane opens.
3. From the toolbar, click **Show Menu Structure**. The preview window opens with the menus listed horizontally. You can click each menu name to expand its contents.

Results

Note: The menu names as they appear in Netcool/OMNIbus Administrator differ from their names on the desktop.

Configuring tools

Tools help operators to manage alerts in the event list. You can create tools that operators can use to run SQL commands on the ObjectServer, or run external commands that start a local application, batch file, or script.

A tool can have an associated SQL statement, executable commands, or both. A tool can also include a prompt window or a pop-up menu for operators to enter or select information.

You can add tools to event list menus and associate tools with alert classes. Such tools are available from the menus only when alerts of the associated classes are selected in the event list.

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 the *IBM Tivoli Netcool/OMNIbus User's Guide*.

Related tasks

"Adding tools, submenus, and separators to a menu" on page 87

Creating and editing tools

When you create a tool, it is added to the tools database. The tools that appear in the Tools pane are links to entries in this database.

To create or edit a tool:

1. From the Netcool/OMNIBus Administrator window, select the **Menu** menu button.
2. Click **Tools**. The Tools pane opens.
3. To add a tool, click **Add Tool** in the toolbar. The Tool Details window opens.
4. To edit a tool, select the tool to edit and then click **Edit Tool** in the toolbar. The Tool Details window opens.
5. Define the tool as follows:

Name Type a unique name for the tool. If you are editing a tool, you cannot change the name.

Enabled

Select this check box to activate the tool on the system and make it available for operators to use. Clear this check box to create the tool without activating it at the present time, or to make the tool unavailable.

6. From the **SQL** tab, enter any SQL commands that should run on the ObjectServer when the tool is selected for use. Complete the tab as follows:

Enabled

Select this check box to specify that the SQL commands entered must run when the tool is used.

Execute for each selected row

Select this check box to specify that the SQL commands must run once for each row in an event list row selection.

SQL Commands

Type the SQL commands that must run when the tool is used.

You can use the SQL helper buttons shown to the right of this field to construct the SQL commands.

Tip: You can include a prompt in an SQL statement so that when the tool runs, it calls up a prompt window or a pop-up menu for users to enter or select information.

7. From the **Executable** tab, enter any external commands that should run when the tool is selected for use. Complete the tab as follows:

Enabled

Select this check box to specify that the executable commands entered must run when the tool is used.

Execute for each selected row

Select this check box to specify that the executable commands must run once for each row in an event list row selection.

Redirect output

Select this check box to specify that output must be echoed through a read-only window when the commands are run. Clear this check box to discard the output.

Redirect errors

Select this check box to specify that any error messages must be

echoed through a read-only window when the commands are run. Clear this check box to discard the error messages.

Executable Commands

Type the commands that must run when this tool is used.

You can use the SQL helper buttons shown to the right of this field to construct the executable commands.

Tip: You can include a prompt in an external command so that when the tool runs, it calls up a prompt window or a pop-up menu for users to enter or select information.

8. From the **Journal** tab, specify journal settings that should apply when the tool is selected for use. Complete the tab as follows:

Force Journal Entry

Select this check box to force users to enter journal text before running 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

Type the text for the journal entry.

You can use the helper buttons shown to the right of this field to construct the journal text.

Tip: You can include a prompt in a journal entry so that when the tool runs, it calls up a prompt window or a pop-up menu for users to enter or select information.

9. From the **Access** tab, specify alert classes to which the tool applies, and the groups of users permitted to use the tool. Complete the tab as follows:

Class Access

Select the alert classes for which the tool can be used. To select all alert classes, click **Tick All** to the immediate right of this list. To clear all your selections, click **Tick None** to the immediate right of this list. You can also individually select each check box required.

Group Access

Select the user groups that can use this tool. To select all groups, click **Tick All** to the immediate right of this list. To clear all your selections, click **Tick None** to the immediate right of this list. You can also individually select each check box required.

10. From the **Platform** tab, specify the operating system platforms on which the tool will be available. Complete the tab as follows:

Available Platforms

Select the check boxes for the operating systems on which the tool will be available. To select all operating systems, click **Tick All** to the right of this list. To clear all your selections, click **Tick None** to the right of this list.

Important: You *must* always specify the operating systems on which the tool can be used. The tool is only available on platforms that are selected.

11. From the **Description** tab, enter an optional text description for this tool. This can be useful to anyone who is trying to understand how the tool works.
12. Save or cancel your changes as follows:
 - OK** Click this button to save the tool details and close the window. New tools are added to the Tools pane.
 - Cancel** Click this button to close the window without saving your changes.

What to do next

The tools shown within the Tools pane can be added to the event list and Conductor desktop menus to help with alert management.

Related tasks

“Adding tools, submenus, and separators to a menu” on page 87

“Creating and editing prompts” on page 94

Related reference

Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361

Deleting tools

To delete a tool:

1. From the Netcool/OMNIBus Administrator window, select the **Menu** menu button.
2. Click **Tools**. The Tools pane opens.
3. Select the tool that you want to delete and click **Delete** in the toolbar. The tool is deleted.

Configuring prompts

An event list tool can include a prompt window or a pop-up menu for users to enter or select information.

When you create or edit a tool, you can include a prompt in an SQL statement, an external command, or a journal entry.

You can create the following types of prompts:

- **String**: This creates a prompt window that accepts one or more characters.
- **Integer**: This creates a prompt window that accepts an integer value.
- **Float**: This creates a prompt window that accepts a floating point number, which can contain a decimal point.
- **Time**: This creates a prompt window that accepts a time.
- **Fixed choice**: This creates a pop-up menu that is populated with options that you specify.
- **Lookup**: This creates a pop-up menu or drop-down list that is populated by the values in a specified file.
- **Password**: This creates a prompt window that accepts one or more characters as a password.
- **Dynamic choice**: This creates a pop-up menu or drop-down list that is populated by the results of a database query.

Creating and editing prompts

You can create prompts to use with tools.

To create or edit a prompt:

1. From the Netcool/OMNIBus Administrator window, select the **Menu** menu button.
2. Click **Prompts**. The Prompts pane opens.
3. To add a prompt, click **Add Prompt** in the toolbar. The Prompt Details window opens. The fields within this window vary based on the type of prompt being created or edited.
4. To edit a prompt, select the tool to edit and then click **Edit Prompt** in the toolbar. The Prompt Details window opens. The fields within this window vary based on the type of prompt being created or edited.
5. To create or edit a string prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select String to create a prompt window that accepts one or more characters.

Default

Type a default value for the prompt to display.

6. To create or edit an integer prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select Integer to create a prompt window that accepts an integer value.

Default

Type a default value for the prompt to display.

7. To create or edit a float prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select Float to create a prompt window that accepts a floating point number, which can contain a decimal point.

Default

Type a default value for the prompt to display.

8. To create or edit a time prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select Time to create a prompt window that accepts a time. For a time prompt, the default is to display the current time.

9. To create or edit a fixed choice prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select Fixed Choice to create a pop-up menu that is populated with options that you specify.

Options

Use this field to specify menu options to include in the pop-up menu. To create a new option, click **Add option** and type a name for the option in the box that is shown within the **Options** list. Click outside the box to complete the entry. You can also edit an existing option by clicking the entry in the **Options** list and then clicking **Edit option**. To delete an option, click the entry and then click **Delete option**.

10. To create or edit a lookup prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select Lookup to create a pop-up menu or drop-down list that is populated by the values in a specified file.

File Type the path and name of the file whose contents must be used to populate the pop-up menu or drop-down list associated with the prompt. Alternatively, click **Browse** to open a file selection window, navigate to the appropriate location, and select the file.

11. To create or edit a password prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select Password to create a prompt window that accepts one or more characters as a password. For a password prompt, the password characters appear as asterisks when the user completes the prompt.

12. To create or edit a dynamic choice prompt, complete the window as follows:

Name Type a unique name for the prompt. This name must comply with the ObjectServer naming conventions. If you are editing a prompt, you cannot change the name.

Prompt

Type the prompt text that should appear when the tool requests information from the user.

Type Select **Dynamic Choice** to create a pop-up menu or drop-down list that is populated by the results of a database query.

Database

Select a database from this drop-down list.

Table Select a table in the selected database, from the drop-down list.

Show Select a column name from the drop-down list. This defines the column that is used to populate the prompt menu.

Assign

Select a column name from the drop-down list. This defines the column that is used to return a value to the SQL command, external command, or journal entry that contains the prompt.

Where Type an SQL search condition. For example: Colname='Severity'.

Order By

Select a column name from the drop-down list. This defines the column by which items are sorted in the prompt menu. Use the **Ascending** and **Descending** options to specify the sort direction.

13. Save or cancel your changes as follows:

OK Click this button to save the prompt details and close the window. New prompts are added to the Prompts pane.

Cancel

Click this button to close the window without saving your changes.

What to do next

The prompts shown within the Prompts pane can be added to tools that are set up to help operators manage alerts in the event list. When such a tool is run, a prompt window or a pop-up menu is displayed for users to enter or select information.

Related concepts

"Naming conventions for ObjectServer objects" on page 139

Related tasks

"Creating and editing tools" on page 91

Deleting prompts

To delete a prompt:

1. From the Netcool/OMNIbus Administrator window, select the **Menu** menu button.
2. Click **Prompts**. The Prompts pane opens.
3. Select the prompt that you want to delete and click **Delete** in the toolbar. The prompt is deleted.

Configuring automations

You can use automation to detect changes in the ObjectServer and to run 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 resolved, 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 resolved, 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 events. Netcool/OMNIbus includes a number of standard automations.

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

Signals and *procedures* are also part of the automation subsystem. Signals can have triggers attached to them, so that the ObjectServer can automatically respond when a signal is raised. Procedures are executable programs that are created to perform common operations, and you can run them in a trigger, or from the SQL interactive interface.

Related concepts

“SQL interactive interface” on page 135

Related reference

“Running procedures” on page 205

“Standard Tivoli Netcool/OMNIbus automations” on page 230

Configuring triggers

There are three types of triggers: database triggers, signal triggers, and temporal triggers. Database triggers fire when a triggering database incident occurs. Signal triggers fire when a system or user-defined signal is raised. Temporal triggers fire repeatedly based on a specified frequency.

Every trigger must belong to a *trigger group*, which is a collection of related triggers. You can enable or disable all of the triggers in a given trigger group, in a single action.

Tip: You can create and edit triggers from the default Netcool/OMNIbus Administrator windows, or from an external text editor of your choice.

Note: The ObjectServer is configured to write output from a trigger to a log file. The file has a defined maximum size, when this size is reached the ObjectServer starts writing to a new log file. There is a maximum number of files. When the maximum size and maximum number of files are exceeded, the files are rotated. During file rotation, do not open the log files with exclusive write access (for example, by using Microsoft Excel), because the ObjectServer is unable to rotate the files.

Creating and editing trigger groups

When creating a trigger, you are required to assign it to a trigger group. You can create trigger groups *before* you create your triggers (as a separate action), or *while* you are creating your triggers. This task describes how to create a trigger group in a separate action.

To create or edit a trigger group:

1. From the Netcool/OMNIbus Administrator window, select the **Automation** menu button.
2. Click **Trigger Groups**. The Trigger Groups pane opens.
3. To add a trigger group, click **Add Trigger Group** in the toolbar. The Trigger Group Details window opens.
4. To edit a trigger group, select the trigger group to edit and then click **Edit Trigger Group** in the toolbar. The Trigger Group Details window opens.
5. Complete the fields as follows:

Group Name

Type a unique name for the trigger group. If you are editing a trigger group, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Enabled

Select this check box to activate the trigger group.

Clear this check box to create the trigger group without activating it at the present time, or to make the trigger group unavailable. When a trigger group is unavailable, all the triggers in the group are also unavailable for use. You can also make individual triggers available or unavailable while creating or editing them.

6. Save or cancel your changes as follows:

OK Click this button to save the trigger group details and close the window. New trigger groups are added to the Trigger Groups pane.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Creating and editing database triggers”

“Creating and editing signal triggers” on page 101

“Creating and editing temporal triggers” on page 103

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.

To create or edit a database trigger:

1. From the Netcool/OMNIBus Administrator window, select the **Automation** menu button.
2. Click **Triggers**. The Triggers pane opens.
This pane lists all database, signal, and temporal triggers that are set up.

Tip: To view only one type of trigger, click **Show Database Triggers Only**, **Show Temporal Triggers Only**, or **Show Signal Triggers Only** in the toolbar.

3. To add a database trigger, click **Add Database Trigger** in the toolbar. The Database Trigger Details window opens.
4. To edit a database trigger, select the database trigger to edit and then click **Edit Trigger** in the toolbar. The Database Trigger Details window opens.
5. Define or edit the trigger setup details as follows:

Name Type a unique trigger name. If you are editing a trigger, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Group Select the trigger group to which you want to assign the trigger.

Add new trigger group

Click this button if you want to create a new trigger group to which the trigger can then be assigned. The Trigger Group Details window opens. Complete this window and save your changes.

When you return to the Database Trigger Details window, the new trigger group is shown as the currently-selected trigger group.

6. Complete the **Settings** tab as follows:

On Select the ObjectServer database and the database table that cause the trigger to fire.

Priority

Select a priority that determines the order in which the ObjectServer fires triggers when this database modification causes more than one trigger to fire. You can select numbers from 1 to 20, with 1 being the highest priority.

Pre database action/Post database action

Click **Pre database action** to indicate that the trigger action should run before the database modification that caused the trigger to fire occurs.

Click **Post database action** to indicate that the trigger action should run after the database modification that caused the trigger to fire occurs.

For example, you can click **Pre database action** to evaluate a user name before a row in the alerts.status table 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. If you click **Post database action**, the database modification always takes place.

Delete/Insert/Reinsert/Update

Use these options to specify the type of database modification that should occur.

Row/Statement

Click **Row** to set the trigger to fire once for each row that matches the trigger condition. Click **Statement** to set the trigger to fire once regardless of the number of matched rows in the table.

Debug

Select this check box to send debugging information to the ObjectServer message log each time the trigger fires.

Enabled

Select this check box to activate the trigger and make it available for use. Clear this check box to create the trigger without activating it at the present time, or to make the trigger unavailable. A disabled trigger does not fire when the associated database modification occurs.

7. From the **When** tab, specify an optional WHEN clause that allows you to test for a particular condition before the action is performed. If the condition is not met, the action is not performed. You can use the helper buttons shown to the right of the field to construct the WHEN clause.

8. From the **Action** tab, enter SQL commands for the 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 (;).

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 (;).

The trigger body has the following syntax:

```
[ DECLARE variable_declaration;...[;] ]  
BEGIN  
    trigger_statement_list  
END;
```

You can use the SQL helper buttons shown to the right of the SQL editor panel to construct the SQL commands.

9. From the **Comment** tab, enter an optional text comment for the trigger. This may be useful to anyone who is trying to understand how the trigger works.
10. Save or cancel your changes as follows:

OK Click this button to save the trigger details and close the window. New triggers are added to the Triggers pane.

Cancel

Click this button to close the window without saving your changes.

Related concepts

“Conditions” on page 172

Related tasks

“Creating and editing trigger groups” on page 98

Related reference

“Creating database triggers (CREATE TRIGGER command)” on page 208

“Best practices for creating triggers” on page 317

Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361

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. *User-defined* signals are explicitly created, raised, and dropped.

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.

To create or edit a signal trigger:

1. From the Netcool/OMNIBus Administrator window, select the **Automation** menu button.
2. Click **Triggers**. The Triggers pane opens.
This window lists all database, signal, and temporal triggers that are set up.

Tip: To view only one type of trigger, click **Show Database Triggers Only**, **Show Temporal Triggers Only**, or **Show Signal Triggers Only** in the toolbar.

3. To add a signal trigger, click **Add Signal Trigger** in the toolbar. The Signal Trigger Details window opens.
4. To edit a signal trigger, select the signal trigger to edit and then click **Edit Trigger** in the toolbar. The Signal Trigger Details window opens.
5. Define or edit the trigger setup details as follows:

Name Type a unique trigger name. If you are editing a trigger, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Group Select the trigger group to which you want to assign the trigger.

Add New Trigger Group

Click this button if you want to create a new trigger group to which the trigger can then be assigned. The Trigger Group Details window opens. Complete this window and save your changes.

When you return to the Signal Trigger Details window, the new trigger group is shown as the currently-selected trigger group.

6. Complete the **Settings** tab as follows:

Signal Select the signal that must cause the trigger to fire.

Priority

Select a priority that determines the order in which the ObjectServer fires triggers when this signal causes more than one trigger to fire. You can select numbers from 1 to 20, with 1 being the highest priority.

Debug

Select this check box to send debugging information to the ObjectServer message log each time the trigger fires.

Enabled

Select this check box to activate the trigger and make it available for use. Clear this check box to create the trigger without activating it at the present time, or to make the trigger unavailable. A disabled trigger does not fire when the associated signal is raised.

7. From the **When** tab, specify an optional WHEN clause that allows you to test for a particular condition before the action is performed. If the condition is not met, the action is not performed. You can use the helper buttons shown to the right of the field to construct the WHEN clause.
8. From the **Evaluate** tab, optionally build a temporary result set from a single SELECT statement to be processed during the trigger action that is defined on the **Action** tab. Complete the tab as follows:

Bind As

Type the name of the temporary table in which to store the result set.

SQL editor panel

Type the statement using the format:

```
EVALUATE SELECT_cmd
```

You can use the SQL helper buttons shown to the right of the field to construct the statement.

9. From the **Action** tab, enter SQL commands for the 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 (;).

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 (;).

The trigger body has the following syntax:

```
[ DECLARE variable_declaration;...[;] ]
BEGIN
  trigger_statement_list
END;
```

You can use the SQL helper buttons shown to the right of the SQL editor panel to construct the SQL commands.

10. From the **Comment** tab, enter an optional text comment for the trigger. This can be useful to anyone who is trying to understand how the trigger works.
11. Save or cancel your changes as follows:

OK Click this button to save the trigger details and close the window. New triggers are added to the Triggers pane.

Cancel

Click this button to close the window without saving your changes.

Related concepts

“Configuring signals” on page 115

“Conditions” on page 172

Related tasks

“Creating and editing trigger groups” on page 98

Related reference

“Creating signal triggers (CREATE TRIGGER command)” on page 212

“System signals and their attributes” on page 217

“Best practices for creating triggers” on page 317

Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361

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.

To create or edit a temporal trigger:

1. From the Netcool/OMNIBus Administrator window, select the **Automation** menu button.
2. Click **Triggers**. The Triggers pane opens.
This pane lists all database, signal, and temporal triggers that are set up.

Tip: To view only one type of trigger, click **Show Database Triggers Only**, **Show Temporal Triggers Only**, or **Show Signal Triggers Only** in the toolbar.

3. To add a temporal trigger, click **Add Temporal Trigger** in the toolbar. The Temporal Trigger Details window opens.
4. To edit a temporal trigger, select the temporal trigger to edit and then click **Edit Trigger** in the toolbar. The Temporal Trigger Details window opens.
5. Define or edit the trigger setup details as follows:

Name Type a unique trigger name. If you are editing a trigger, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (`_`) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Group Select the trigger group to which you want to assign the trigger.

Add New Trigger Group

Click this button if you want to create a new trigger group to which the trigger can then be assigned. The Trigger Group Details window opens. Complete this window and save your changes.

When you return to the Temporal Trigger Details window, the new trigger group will be added to the **Group** list. You can then assign the trigger to this trigger group.

6. Complete the **Settings** tab as follows:

Every Define when the trigger must fire. Specify a numeric value and select **hours**, **minutes**, or **seconds** from the drop-down list.

Priority

Select a priority that determines the order in which the ObjectServer fires triggers when more than one trigger occurs at this frequency. You can select numbers from 1 to 20, with 1 being the highest priority.

Debug

Select this check box to send debugging information to the ObjectServer message log each time the trigger fires.

Enabled

Select this check box to activate the trigger and make it available for use. Clear this check box to create the trigger without activating it at the present time, or to make the trigger unavailable. A disabled trigger does not fire.

7. From the **When** tab, specify an optional WHEN clause that allows you to test for a particular condition before the action is performed. If the condition is not met, the action is not performed. You can use the helper buttons shown to the right of the field to construct the WHEN clause.
8. From the **Evaluate** tab, optionally build a temporary result set from a single SELECT statement to be processed during the trigger action that is defined on the **Action** tab. Complete the tab as follows:

Bind As

Type the name of the temporary table in which to store the result set.

SQL editor panel

Type the statement using the format:

```
EVALUATE SELECT_cmd
```

You can use the SQL helper buttons shown to the right of the SQL editor panel to construct the statement.

9. From the **Action** tab, enter SQL commands for the 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 (;).

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 (;).

The trigger body has the following syntax:

```
[ DECLARE variable_declaration;...[;] ]  
BEGIN  
  trigger_statement_list  
END;
```

You can use the SQL helper buttons shown to the right of the SQL editor panel to construct the SQL commands.

10. From the **Comment** tab, enter an optional text comment for the trigger. This can be useful to anyone who is trying to understand how the trigger works.
11. Save or cancel your changes as follows:

OK Click this button to save the trigger details and close the window. New triggers are added to the Triggers pane.

Cancel

Click this button to close the window without saving your changes.

Related concepts

"Conditions" on page 172

Related tasks

"Creating and editing trigger groups" on page 98

Related reference

"Creating temporal triggers (CREATE TRIGGER command)" on page 210

"Best practices for creating triggers" on page 317

Appendix B, "SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists," on page 361

Using an external editor to create and edit triggers

From Netcool/OMNIbus Administrator, you can configure an external text editor and then use it to create or edit triggers.

Configuring an external editor for triggers:

If required, you can create and edit triggers from an external text editor of your choice. You must first configure which editor you want to use.

To configure an external editor:

1. From Netcool/OMNIbus Administrator, click **Tools > Configure Tools**. The Choose Tool window opens.
2. Select **Text Editor**. The External Program window opens.
3. Complete this windows as follows:

Tool name

This field displays the type of tool that you are configuring.

Executable

Type the full path and name of the executable program for the tool type. Alternatively, click the button to the right of the field to search for and select the executable program.

Arguments

Type any command-line arguments to run with this executable program.

Run time environment

Select the runtime environment.

4. Save or cancel your changes as follows:

OK Click this button to save the external program details and close the window.

Cancel

Click this button to close the window without saving your changes.

Related tasks

"Creating and editing triggers in an external editor" on page 106

Creating and editing triggers in an external editor:

You can use an external editor to create or edit a database, signal, or temporal trigger.

Before you begin

You must have configured an external editor to use for creating or editing triggers.

To create or edit a trigger:

1. From the Netcool/OMNIbus Administrator window, select the **Automation** menu button.
2. Click **Triggers**. The Triggers pane opens. This window lists all database, signal, and temporal triggers that are set up.

Tip: To view only one type of trigger, click **Show Database Triggers Only**, **Show Temporal Triggers Only**, or **Show Signal Triggers Only** in the toolbar.

3. To create a trigger, make sure that no row is selected in the Triggers pane, and then click **Edit in External Editor** in the toolbar. (You can deselect a row by pressing Ctrl and then clicking the row.) The Select Trigger Type dialog box opens. Proceed as follows:
 - a. From the **Trigger Template** list, select the template for the type of trigger that you want to create. If you select **Database**, **Signal**, or **Temporal**, the external editor opens with standard syntax for that trigger type. If you select **<blank>**, the external editor opens with no text shown.
 - b. Complete the syntax for the trigger. If you are using a template, replace the *trigger_name* and *group_name* variables with real values. Additionally, for a signal trigger, replace the *signalName* variable with a real value, and for a database trigger, replace the *database_name.table_name* variable with a real value. Add trigger-specific statements, optional clauses, and variable declarations, as required. Then add the body of the trigger between the Begin and End keywords. The template includes comments (preceded by --) for the placement of text.
 - c. Save your entries. When you save, the trigger is saved to the ObjectServer. If there are any syntax errors, you are prompted to reload the contents of the external editor.
 - d. Close the external editor.
4. To edit a trigger, select the trigger to edit and then click **Edit in External Editor** in the toolbar, or right-click and select **Edit in External Editor** from the pop-up menu. The external editor opens, with the trigger syntax displayed. Proceed as follows:
 - a. Edit the trigger syntax and save your changes. When you save, the trigger is saved to the ObjectServer. If there are any syntax errors, you are prompted to reload the contents of the external editor.
 - b. Close the external editor.

Results

Tip: The SQL that you enter in an external editor is saved to the ObjectServer as a .ed file. You can check the validity of the syntax in .ed and other .sql files from the SQL interactive interface (running in GUI mode).

Related tasks

“Configuring an external editor for triggers” on page 105

“Using the SQL interactive interface in GUI mode” on page 131

Related reference

“Creating database triggers (CREATE TRIGGER command)” on page 208

“Creating signal triggers (CREATE TRIGGER command)” on page 212

“Creating temporal triggers (CREATE TRIGGER command)” on page 210

Deleting trigger groups

You cannot delete a trigger group that contains triggers.

To delete a trigger group:

1. From the Netcool/OMNIbus Administrator window, select the **Automation** menu button.
2. Click **Trigger Groups**. The Trigger Groups pane opens.
3. Select the trigger group that you want to delete and click **Delete** in the toolbar. The trigger group is deleted.

Deleting triggers

To delete a trigger:

1. From the Netcool/OMNIbus Administrator window, select the **Automation** menu button.
2. Click **Triggers**. The Triggers pane opens.
3. Select the trigger that you want to delete and click **Delete** in the toolbar. 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 as follows:

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

Tip: You can create and edit procedures from the default Netcool/OMNIbus Administrator windows, or from an external text editor of your choice.

Creating and editing SQL procedures

SQL procedures have the following major components: parameters, local variable declarations, and the procedure body.

Parameters are values that are passed into or out of a procedure. You declare parameters when you create a procedure, and you specify what values are passed as parameters when you run the procedure.

Local variable declarations are declared before the procedure body, and can be used to define values that are used when the procedure runs.

To create or edit an SQL procedure:

1. From the Netcool/OMNIbus Administrator window, select the **Automation** menu button.

2. Click **Procedures**. The Procedures pane opens.
3. To add an SQL procedure, click **Add SQL Procedure** in the toolbar. The SQL Procedure Details window opens.
4. To edit an SQL procedure, select the SQL procedure to edit and then click **Edit Procedure** in the toolbar. The SQL Procedure Details window opens.
5. Complete this window as follows:

Name Type a unique name for the procedure. If you are editing a procedure, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Parameters

This area displays the parameters that have been created for passing into and out of the procedure.

You can use the up and down arrows to the right of the list box to change the order of any selected parameter. You can also click **Remove parameter** to the right of the list box to remove any selected parameter from the list.

To create a parameter, use the **In/Out**, **Name**, and **Data Type** fields, the **Array** check box (if necessary), and the **Add parameter to the list** button.

In/Out

Select a mode for the parameter being created. 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.

Name Type a name for the parameter being created. Parameter names must be unique within the procedure.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Data Type

Select the type of data that the parameter can pass into or out of the procedure. The data type can be any valid ObjectServer data type except VARCHAR or INCR.

Array If you selected the **in** mode from the **In/Out** drop-down list, you can select this check box to pass an array of the selected data type into the procedure.

Add parameter to the list

After completing the **In/Out**, **Name**, and **Data Type** fields, and the optional **Array** check box, click this button to add the parameter to the parameter list.

Actions

Type 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 (;).

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 (;).

You can use the SQL helper buttons shown to the right of the SQL editor panel to construct the SQL commands.

6. Save or cancel your changes as follows:

OK Click this button to save the SQL procedure and close the window. New SQL procedures are added to the Procedures pane.

Cancel Click this button to close the window without saving your changes.

What to do next

After you create a procedure in the ObjectServer, you can run it from the SQL interactive interface (**iSQL**), or run it in a trigger using the EXECUTE PROCEDURE command.

Related reference

“Creating SQL procedures (CREATE PROCEDURE command)” on page 196

“Running procedures” on page 205

“Specifying data types for columns” on page 145

Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361

Example: SQL procedure

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

Within the SQL Procedure Details window, the SQL procedure named node_report is created with the following entries:

Table 20. Entries for the node_report SQL procedure in the SQL Procedure Details window

Field	Entry
Name	node_report
Parameters	<p>in node_name Char(255)</p> <p>This read-only entry in the Parameters list is constructed from entries made in the In/Out, Name, Data Type, and Length fields in the Parameters area. For example:</p> <ul style="list-style-type: none">• In/Out: in• Name: node_name• Data Type: Char• Length: 255

Table 20. Entries for the node_report SQL procedure in the SQL Procedure Details window (continued)

Field	Entry
Actions	<pre> 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 </pre>

The SQL command to create the node_report_file ObjectServer file and the full SQL text of the same node_report procedure is as follows:

```

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 program on a local or remote system.

To create or edit an external procedure:

1. From the Netcool/OMNIbus Administrator window, select the **Automation** menu button.
2. Click **Procedures**. The Procedures pane opens.
3. To add an external procedure, click **Add External Procedure** in the toolbar. The External Procedure Details window opens.
4. To edit an external procedure, select the external procedure to edit and then click **Edit Procedure** in the toolbar. The External Procedure Details window opens.
5. Complete this window as follows:

Name Type a unique name for the procedure. If you are editing a procedure, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Parameters

This area displays the parameters that have been created for the

procedure. These parameters can be base type variables, arrays of base type values, or rows of named tables.

You can use the up and down arrows to the right of the list box to change the order of any selected parameter. You can also click **Remove parameter** to the right of the list box to remove any selected parameter from the list.

To create a parameter, use the **In/Out**, **Name**, and **Data Type** fields, the **Array** check box, and the **Add parameter to the list** button.

In/Out

The **in** mode is selected by default. You cannot change this value because 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 Type a name for the parameter being created. Parameter names must be unique within the procedure.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Data Type

Select the type of data that the parameter can pass into the procedure. The data type can be any valid ObjectServer data type except VARCHAR or INCR.

Array Select this check box to pass an array of the selected data type into the procedure.

Add parameter to the list

After completing the **In/Out**, **Name**, and **Data Type** fields, and the **Array** check box, click this button to add the parameter to the parameter list.

Executable

Type the full path for the command to run. You can click **Browse** to help locate and select the file.

Arguments

Type any command-line arguments for the command to run.

Host Type the name of the host computer on which to run the procedure. The logged-in computer name is shown by default.

User ID

Specify the (UNIX) user ID under which to run the executable program.

Group ID

Specify the (UNIX) group ID under which to run the executable program.

6. Save or cancel your changes as follows:

OK Click this button to save the external procedure and close the window. New external procedures are added to the Procedures pane.

Cancel

Click this button to close the window without saving your changes.

What to do next

After you create a procedure in the ObjectServer, you can run it from the SQL interactive interface (**iSQL**), or run it in a trigger using the EXECUTE PROCEDURE command.

Note: To run an external procedure, you must have a process control agent daemon (**nco_pad**) running on the host where the executable file is stored.

Related concepts

Chapter 7, “Using process control to manage processes and external procedures,” on page 249

Related reference

“Specifying data types for columns” on page 145

“Running procedures” on page 205

Example: External procedure

This example external procedure calls a program called **nco_mail**, which sends e-mail about unacknowledged critical alerts.

Within the External Procedure Details window, the external procedure named **send_email** is created with the following entries:

Table 21. Entries for the send_email external procedure in the External Procedure Details window

Field	Entry
Name	send_email
Parameters	<div>in node Char(255) in severity Integer in subject Char(255) in email Char(255) in summary Char(255) in hostname Char(255)</div> <div>These read-only entries in the Parameters list are constructed from entries made in the In/Out, Name, Data Type, and Length fields in the Parameters area. For example, for in node Char(255), the entries are:</div> <ul style="list-style-type: none">• In/Out: in• Name: node• Data Type: Char• Length: 255
Executable	\$NCHOME/omnibus/utils/nco_mail
Arguments	'\'+node+'\'' , severity, '\'+subject+'\'' , '\'+email+'\'' , '\'+summary+'\''
Host	localhost

The full SQL text of the same **send_email** procedure is as follows:

```
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 '$NCHOME/omnibus/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 quotation marks, and the quotation marks must be escaped with backslashes. All quotation marks in this example are single quotation marks.

Using an external editor to create and edit procedures

From Netcool/OMNIBus Administrator, you can configure an external text editor and then use it to create or edit procedures.

Configuring an external editor for procedures:

If required, you can create and edit procedures from an external text editor of your choice. You must first configure which editor you want to use.

To configure an external editor:

1. From Netcool/OMNIBus Administrator, select **Tools > Configure Tools**. The Choose Tool window opens.
2. Select **Text Editor**. The External Program window opens.
3. Complete this windows as follows:

Tool name

This field displays the type of tool that you are configuring.

Executable

Type the full path and name of the executable program for the tool type. Alternatively, click the button to the right of the field to search for and select the executable program.

Arguments

Type any command-line arguments to run with this executable program.

Run time environment

Select the runtime environment.

4. Save or cancel your changes as follows:

OK Click this button to save the external program details and close the window.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Creating and editing procedures in an external editor” on page 114

Creating and editing procedures in an external editor:

You can use an external editor to create or edit an SQL or external procedure.

Before you begin

You must have configured an external editor to use for creating or editing procedures.

To create or edit a procedure:

1. From the Netcool/OMNIbus Administrator window, select the **Automation** menu button.
2. Click **Procedures**. The Procedures pane opens.
3. To create a procedure, make sure that no row is selected in the Procedures pane, and then click **Edit in External Editor** in the toolbar. (You can deselect a row by pressing Ctrl and then clicking the row.) The Select Procedure Type dialog box opens. Proceed as follows:
 - a. From the **Procedure Template** list, select the template for the type of procedure that you want to create. If you select **SQL** or **External**, the external editor opens with standard syntax for that procedure type. If you select **<blank>**, the external editor opens with no text shown.
 - b. Complete the syntax for the procedure. If you are using a template, replace the *procedure_name* variable with a real value. Additionally, for an external procedure, replace the *executableName*, *hostName*, *userID*, and *groupID* variables with real values. Add procedure-specific statements and other declarations, as required. The template might include comments (preceded by --) for the placement of text.
 - c. Save your entries. When you save, the procedure is saved to the ObjectServer. If there are any syntax errors, you are prompted to reload the contents of the external editor.
 - d. Close the external editor.
4. To edit a procedure, select the procedure to edit and then click **Edit in External Editor** in the toolbar, or right-click and select **Edit Procedure in External Editor** from the pop-up menu. The external editor opens, with the procedure syntax displayed. Proceed as follows:
 - a. Edit the procedure syntax and save your changes. When you save, the procedure is saved to the ObjectServer. If there are any syntax errors, you are prompted to reload the contents of the external editor.
 - b. Close the external editor.

Results

Tip: The SQL that you enter in an external editor is saved to the ObjectServer as a .ed file. You can check the validity of the syntax in .ed and other .sql files from the SQL interactive interface (running in GUI mode).

Related tasks

“Configuring an external editor for procedures” on page 113

“Using the SQL interactive interface in GUI mode” on page 131

Related reference

“Creating external procedures (CREATE PROCEDURE command)” on page 203

“Creating SQL procedures (CREATE PROCEDURE command)” on page 196

Deleting procedures

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

To delete a procedure:

1. From the Netcool/OMNIBus Administrator, select the **Automation** menu button.
2. Click **Procedures**. The Procedures pane opens.
3. Select the procedure that you want to delete and click **Delete** in the toolbar. The procedure is deleted.

Configuring signals

Signals are occurrences within the ObjectServer that can be detected and acted upon. Signals are part of the automation subsystem.

The types of signals are as follows:

- System signals
- User-defined signals

An ObjectServer automatically raises system signals when certain changes in the system occur; for example, during system startup or a connection failure. You cannot create or modify these signals. You can attach triggers to system signals to create automatic responses to incidents in the ObjectServer.

User-defined signals are defined by you. You can use Netcool/OMNIBus Administrator to create your own user-defined signals.

Related tasks

“Creating and editing signal triggers” on page 101

Creating and editing user-defined signals

Unlike system signals, which are predefined and cannot be configured, user-defined signals must be explicitly created or deleted.

To create or edit a user-defined signal:

1. From the Netcool/OMNIBus Administrator window, select the **Automation** menu button.
2. Click **User Defined Signals**. The User Defined Signals pane opens.
3. To add a user-defined signal, click **Add User Defined Signal** in the toolbar. The User Defined Signal Details window opens.
4. To edit a user-defined signal, select the user-defined signal to edit and then click **Edit User Defined Signal** in the toolbar. The User Defined Signal Details window opens.
5. Complete this window as follows:

Signal Name

Type a unique name for the signal. If you are editing a signal, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Comment

Type a text comment for the signal. For example, you can add a comment to state which trigger fires when this signal is raised.

Parameters

This area displays the parameters that comprise the user-defined signal.

The order in which the parameters appear must match the order that they appear in the RAISE SIGNAL command for the trigger. You can use the up and down arrows to the right of the list box to change the order of any selected parameter. You can also click **Remove parameter** to the right of the list box to remove any selected parameter from the list.

To create a parameter, use the **Name** and **Data Type** fields, and the **Add parameter to the list** button.

Name Type a name for the parameter being created. Parameter names must be unique within the signal.

Data Type

Select the type of data the parameter can pass into the signal. The data type can be any valid ObjectServer data type except VARCHAR or INCR.

Data Length

For **Char** data types only, type the parameter length.

Add parameter to the list

After completing the **Name**, **Data Type**, and, where necessary, **Data Length** fields, click this button to add the parameter to the parameter list.

6. Save or cancel your changes as follows:

OK Click this button to save the user-defined signal and close the window. New user-defined signals are added to the User Defined Signals pane.

Cancel

Click this button to close the window without saving your changes.

Results

Example: User-defined signal and trigger

This example shows a user-defined signal called `illegal_delete`, and the `DETECT_AN_ILLEGAL_DELETE` database trigger in which it is used. The database trigger uses the signal to trap deletes that occur outside of standard office hours.

Within the User Defined Signal Details window, the user-defined signal called `illegal_delete` is created with the following entries:

Table 22. Entries for the `illegal_delete` user-defined signal in the User Defined Signal Details window

Field	Entry
Signal Name	<code>illegal_delete</code>
Comment	To be used with the <code>DETECT_AN_ILLEGAL_DELETE</code> trigger.
Parameters	<code>user_name Char(20)</code> <code>row_summary Char(20)</code> These read-only entries in the Parameters list are constructed from entries made in the Name , Data Type , and Data Length fields in the Parameters area. For example, for <code>user_name Char(20)</code> , the entries are: <ul style="list-style-type: none">• Name: <code>user_name</code>• Data Type: <code>Char</code>• Data Length: <code>20</code>

In the following SQL text for the `DETECT_AN_ILLEGAL_DELETE` pre-insert database trigger, 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, old.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

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

To delete a user-defined signal:

1. From the Netcool/OMNIBus Administrator window, select the **Automation** menu button.
2. Click **User Defined Signals**. The User Defined Signals pane opens.
3. Select the user-defined signal that you want to delete and click **Delete** in the toolbar. The user-defined signal is deleted.

Configuring the visual appearance of the event list

Conversions, colors, column visuals, and classes determine how alert information is displayed in the event list.

Creating and editing conversions

Conversions are associated with the columns in the ObjectServer alerts.status table, and they map integer values for the columns to strings. The conversions that are configured in the Netcool/OMNIBus Administrator are used in the event list, to translate integer values into strings, for readability.

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

To create or edit a conversion:

1. From the Netcool/OMNIBus Administrator window, select the **Visual** menu button.
2. Click **Conversions**. The Conversions pane opens.
To see the existing conversions for a column, double-click the column name. You can also click the round symbol (on UNIX) or the plus (+) symbol (on Windows) that is shown to the left of the column name.
3. To add a conversion, click **Add Conversion** in the toolbar. The Conversion Details window opens.
4. To edit a conversion, select the conversion to edit and then click **Edit Conversion** in the toolbar. The Conversion Details window opens.
5. Complete this window as follows:

Column

Select the name of the column containing the data to be converted.

Value Specify the integer value to be converted.

Conversion

Type the string to display instead of the value.

6. Save or cancel your changes as follows:

OK Click this button to save the conversion and close the window. New conversions are added to the Conversions pane.

Cancel

Click this button to close the window without saving your changes.

Deleting conversions

To delete a conversion:

1. From the Netcool/OMNIBus Administrator window, select the **Visual** menu button.
2. Click **Conversions**. The Conversions pane opens.
3. Select the conversion that you want to delete and click **Delete** in the toolbar. The conversion is deleted.

Creating and editing event severity colors for Windows event lists

In event lists, different colors are used to depict the different degrees of event severities. You can view, create, and modify the severity colors used in Windows event lists. You can select different colors for acknowledged and unacknowledged alerts.

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

1. From the Netcool/OMNIBus Administrator window, select the **Visual** menu button.
2. Click **Colors**. The Colors pane opens.
3. To add a severity color, click **Add Color Definition** in the toolbar. The Color Details window opens.
4. To edit a severity color, select the severity color to edit and then click **Edit Color Definition** in the toolbar. The Color Details window opens.
5. Complete this window as follows:

Severity

If you are creating a new color, specify the alert severity value.

Conversion

This field displays the conversion for this alert severity (if one exists). Conversions are used to translate integer values into strings for readability. For example, the default conversion for a severity of 4 is Major.

Unacknowledged

This area displays the color for the 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

Click the **Show color picker** button to select the color for unacknowledged alerts of that severity. From the resulting Color Picker dialog box, choose a color using its swatch, HSB, and RGB values, and then click OK.

Acknowledged

This area displays the color for the 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

Click the **Show color picker** button to select the color for acknowledged alerts of that severity. From the resulting Color Picker dialog box, choose a color using its swatch, HSB, and RGB values, and then click OK.

6. Save or cancel your changes as follows:

OK Click this button to save the color details and close the window. New severity colors are added to the Colors pane.

Cancel Click this button to close the window without saving your changes.

Creating and editing column visuals

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

To create or edit a column visual:

1. From the Netcool/OMNIBus Administrator window, select the **Visual** menu button.
2. Click **Column Visuals**. The Column Visuals pane opens.
3. To add a column visual, click **Add Column Visual** in the toolbar. The Column Visual Details window opens.
4. To edit a column visual, select the column visual to edit and then click **Edit Column Visual** in the toolbar. The Column Visual Details window opens.
5. Complete this window as follows:

Column

If you are creating a new column visual, select the column for which you are adding the visual.

Title Type the title that you want to display as the column header in Tivoli Netcool/OMNIBus event lists.

Default

Specify the default column width (in characters).

Maximum

Specify the maximum column width (in characters).

Title Select the justification or alignment of the column title.

Column

Select the justification or alignment of the information in the column.

6. Save or cancel your changes as follows:

OK Click this button to save the column visual details and close the window. New column visuals are added to the Column Visuals pane.

Cancel Click this button to close the window without saving your changes.

Deleting column visuals

To delete a column visual:

1. From the Netcool/OMNIbus Administrator window, select the **Visual** menu button.
2. Click **Column Visuals**. The Column Visuals pane opens.
3. Select the column visual that you want to delete and click **Delete** in the toolbar. The column visual is deleted.

Creating and editing classes

Events in the ObjectServer have a class that is assigned by the probe. Each class can be associated with an event list tool menu that contains useful tools for events of that class.

To create or edit a class:

1. From the Netcool/OMNIbus Administrator window, select the **Visual** menu button.
2. Click **Classes**. The Classes pane opens.
3. To add a class, click **Add Class** in the toolbar. The Class Details window opens.
4. To edit a class, select the class to edit and then click **Edit Class** in the toolbar. The Class Details window opens.
5. Complete this window as follows:

Identifier

If you are creating a new class, specify the class identifier for the class. Alerts in the ObjectServer are assigned a class identifier by the probe.

IBM defines class identifiers for particular equipment types. Contact IBM Support if you want to reserve a range of classes for your equipment type. A customer-reserved range of 88000 to 89000 is also available, which all customers are free to use.

Name Type a label for the equipment type to be associated with the class number.

6. Save or cancel your changes as follows:

OK Click this button to save the class details and close the window. New classes are added to the Classes pane.

Cancel

Click this button to close the window without saving your changes.

Deleting classes

To delete a class:

1. From the Netcool/OMNIbus Administrator window, select the **Visual** menu button.
2. Click **Classes**. The Classes pane opens.
3. Select the class that you want to delete and click **Delete** in the toolbar. The class is deleted.

Configuring ObjectServer databases, files, properties, connections, and channels

You can configure the following ObjectServer system components: ObjectServer database structures, files, properties, connections, and channels.

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

An ObjectServer file provides a way to log or report information about ObjectServer events.

ObjectServer properties control the behavior of the ObjectServer.

Connections to the ObjectServer can be viewed and managed.

Channels are used to define the type of event data to broadcast in accelerated event notifications, and the data recipients.

Related concepts

Chapter 6, “Configuring accelerated event notification,” on page 239

Configuring databases

Database configuration involves the creation and maintenance of database tables, and columns within the tables.

You can create and drop databases, and you can create, drop, and edit (or alter) database tables and columns.

Restriction: You are not permitted to make changes to system databases. In Netcool/OMNIbus Administrator, system databases are shown with a lock icon next to them.

Related concepts

“System-initialized databases” on page 143

Creating databases

You can use Netcool/OMNIbus Administrator to create and manage ObjectServer databases.

To create an ObjectServer database:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Databases**. The Databases, Tables and Columns pane opens.

You can view and configure table and column information for each non-system database listed. Use the **Data View** tab to view data contained in the table. Use the **Column Definitions** tab to see information about the table columns, such as their data types and attributes.

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

3. From the toolbar, click **Create Database**. The Database Details window opens.
4. Complete this window as follows:

Name Type a unique name for the database.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

OK Click this button to save the database and close the window. New databases are added to the ObjectServer and to the Databases, Tables and Columns pane.

Cancel Click this button to close the window without saving your changes.

What to do next

You can now add tables to the database.

Related tasks

“Creating tables”

Creating tables

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.

To create a table:

1. From the Netcool/OMNIBus Administrator window, select the **System** menu button.
2. Click **Databases**. The Databases, Tables and Columns pane opens.
3. Select the database in which you are creating the table.
4. From the toolbar, click **Create Table**. The Table Details window opens.
5. Complete this window as follows:

Name Type the table name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Type Select one of the following table types:

- **Persistent:** When the ObjectServer restarts, a persistent table is recreated with all data.
- **Virtual:** When the ObjectServer restarts, a virtual table is re-created with the same table description, but without any data.

Table area

This area lists details for all columns in the table. You can use the up and down arrows to the right to change the order of a selected column in the table.

Add column

Click this button if you want to add a new column to the table. The Column Details window opens. Complete this window and save your changes.

When you return to the Table Details window, the new column is added to the list of columns.

Edit column

Click this button if you want to edit the details of a selected column. The Column Details window opens. Edit the details and save your changes.

When you return to the Table Details window, the updates to the column details are reflected in the column list.

Delete column

Click this button if you want to drop a selected column from the table. No confirmation is required for the deletion.

6. Save or cancel your changes as follows:

OK Click this button to save the table details and close the window. New tables are added to the Databases, Tables and Columns pane.

Cancel

Click this button to close the window without saving your changes.

Results

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

Related tasks

“Adding and editing table columns”

Adding and editing table columns

To add or edit a table column:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Databases**. The Databases, Tables and Columns pane opens.
3. Select the table in which you are adding or editing the column.
4. Click the **Column Definitions** tab.
5. To add a column, click **Add Column** in the toolbar. The Column Details window opens.
6. To edit a column, select the column to edit and then click **Edit Column** in the toolbar. The Column Details window opens.
7. Complete this window as follows:

Column Name

Type the column name. If you are editing the column, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in

length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Data Type

Select a data type for the column. 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. You can select from the following data types:

- Integer: 32-bit signed integer
- UTC: Time, stored as the number of seconds since midnight January 1, 1970
- VarChar: Variable size character string, up to 8192 Bytes in length
- Incr: 32-bit unsigned auto-incrementing integer that can only be updated by the system
- Char: Fixed size character string, up to 8192 Bytes in length
- Unsigned: 32-bit unsigned integer
- Boolean: TRUE or FALSE
- Real: 64-bit signed floating point number
- Integer64: 64-bit signed integer
- Unsigned64: 64-bit unsigned integer

Length

This field is available only when you select VarChar or Char from the **Data Type** list. Specify the column length.

Primary Key

Select this check box to indicate that the column is a primary key. The primary key column or columns uniquely identify each row. A primary key column must have a default value.

No Modify

Select this check box to indicate that users cannot modify data in this column.

No Default

Select this check box to indicate that a value must be specified for this column in any INSERT command.

8. Save or cancel your changes as follows:

Apply If you want to add multiple columns without exiting the Column Details window, click this button to save the column details after adding each set of entries.

OK After entering the values for the last column that you want to add at the present time, click this button to save the column details and close the window. New columns are added to the Databases, Tables and Columns pane.

Cancel

Click this button to close the window without saving your changes.

Results

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 KB.

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

Deleting databases

You are not permitted to drop system databases, which have a lock icon next to them.

Attention: When you drop a database that contains table data, the tables are first emptied and dropped.

To delete a database:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Databases**. The Databases, Tables and Columns pane opens.
3. Select the database that you want to delete and click **Drop Database** in the toolbar. The database is removed from the ObjectServer.

Deleting tables

You are not permitted to delete tables in system databases.

To delete a table:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Databases**. The Databases, Tables and Columns pane opens.
3. Select the database containing the table to drop.
4. Select the table that you want to delete and click **Drop Table** in the toolbar. All data is removed from the table and the table is then removed from the database.

Deleting table columns

You are not permitted to delete primary key columns or columns in system tables.

Attention: Dropping a column requires a considerable amount of preliminary action to identify and remove any external dependencies on the column. This involves searching for any references to the column within triggers, procedures, views, restriction filters, probes rules files, and gateway mapping files. Be aware also that if you drop a column that has triggers, procedures, views, or restriction filters that depend on it, these dependent objects are also deleted, with a warning being written to the ObjectServer log file.

To delete a table column after confirming that no external dependencies exist:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Databases**. The Databases, Tables and Columns pane opens.
3. Select the table containing the column to delete.
4. Click the **Column Definitions** tab.
5. Select the column that you want to delete and click **Drop Column** in the toolbar. The column is removed from the table.

Related reference

“Dropping a column” on page 148

Viewing and changing ObjectServer properties

ObjectServer properties help to determine the behavior of the ObjectServer. You can view and change ObjectServer properties using Netcool/OMNIbus Administrator. You cannot add ObjectServer properties; you can only edit existing ones.

The default location of the ObjectServer properties file is `$NCHOME/omnibus/etc/servername.props`. The ObjectServer reads the properties file when it starts.

Important: It is essential that you are familiar with the ObjectServer properties before modifying them. Incorrect configuration can negatively impact system performance and functionality.

To change the value of an ObjectServer property:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Properties**. The ObjectServer Properties pane opens.

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

3. To edit a property, select the column to edit and then click **Edit Property** in the toolbar. The Property Details window opens.
4. Complete this window as follows:

Name The unique name allocated to the ObjectServer property is shown here. You cannot change this value.

Description

A description of the ObjectServer property is shown here. You cannot change this value.

Value Edit the property value as required.

5. Save or cancel your changes as follows:

OK Click this button to save the property details and close the window. The updated value is reflected in the ObjectServer Properties pane.

Cancel

Click this button to close the window without saving your changes.

Results

Tip: Changes to some ObjectServer properties do not take effect until you restart the ObjectServer. These properties have the text `false` in the **Immediate** column.

Related reference

“ObjectServer properties and command-line options” on page 3

Configuring ObjectServer files

ObjectServer files are user-defined storage objects that hold log or report data.

An ObjectServer file is a logical file that 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.

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 KB 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 KB).
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 repeats until the file is altered or dropped.

Creating and editing ObjectServer files

An ObjectServer file provides a way to log or report information about ObjectServer events.

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.

To create or edit an ObjectServer file:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Log Files**. The Log Files pane opens.
3. To add a file, click **Add Log File** in the toolbar. The File Details window opens.
4. To edit the file details, select the file to edit and then click **Edit Log File** in the toolbar. The File Details window opens.
5. Complete this window as follows:

Name Type a unique name for the ObjectServer file; for example, a name that provides some meaning about its usage. Note that this is not the file name as it will be created on the file system; to specify this, use the **Full File Path** field. If you are editing a file, you cannot change the name.

Tip: When creating ObjectServer objects, their names must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (_) characters, up to 40 characters in

length. *User, group, and role names can be any text string up to 64 characters in length and can include spaces.* Names of ObjectServer objects are case-sensitive.

Full File Path

Type the full path and file name of the physical file; for example, `/opt/netcool/omnibus/log/status.log`.

Note: A number is automatically appended to the file name on the file system.

Enabled

Select this check box to activate the ObjectServer file. If not activated, the file will exist on the file system, but cannot be written to. You can specify the ObjectServer file information and then activate the file at a later time.

Unlimited File Size

Select this check box if you want information to be written to a single file with an unlimited size. If you choose this setting, the **Max. Size** and **Max. Files** fields are not displayed in the window.

When this check box is clear, information can be written to a single file, or to a pool of files that will each be written to in turn when a specified maximum size is reached. If you choose this setting, you must specify associated values in the **Max. Size** and **Max. Files** fields.

Truncate

Click this button to clear any information that has been written to the physical file. This 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.

Note: This button is visible only when you are editing file details.

Max. Size

Specify the maximum ObjectServer file size and then select a unit of measurement. The minimum file size is 1 KB and the maximum file size is 4 GB.

Note: The operating system may place further restrictions on the maximum size of a single file.

Max. Files

Specify the maximum number of ObjectServer files to create.

6. Save or cancel your changes as follows:

OK Click this button to save the file details and close the window. New file details are added to the Log Files pane.

Cancel

Click this button to close the window without saving your changes.

Related concepts

“ObjectServer file creation sequence” on page 128

Deleting ObjectServer files

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

To delete an ObjectServer file:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Log Files**. The Log Files pane opens.
3. Select the ObjectServer file that you want to delete and click **Delete** in the toolbar. The ObjectServer file is deleted. The ObjectServer no longer writes information to this file.

Results

When you delete a file, only the ObjectServer file is deleted; physical files created in the file system are not deleted.

Monitoring ObjectServer connections

You can view all current connections to the ObjectServer and disconnect one or more of the connections. You must be assigned the ALTER SYSTEM DROP CONNECTION permission to disconnect ObjectServer connections.

To view and disconnect connections to the ObjectServer:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Connections**. The ObjectServer Connections pane opens, showing a row of information for each application that is currently connected.
3. Select the rows that you want to disconnect. You can use the Shift key for consecutive selections, or the Ctrl key for non-consecutive selections.
4. Click **Disconnect** in the toolbar. You are prompted in turn for confirmation that you want to disconnect each of the applications that you selected.
5. Click **Yes** for each application to be disconnected, and click **No** to cancel a disconnection.

Related tasks

“Viewing user connections to the ObjectServer” on page 83

Configuring channels

The Accelerated Event Notification (AEN) system enables you to accelerate high-priority events to help ensure that systems can continue to run without interruption. Use channels to define the type of event data to be included in accelerated event notifications, and the recipients of this event data.

Related concepts

Chapter 6, “Configuring accelerated event notification,” on page 239
“Configuring channels to broadcast event data” on page 241

Related tasks

“Creating and editing channels” on page 241
“Copying and pasting channels” on page 244
“Deleting a channel” on page 245
“Sending messages to channel recipients” on page 245
“Disconnecting Accelerated Event Notification clients” on page 245
“Shutting down Accelerated Event Notification clients” on page 246

Using the SQL interactive interface in GUI mode

You can use the SQL interactive interface to configure the ObjectServer by issuing SQL commands.

Note: Only users that are members of a group granted the ISQL role can access an ObjectServer by using the SQL interactive interface. Only users that are members of a group granted the ISQLWrite role can update ObjectServer data by using the SQL interactive interface.

To open the SQL interactive interface in GUI mode:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **SQL**. The SQL pane opens.
3. Complete this window as follows:









SQL editor

Use the text field and buttons in this area to issue commands. Type SQL in the text field, and use a semi-colon to separate multiple commands. You can use the SQL helper buttons and the additional buttons to facilitate the creation of SQL commands.

When typing SQL commands within the Tivoli Netcool/OMNIbus SQL editor panels, you can type one or more characters and then press Ctrl+F1 to obtain a dialog box with a list of keywords that might match your entry. Select the required keyword and click **OK** to complete your entry. If only one keyword matches your typed characters, the keyword is automatically completed for you. If you press Ctrl+F1 after typing a database-related keyword, the dialog box provides a list of possible ObjectServer databases from which you can select. If you press Ctrl+F1 after typing a database name followed by a dot (for example: alerts.), you can press Ctrl+F1 again to view and select from a list of tables in the database.

The following table describes the helper buttons.

Table 23. SQL interactive interface buttons

Button	Description
	<p>Click this button to select an SQL command from the pop-up menu. Based on the command that you select, complete the resulting window as follows:</p> <ul style="list-style-type: none"> • Select: Select the database and table on which to run the SELECT command. Then, choose the table columns to select. • Insert: Select the database and table on which to run the INSERT command. Then, select the table columns in which to insert values. For each selected column, enter the value to insert. For insert statements, you must include the primary key. Primary keys are indicated with an asterisk (*). • Update: Select the database and table on which to run the command. Then, select the table columns to update. For each selected column, enter the new value. For update statements, you must exclude the primary key. Primary keys are indicated with an asterisk (*). <p>Note: For inserts and updates to the alerts.status table, any existing conversions appear in the drop-down lists.</p> <ul style="list-style-type: none"> • Delete: Select the table to delete. • Use: Select the database to use. • Service: Select a service name and a value. Values can be Good, Marginal, or Bad.
	<p>Click this button to select a table column name to add to the command. The column name is substituted for the corresponding event list row value when the tool runs. 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: RemoteNodeAlias = '@LocalNodeAlias'</p>
	<p>Click this button to select from a list of available conversions. Double-click to add the conversion.</p>
	<p>Click this button to clear the entered SQL.</p>
	<p>Click this button to bring up a list of keywords that complete the entered SQL.</p>
	<p>Click this button to check the validity of the entered SQL syntax.</p>
	<p>Click this button to locate a file of type .sql or .ed and check the validity of its syntax. On completion, the results are displayed. (When you use an external editor to create or edit triggers and procedures, they are saved as .ed files.)</p>
	<p>Click this button to submit the SQL commands.</p>

After you complete the SQL command, click **Submit**.

History

This drop-down list provides a history of the SQL commands entered. You can select a previously-issued command from the list. You can also clear the list of previously-issued commands by right-clicking over the list and selecting **Clear History**.

Result View

After issuing the command, a visual representation of the table on which you performed the SQL command is displayed in this tab.

Console View

A command history is displayed in this tab.

Related concepts

Chapter 5, “ObjectServer SQL,” on page 135

Related tasks

“Creating and editing conversions” on page 118

Chapter 5. 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 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 events automatically.

You can use the SQL interactive interface to connect to an ObjectServer and run ObjectServer SQL commands.

Tip: Many of the tasks performed by running ObjectServer SQL commands from the SQL interactive interface can also be performed from the Netcool/OMNIbus Administrator interface.

Related tasks

“Using the SQL interactive interface in GUI mode” on page 131

SQL interactive interface

You can 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.

While running the SQL interactive interface, you can perform tasks such as creating a new database table or stopping the ObjectServer.

Note: Only users who are members of a group granted the ISQL role can connect to an ObjectServer by using the SQL interactive interface. Only users who are members of a group granted the ISQLWrite role can modify ObjectServer data by using the SQL interactive interface. These roles are predefined in Tivoli Netcool/OMNIbus.

Related concepts

“Configuring roles” on page 71

“Configuring groups” on page 76

“Using roles to assign permissions to users” on page 188

Starting the SQL interactive interface

Before you start the SQL interactive interface, you are required to connect to an ObjectServer as a specific user.

To start the SQL interactive interface:

Run the **nco_sql** command on UNIX and **isql** command on Windows, as follows:

Option	Description
UNIX	<code>\$NCHOME/omnibus/bin/nco_sql -server <i>servername</i> -user <i>username</i></code>
Windows	<code>%NCHOME%\omnibus\bin\isql -S <i>servername</i> -U <i>username</i></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).

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

Results

Attention: Be aware that specifying the password on the command line makes the password visible. If you do not specify a password, you are prompted for one.

A number of command-line options are available for use with these commands.

Command-line options for starting the SQL interactive interface

When you use the **nco_sql** or **isql** command to start the SQL interactive interface, you can specify a number of command-line options to modify the configuration. You can run the **nco_sql** or **isql** command from the `$NCHOME/omnibus/bin` directory.

The command-line options for the SQL interactive interface are described in the following table.

Table 24. Command-line options for the *nco_sql* and *isql* commands

Option	Description
<code>-help</code>	Displays help information about the command-line options and exits.

Table 24. Command-line options for the `nco_sql` and `isql` commands (continued)

Option	Description
<p><code>-networktimeout integer</code></p> <p><code>-l logintimeout</code> and <code>-t timeout</code> on Windows</p>	<p>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 SQL interactive interface attempts to reconnect to the ObjectServer. If the connection is unsuccessful after a second timeout period, the SQL interactive interface will attempt to connect to a backup ObjectServer, where available. By default, no timeout is specified.</p> <p>On Windows, <code>-l</code> specifies the maximum timeout value allowed when connecting to the server, and <code>-t</code> specifies the number of seconds before a command times out. If you do not specify a timeout value, a command runs indefinitely. This affects commands issued from within <code>isql</code>, not the connection time. The default timeout for logging into <code>isql</code> is 60 seconds.</p> <p>Note: The <code>nco_sql</code> command runs <code>nco_get_login_token</code> to obtain a login token and then runs the SQL interactive interface (<code>isql</code>) with this token. The specified network timeout is passed to both the <code>nco_get_login_token</code> and <code>isql</code> binaries when they are launched. If you run <code>nco_sql</code> with <code>-secure</code>, do <i>not</i> set the timeout to a value greater than 14 seconds because a secure login token is only valid for 15 seconds. A larger timeout can be used with the <code>-nosecure</code> option.</p>
<code>-nosecure</code>	When specified, login information is not encrypted when it is transmitted between components.
<p><code>-password password</code></p> <p><code>-P password</code> on Windows</p>	<p>Specifies the password for the user.</p> <p>On Windows, if you are using an empty password, such as with the default NCOMS ObjectServer, the <code>-P</code> option must be specified as the last item. For example:</p> <pre>"%NCHOME%\omnibus\bin\isql" -U root -S NCOMS -i update71to72.sql -P</pre> <p>Attention: The password is visible if it is specified on the command line. If not specified, you are prompted for the password.</p>
<code>-secure</code>	<p>When specified, login information is automatically encrypted when it is transmitted between components.</p> <p>This is the default for all supported releases of Tivoli Netcool/OMNIBus.</p>
<p><code>-server servername</code></p> <p><code>-S servername</code> on Windows</p>	Specifies the name of the ObjectServer to which to connect. The default is NCOMS.
<p><code>-user username</code></p> <p><code>-U username</code> on Windows</p>	<p>Specifies the name of a Tivoli Netcool/OMNIBus user. The default is the user running the command.</p> <p>Note: The SQL interactive interface does not allow spaces in user names.</p>

After connection, you can enter ObjectServer SQL commands.

Related concepts

“Running the SQL interactive interface in secure mode” on page 141

Related tasks

“Running SQL commands in the SQL interactive interface”

Running SQL commands in the SQL interactive interface

After connecting to the SQL interactive interface with a user name and password, a numbered prompt is displayed. Enter ObjectServer SQL commands at the prompt.

The prompt is shown as:

1>

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 Enter.

Note: The `nco_sql` utility does not allow whitespace preceding the `go` keyword. For example, if you run `nco_sql` non-interactively from a script, and use whitespace to indent the `go` keyword, the SQL statements will fail.

- Multiple commands, separated by a semicolon, can be run with a single `go` command.
- You can enter a command of up to 4094 characters.

Additionally:

- To cancel a command, enter `reset` at the beginning of a new line, or press Ctrl+C anywhere on a line. Any commands that have not been run are discarded.
- To run the default editor (as defined by the EDITOR environment variable) in the `nco_sql` utility, 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.

SQL syntax notation

An SQL syntax notation is used to describe ObjectServer SQL commands that you can run from the SQL interactive interface.

An example syntax notation for table creation is as follows:

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

Tip: When entering an SQL command, you must specify the keywords in the order listed in the syntax descriptions.

The following table describes the syntax notation used for SQL commands.

Table 25. SQL syntax notation

Syntax	Description
{ 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.

Within the syntax:

- SQL keywords are shown in uppercase; for example, CREATE, TABLE, and PERSISTENT. Note, however, that SQL keywords are not case-sensitive, and can appear in uppercase, lowercase, or mixed case.
- Variable values are depicted using italic text. For example, *database_name* requires the entry of an actual database name, *table_name* requires the entry of an actual table name, *column_name* requires the entry of an actual column name, and *data_type* requires the entry of an actual data type.

Naming conventions for ObjectServer objects

When issuing SQL commands, you must adhere to the naming conventions defined for ObjectServers.

The name of an ObjectServer must consist of 29 or fewer uppercase letters and cannot begin with an integer.

When creating an 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.

Note: User, group, and role names can be any text string enclosed in quotation marks up to 64 characters in length.

Names of ObjectServer objects and identifiers are case-sensitive.

Specifying paths in the SQL interactive interface

Some SQL commands require you to enter path names.

For example, on UNIX, you can create a file by entering the following command:

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

On Windows, you must escape the backslash character in file paths or the path 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 UNIX path separator when specifying paths on Windows. 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 function is useful when you need to perform repetitive tasks.

The text file must contain only SQL commands and must end with the **go** keyword. Otherwise, the commands will not be processed.

For example, to run 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
```

Windows You cannot use the std redirection characters `<` and `>` to redirect input to a text file, or redirect output from a text file. Instead, use the command-line argument `-i filename` in place of `<` and use the command-line argument `-o` in place of `>`.

Example: SQL interactive interface session on UNIX

This example shows an SQL interactive interface session on UNIX, for running **nco_sql** and entering commands.

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

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

The results of the command are displayed.

Running the SQL interactive interface in secure mode

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.

The SQL interactive interface runs in secure mode unless you specify the `-nosecure` command-line option when starting the SQL interactive interface.

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.

Related tasks

“Starting the SQL interactive interface” on page 136

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**. This is applicable only when running in non-FIPS 140-2 mode.

When running in FIPS 140-2 mode, leave the passwords in plain text in the scripts, or use the **nco_aes_crypt** utility with the `-d` option to decrypt any sensitive data before use.

To encrypt and use a plain text password in non-FIPS 140-2 mode:

1. Enter the following command:

```
$NCHOME/omnibus/bin/nco_sql_crypt plaintext_password
```

In this command, *plaintext_password* represents 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.

Results

Passwords encrypted by using the **nco_sql_crypt** utility are decrypted by the ObjectServer when the connection is made.

Exiting the SQL interactive interface

To exit the SQL interactive interface:

Perform the appropriate action for your operating system:

Option	Description
UNIX	Press Ctrl+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.

Creating, altering, and dropping ObjectServer objects

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

You can use Data Definition Language (DDL) commands to create, alter, and drop ObjectServer objects. The following table lists each object and the DDL commands that can be used.

Table 26. ObjectServer objects and associated DDL commands

ObjectServer object	Allowed DDL commands
DATABASE	CREATE DATABASE DROP DATABASE
TABLE	CREATE TABLE ALTER TABLE DROP TABLE
INDEX	CREATE INDEX DROP INDEX
VIEW	CREATE VIEW DROP VIEW
RESTRICTION FILTER	CREATE RESTRICTION FILTER DROP RESTRICTION FILTER
FILE	CREATE FILE ALTER FILE DROP FILE

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.

You can create and drop databases using ObjectServer SQL.

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 ObjectServer naming conventions.

A database is always persistent.

Example

```
create database mydb;
```

Related concepts

“Naming conventions for ObjectServer objects” on page 139

Dropping a database

Use the DROP DATABASE command to drop an existing database.

You cannot drop a database if it contains any objects. You also cannot drop the security or catalog databases, which are system-initialized databases.

Syntax

```
DROP DATABASE database_name;
```

Example

```
drop database mydb;
```

Related concepts

“System-initialized databases”

System-initialized databases

When you initialize an ObjectServer, a number of default databases are created.

The following table describes these system-initialized databases.

Table 27. System-initialized databases

Database name	Type of database	Description
security	System	Contains information about the security system, including users, roles, groups, and permissions.
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 IBM Tivoli Composite Application Manager for Internet Service Monitoring.
custom	User	Can be used for tables added by users.
persist	System	Records internal ObjectServer state information.
transfer	System	Used internally by the ObjectServer unidirectional and bidirectional gateways to synchronize security information between ObjectServers.
master	User	Used for compatibility with prior releases of Tivoli Netcool/OMNIBus. Tables in the master database also support the desktop ObjectServer architecture. For details about the desktop ObjectServer architecture, see the <i>IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide</i> .

Table 27. System-initialized databases (continued)

Database name	Type of database	Description
tools	User	Used for compatibility with prior releases of Tivoli Netcool/OMNIBus.
iduc_system	User	Contains all of the required IDUC application support tables for accelerated event notification, sending information messages, and invoking commands.
precision	User	Used by IBM Tivoli Network Manager IP Edition to implement the service-affected events application.

Restriction: The ObjectServer maintains system databases. You can view, but cannot modify, the data in them.

Related concepts

“Configuring databases” on page 122

Related reference

Appendix A, “ObjectServer tables,” on page 321

Tables

The main storage structure of the ObjectServer 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 ObjectServer naming conventions.

The storage type is either PERSISTENT or VIRTUAL. A persistent table is re-created, 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, you must specify the column name and data type, and can also specify optional properties.

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 KB.

Example

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

Related concepts

“Naming conventions for ObjectServer objects” on page 139

Specifying data types for columns:

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.

When creating a table using the CREATE TABLE command, you must specify a data type for each column that you define. The data types supported by the ObjectServer are listed in the following table.

Table 28. ObjectServer data types

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 28. ObjectServer data types (continued)

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 char type is identical in operation to varchar, 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 varchar type uses less storage space than the char 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.

Related reference

“Creating a table” on page 144

“alerts.status table” on page 321

“Creating a user-defined signal” on page 213

Specifying optional properties for columns:

You can specify optional properties for the columns that you define when creating a table.

The optional column properties are described in the following table.

Table 29. 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.

Table 29. Column properties (continued)

Column property	Description
NODEFAULT	The value of this column must be specified in the initial INSERT command. You use the INSERT command to insert a new row of data into an existing table.
NOMODIFY	The value of this column cannot be changed after the initial INSERT 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.

In the CREATE TABLE command, the syntax for specifying which columns are primary keys is as follows:

```
(column_name data_type [ PRIMARY KEY | NODEFAULT | NOMODIFY | HIDDEN ],...
[, PRIMARY KEY(column_name,...) ] );
```

Based on this syntax, you can create columns as primary keys in either or both of the following ways:

- Specify the PRIMARY KEY column property as part of a column definition.
- 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.

Related reference

“Creating a table” on page 144

“Inserting a new row of data into a table (INSERT command)” on page 173

Altering a table

Use the ALTER TABLE command to change the characteristics of an existing table and its columns. You can add, drop, and alter columns.

Restriction: You cannot alter system tables that contain metadata about ObjectServer objects.

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 }
ALTER [COLUMN] column_name SET WIDTH value;
```

You can specify more than one ADD, DROP, or ALTER setting in a single ALTER TABLE command.

Example

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

Related concepts

“System tables” on page 151

Adding a column:

To add columns to an existing table, use the ADD COLUMN setting with the ALTER TABLE command.

In this command, the syntax for adding columns is as follows:

```
ADD [COLUMN] column_name data_type [ NODEFAULT | NOMODIFY | HIDDEN ]
```

When you add columns, you must specify the column name and data type. You can also specify optional properties.

You cannot add primary keys to an existing table.

When a new column is added to the table using the NODEFAULT clause, any INSERT statements that are sent from the probes or gateways fail. The failure occurs because they do not comply with the NODEFAULT constraint.

When you mark a column as NODEFAULT and a row is inserted into that table, the insert statement must explicitly set a value for that column or the INSERT statement becomes invalid.

This occurs because the INSERT statement will not attempt to populate the NODEFAULT column. The trigger will also be invalid and cannot be successfully recompiled.

Related reference

“Altering a table” on page 147

“Specifying data types for columns” on page 145

“Specifying optional properties for columns” on page 146

Dropping a column:

To drop columns from an existing table, use the DROP COLUMN setting with the ALTER TABLE command.

In this command, the syntax for dropping columns is as follows:

```
DROP [COLUMN] column_name
```

You cannot drop a column if the column is a primary key.

When dropping a column, a considerable amount of preliminary action is required to identify and remove any external dependencies on the column. You must search for any references to the column within triggers, procedures, views, and restriction filters by querying the relevant database tables. You must also search your probe rules files and gateway mapping files for references to the column.

Attention: If you drop a column on which triggers, procedures, views, restriction filters, or indexes depend, these dependent objects are also deleted, and a warning is written to the ObjectServer log file. To avoid inadvertently deleting triggers, procedures, views, or restriction filters, read the following guidelines for dropping columns. (Because indexes are directly linked to columns, indexes are always deleted when their associated columns are dropped.)

The following guidelines are based on an example scenario where you want to drop the Country column from your ObjectServer:

1. Connect to your ObjectServer (for example, OWL) using the SQL interactive interface, as shown in the following table. Your user name is assumed by default, but you are required to enter your password.

Table 30. Starting the SQL interactive interface

Option	Description
UNIX	Enter: \$NCHOME/omnibus/bin/nco_sql -server OWL
Windows	Enter: %NCHOME%\omnibus\bin\isql -S OWL

2. Back up your ObjectServer to a temporary location (for example, /tmp/mybackup) using the ALTER SYSTEM BACKUP command. This precautionary measure ensures that you can restore your system if required.
1> alter system backup '/tmp/mybackup';
2> go
3. List details of your triggers, as stored in the catalog.triggers table:
1> describe catalog.triggers;
2> go
The type of key, name, data type, and length of each column in the table are output to the screen.
4. Retrieve the names of all triggers that reference Country in the body of the trigger:
1> select TriggerName from catalog.triggers where CodeBlock like 'Country';
2> go
The trigger names of all affected triggers are listed.
5. Make a note of all listed triggers and remove the Country references by editing each trigger. You can do this from the Trigger Details window (**Action** tab) in the Netcool/OMNibus Administrator.
6. Repeat steps 3 to 5 to identify any other objects that reference the Country column, and to remove all instances of the reference. The following table lists the database tables that you need to search, the relevant SELECT statements, and the Netcool/OMNibus Administrator windows that you can use to edit the object.

Table 31. System catalog tables to be searched, SELECT statements, and Netcool/OMNibus Administrator windows

Object type	Table name	SELECT statement	Netcool/OMNibus Administrator window
Procedures	catalog.sql_procedures	select ProcedureName from catalog.sql_procedures where CodeBlock like 'Country';	SQL Procedure Details window
Restriction Filters	catalog.restrictions	select RestrictionName from catalog.restrictions where ConditionText like 'Country';	Restriction Filter Details window
Views	catalog.views	select ViewName from catalog.views where CreationText like 'Country';	

7. Search your probe rules files \$NCHOME/omnibus/probes/arch/*.rules and remove any references to the column.

8. Search your gateway mapping files \$NCHOME/omnibus/gates/objserv_type/objserv_type.map, where *type* represents uni or bi. Remove any references to the column.
9. After all the references have been removed, drop the Country column using the ALTER TABLE ... DROP COLUMN syntax.

Related concepts

“Retrieving data from a table or view (SELECT command)” on page 175

Related reference

“Changing the default and current settings of the ObjectServer (ALTER SYSTEM command)” on page 181

“Displaying details of columns in a table or view (DESCRIBE command)” on page 180

“Altering a table” on page 147

Altering a column:

To alter columns in an existing table, use the ALTER COLUMN setting with the ALTER TABLE command.

In this command, the syntax for altering columns is as follows:

```
ALTER [COLUMN] column_name SET NOMODIFY { TRUE | FALSE }  
ALTER [COLUMN] column_name SET HIDDEN { TRUE | FALSE }  
ALTER [COLUMN] column_name SET NODEFAULT { TRUE | FALSE }  
ALTER [COLUMN] column_name SET WIDTH value
```

Use the following guidelines to alter column properties:

- To alter the NOMODIFY, HIDDEN, and NODEFAULT properties of an existing column, set the appropriate property to TRUE or FALSE. A primary key column must have a default value and cannot be hidden.
- To alter the width of a column with a data type of varchar, use the WIDTH property and specify the *value* setting as a length in bytes. You cannot alter the width of primary keys.

When you mark a column as NODEFAULT and a row is inserted into that table, the insert statement must explicitly set a value for that column or the INSERT statement becomes invalid.

This occurs because the INSERT statement will not attempt to populate the NODEFAULT column. The trigger will also be invalid and cannot be successfully recompiled.

Related reference

“Specifying optional properties for columns” on page 146

“Altering a table” on page 147

Dropping a table

Use the DROP TABLE command to drop an existing table.

You cannot drop a table if it is referenced by other objects, such as triggers, or if it contains any data. You also cannot drop system tables, which hold metadata about ObjectServer objects.

Syntax

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

Example

To delete all rows of a table:

```
delete from mytab;
```

To drop the table:

```
drop table mytab;
```

Related concepts

“System tables”

System tables

System tables are special tables maintained by the ObjectServer, and they 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 by using the SELECT and DESCRIBE commands, but you cannot add, modify, or delete system tables or their contents by using ObjectServer SQL.

Related concepts

“Retrieving data from a table or view (SELECT command)” on page 175

Related reference

“Displaying details of columns in a table or view (DESCRIBE command)” on page 180

“System catalog tables” on page 338

Indexes

You can use indexes to improve the performance of the ObjectServer database. The use of well-designed indexes can reduce or eliminate the need for full table scans during the execution of SQL queries, and result in faster data retrieval.

Creating an index

Use the CREATE INDEX command to create an index on a database table.

Tip: SQL query guidelines and indexing guidelines are available to help you determine which columns to index, and what type of index to create for a column.

Syntax

```
CREATE INDEX index_name
ON database_name.table_name
[USING { HASH | TREE }] (column_name);
```

The *index_name* value must be unique within the ObjectServer and comply with the ObjectServer naming conventions. For ease of identification and uniqueness, consider using a naming convention for your indexes; for example, *column_nameIdx* or *column_nameIndex*, where *column_name* is the name of the column.

The table name specified after the ON keyword must be fully qualified with the database name; for example, alerts.status.

Restriction: You cannot create indexes on the columns in system tables. These tables contain metadata about ObjectServer objects and are stored in the catalog database.

Use the optional USING setting to create a hash or tree index. If omitted, a hash index is created by default. A hash index is appropriate for use only with SQL queries that denote equality. A tree index can additionally be used for ordered queries.

Restriction: You cannot create a hash index on a single primary key field. You cannot create a tree index on columns with Boolean data values.

You must specify the name of the single column that is being indexed.

Example

```
create index SeverityIdx on alerts.status (Severity);
create index ExpireTimeIdx on alerts.status using tree (ExpireTime);
```

Related concepts

“Naming conventions for ObjectServer objects” on page 139

Related reference

“SQL query guidelines” on page 311

“Indexing guidelines” on page 314

“System catalog tables” on page 338

Dropping an index

Use the DROP INDEX command to remove a redundant index on a database table.

Syntax

```
DROP INDEX index_name;
```

The *index_name* value is the unique name for the index being dropped.

Note: If an indexed column is dropped, the index is automatically dropped.

Example

```
drop index SeverityIdx;
```

Viewing index details

To see what columns are currently indexed, you can examine the contents of the catalog.indexes table either from the Netcool/OMNIbus Administrator interface, or by using the SELECT command.

To view index details in the catalog.indexes table, perform either of the following steps:

- From the Netcool/OMNIbus Administrator window:
 1. Select the **System** menu button.
 2. Click **Databases**. The Databases, Tables and Columns pane opens.
 3. Select **catalog.indexes**.
 4. Click the **Data View** tab on the Databases, Tables and Columns pane to view the table data.
- From the SQL interactive interface, enter the following command:

```
select * from catalog.indexes;
```

Related concepts

"SQL interactive interface" on page 135

"Retrieving data from a table or view (SELECT command)" on page 175

Related tasks

"Starting Netcool/OMNIbus Administrator" on page 60

Related reference

"catalog.indexes table" on page 348

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.

Note: Views are primarily intended for internal use. Do not use views in automations.

Related concepts

"Expressions" on page 171

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 there is a possibility that a view already exists 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 ObjectServer naming conventions. 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), 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.

Example

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

Related concepts

“Naming conventions for ObjectServer objects” on page 139

“Retrieving data from a table or view (SELECT command)” on page 175

Dropping a view

Use the DROP VIEW command to drop an existing view.

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

Syntax

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

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

Example

```
drop view myview;
```

Restriction filters

A restriction filter provides a way to restrict the rows that are displayed when a user views a table.

After the restriction filter has been assigned to a user or group, the restriction filter controls the data that can be displayed and modified from client applications, and modified in INSERT, UPDATE, and DELETE commands. Only rows that meet the criteria specified in the restriction filter condition are returned.

You can assign only one restriction filter per table to a user or a group. If multiple restriction filters apply to a user or group, the resulting data is a combination of all applicable restriction filters for the user or group.

If you are using multiple restriction filters, make sure that you have set the ObjectServer **RestrictionFiltersAND** property appropriately.

Related reference

“ObjectServer properties and command-line options” on page 3

“Modifying the details of an existing user (ALTER USER command)” on page 185

“Modifying the details of an existing group (ALTER GROUP command)” on page 186

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 there is a possibility that a restriction filter already exists 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 restriction 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 ObjectServer naming conventions.

The table name specified after the ON keyword must be fully qualified with the database name; for example, alerts.status.

The *condition* consists of one or more expressions that return a subset of rows of the table. Where applicable, you must specify fully-qualified table names within the WHERE clause and any SELECT statements in the condition. Use the format *database_name.table_name* for a fully-qualified table name.

A restriction filter is always persistent, and is recreated when the ObjectServer restarts.

Example

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

Tip: You can also create restriction filters in the Filter Builder.

Related concepts

“Naming conventions for ObjectServer objects” on page 139

“Conditions” on page 172

Dropping a restriction filter

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

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

Syntax

```
DROP RESTRICTION FILTER filter_name;
```

Example

```
drop restriction filter myfilter;
```

Files

ObjectServer files are user-defined storage objects for log or report data.

An ObjectServer file is a logical file that 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.

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 there is a possibility that an ObjectServer file already exists 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 ObjectServer naming conventions.

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 (for example: c:\\tmp\\testfile.txt) or use the equivalent UNIX path (for example: c:/tmp/testfile.txt).

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.

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 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 KB. The maximum size is 4 GB.

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

Example

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

If you run this example command, the following sequence of files are created and used:

1. 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 of 20 KB.
3. The ObjectServer renames logfile1 to logfile2. It then creates a new logfile1 and writes to this file 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 this file 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 this file until it exceeds the maximum size.

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

Related concepts

“Naming conventions for ObjectServer objects” on page 139

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 you want to stop logging temporarily, but do not want to discard the file you have configured.

Example

```
alter file logit truncate;
```

Related reference

“Logging information to ObjectServer files (WRITE INTO command)” on page 179

Dropping a file

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

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

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.

Example

```
drop file logit;
```

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 32. ObjectServer reserved words

ACTCMD	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

Table 32. ObjectServer reserved words (continued)

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	EVTFT	EXECUTABLE
EXEC[UTE]	EXTENSION	EXTERNAL	FALSE	FANP
FEB[RUARY]	FILE	FILTER	FLUSH	FOR
FORMAT	FRI[DAY]	FROM	FULL	GET
GETIDUC	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]	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	SNDMSG	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	VIRTUAL
WAIT	WED[NESDAY]	WHEN	WHERE	WIDTH
WITH	WORK	WRITE	YES	XST

SQL building blocks

Use the following building blocks to manipulate data in ObjectServer SQL commands: operators, functions, expressions, and 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*. 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, and 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

Related concepts

"Expressions" on page 171

Math and string operators

Use math operators to add, subtract, multiply, and divide numeric operands in expressions. Use string operators to manipulate character strings (VARCHAR and CHAR data types).

The following table describes the math operators supported by the ObjectServer.

Table 33. Math operators

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

The following table describes the string operator supported by the ObjectServer.

Table 34. String operator

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

Binary comparison operators

Use binary comparison operators to compare numeric and string values for equality and inequality.

The following table describes the comparison operators supported by the ObjectServer.

Table 35. Comparison operators

Operator	Description	Example
=	Tests for equality.	SELECT * FROM london.status WHERE Severity = 3;
!= <>	Tests for inequality.	SELECT * FROM london.status WHERE Severity <> 1;
< > <= >=	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.	SELECT * FROM london.status WHERE Severity > 5;
%= %!= %<>	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.	SELECT * FROM london.status WHERE Location %= 'New York';
%< %> %<= %>=	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 that 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.	SELECT * FROM london.status WHERE site_code %< 'UK3';

Table 35. Comparison operators (continued)

Operator	Description	Example
[NOT] LIKE	<p>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 syntax described in the <i>IBM Tivoli Netcool/OMNIBus User's Guide</i>.</p> <p>The NOT keyword inverts the result of the comparison.</p>	<pre>SELECT * FROM london.status WHERE Summary LIKE 'down';</pre> <p>The result is all rows in which Summary contains the substring down.</p>

The LIKE and NOT LIKE comparison operators allow regular expression pattern-matching in the string being compared to the column expression. Regular expressions are sequences of *atoms* that are made up of normal characters and metacharacters. An atom is a single character or a pattern of one or more characters in parentheses. Normal characters include uppercase and lowercase letters, and numbers. Metacharacters are non-alphabetic characters that possess special meanings in regular expressions. The ObjectServer supports two types of regular expression libraries:

- NETCOOL: Use this default library for single-byte character processing.
- TRE: This library enables usage of the POSIX 1003.2 extended regular expression syntax, and provides support for both single-byte and multi-byte character languages.

For further information on these libraries, as well as descriptions of the regular expression syntax formats and examples of usage, see the *IBM Tivoli Netcool/OMNIBus User's Guide*.

List comparison operators

Use list comparison operators to compare a value to a list of values.

Conditions using list comparison operators use the binary comparison operators with the logical operators (ANY, ALL, IN, or NOT IN).

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 <>ALL comparison.

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

Example

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

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

Related concepts

“Binary comparison operators” on page 161

“Logical operators”

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

You can combine comparisons using logical operators.

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 the following table.

Table 36. Truth table for NOT operator

A	NOT A
FALSE	TRUE
TRUE	FALSE

An AND expression is true only if all of its inputs are TRUE, as shown in the following table.

Table 37. 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 the following table.

Table 38. Truth table for OR operator

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

Example

The following query combines comparisons using logical operators:

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

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);
```

Bitwise operators

Use bitwise operators to form expressions that test or manipulate operands at the binary level.

The ObjectServer supports bitwise operators on the following data types: integer, unsigned, Boolean, integer64 and unsigned 64. If you attempt to perform bitwise operations on unsupported data types, an “invalid type” error is generated.

The following table describes the bitwise operators supported by the ObjectServer:

Table 39. Bitwise operators

Operator	Description	Example
&	Bitwise AND	In the following example, the result is 0: SET ValueA = 1; SET ValueB = 0; SET result = ValueA & ValueB;
	Bitwise OR	In the following example, the result is 1: SET ValueA = 1; SET ValueB = 0; SET result = ValueA ValueB;
~	Bitwise NOT	In the following example, the result is -1: SET ValueA = 0; SET result = ~ValueA;

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. The following table shows the order of precedence of all ObjectServer operators.

Table 40. Operator precedence

Highest Precedence
Unary + - ~
Math * /
Binary + - &
Comparison operators (including list comparisons)
NOT
AND
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.

The following table describes the functions supported by the ObjectServer.

Table 41. ObjectServer functions

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.
<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

Table 41. ObjectServer functions (continued)

Function	Description	Example
<code>dayasnum(<i>time</i>)</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(<i>time</i>)</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(<i>time</i>)</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(<i>time</i>)</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).	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(<i>string</i>)</code>	Returns the value of the specified environment variable as a string.	<code>getenv('NCHOME')</code> returns a directory name, for example, <code>/opt/netcool</code> .
<code>get_prop_value(<i>string</i>)</code>	Returns the value of the specified ObjectServer property as a string.	<code>get_prop_value('Name')</code> returns the ObjectServer name, for example, <code>NCOMS</code> .
<code>getservername()</code>	Takes no arguments and returns the name of the ObjectServer as a string.	<code>select LastOccurrence from alerts.status where ServerName = getservername();</code>
<code>hourofday(<i>time</i>)</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.	<code>select hourofday>LastOccurrence) from mytab;</code>
<code>instance_of(<i>class</i>, <i>parent_class</i>)</code>	Returns TRUE if <i>class</i> is a subclass of <i>parent_class</i> or if they are equal, using the hierarchy defined in the <code>master.class_membership</code> table. Otherwise returns FALSE. The variables <i>class</i> and <i>parent_class</i> can both be either string or integer.	<code>select Node, Summary, AlertGroup, Server from alerts.status where instance_of(Class, 'DB2') = true;</code>
<code>is_env_set(<i>string</i>)</code>	Returns 1 if the specified environment variable is set; 0 otherwise.	When the <code>NCHOME</code> environment variable is set, <code>is_env_set('NCHOME')</code> returns 1.
<code>log_2(<i>real</i>)</code>	Takes a positive real argument and returns the logarithm to base 2.	<code>log_2(4.0)</code> returns 2.000000
<code>lower(<i>string</i>)</code>	Converts a character string argument into lowercase characters.	<code>lower('LIMA')</code> returns <code>lima</code>
<code>ltrim(<i>string</i>)</code>	Removes whitespace from the left of a string.	<code>ltrim(' tree')</code> returns <code>tree</code> .

Table 41. ObjectServer functions (continued)

Function	Description	Example
<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.	<code>select minuteofhour>LastOccurrence) from mytab;</code>
<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.	<code>select monthasnum>LastOccurrence) from mytab;</code> 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.	<code>select monthname>LastOccurrence) from mytab;</code> 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.	<code>select monthofyear>LastOccurrence) from mytab;</code> Unlike <code>monthasnum</code> , January is 1, February is 2, and so on.
<code>nvp_exists(string_nameval_pairs, string_name)</code>	Verifies that a name-value pair exists. Used with extended attributes.	<code>nvp_exists(ExtendedAttr, 'Region')</code> Returns TRUE if Region exists in the extended attributes as the name of a name-value pair. If the name does not exist, the function will return FALSE.
<code>string nvp_get(string name_value_pairs, string key)</code>	Retrieves the value of <i>name</i> in a name-value pair. If the name exists in the attribute, the function returns the value. If <i>name_value_pairs</i> is not valid, the empty string is returned and an error logged. If the <i>key</i> is not present, the empty string is returned (""). If there are multiple entries for the name, the first one is returned.	<code>nvp_get(ExtendedAttr, 'Region')</code> Returns the Region attribute.

Table 41. ObjectServer functions (continued)

Function	Description	Example
<code>string</code> <code>nvp_set(string</code> <code>name_value_pairs,</code> <code>string key,</code> <code>any value)</code>	<p>Replaces or adds a name-value pair to a name-value pair string. Returns a new name-value pair string with the new name-value added or replaced.</p> <p>Adds or replaces keys from a name-value pair string and returns the new name-value pair string. A date is stored in seconds since 1970; that is, the UNIX epoch format.</p> <p>If <code>name_value_pairs</code> is not a valid string (that is, it is not an empty string or contains entries that do not conform to the correct format), then <code>\$key="\$value"</code> is returned and an error logged.</p> <p>If there are multiple entries for the key, only the first is replaced.</p>	<p><code>ExtendedAttr = nvp_set(ExtendedAttr, 'Region', 'EMEA');</code></p> <p>Sets the Region attribute in the extended attributes.</p>
<code>power(real1, real2)</code>	Takes two real arguments and returns <code>real1</code> raised to the power of <code>real2</code> .	<code>power(2.0, 3.0)</code> returns 8.000000
<code>rtrim(string)</code>	Removes whitespace from the right of a string.	<code>rtrim('tree ')</code> returns tree.
<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.	<code>select secondofminute>LastOccurrence) from mytab;</code>
<code>split_multibyte(</code> <code>string_message,</code> <code>int_chunk_no,</code> <code>int_chunk_len)</code>	<p>Returns the complete multi-byte string <code>strout</code> of at most <code>int_chunk_len</code> bytes, starting from byte <code>(int_chunk_no -1) * int_chunk_len</code> of <code>string</code> message.</p> <p>If the split will cause a multi-byte character to be incomplete in the target string, the function returns the largest complete string it can. The next call to the function (providing the <code>int_chunk_len</code> is the same at the previous call) will start from the character that could not be completely extracted.</p> <p>The function will split a multi-byte string into smaller strings that hold only complete multi-byte characters. This is primarily meant for storing large strings into several smaller database fields.</p> <p>The <code>int_chunk_len</code> in all calls to the <code>split</code> function on the same string must be the same.</p>	See the example that follows this table.

Table 41. ObjectServer functions (continued)

Function	Description	Example
<code>substr(string_message, int_startpos, int_len)</code>	Extracts a substring, starting at the position specified in the second parameter, for the number of characters specified by the third parameter. The string is indexed from 1.	<code>substr('abcdefg', 2, 3)</code> returns bcd, starting at the second character, returning the next 3.
<code>to_char(argument [, 'conversion_spec'])</code>	<p>Converts the argument to a string. The argument can be of any data type except a string.</p> <p>The comma (,) is required only if a conversion string is specified.</p> <p>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 defined in reference date/time format section. The default format is EEE MMM dd HH:mm:ss yyyy or in POSIX format %a %b %d %T %Y. Note that the POSIX format with % is deprecated.</p>	<p><code>to_char(73)</code> returns 73</p> <p><code>to_char(FirstOccurrence)</code> returns a string such as Thu Dec 11 16:02:05 2003</p> <p><code>to_char>LastOccurrence, '%Y')</code> returns a string such as 2010</p>
<code>to_int('argument')</code>	<p>Converts the argument to an integer. The argument can be of any data type except integer.</p> <p>This function strips any leading white space from the argument, and then scans the remaining string. The scan stops when it encounters a character that cannot be converted to a decimal character, or when it reaches the end of the string, whichever happens first. When the scan stops, the function converts the characters to their decimal value, or returns 0 if it failed to encounter any characters that could be converted to decimal.</p>	<p><code>to_int('73')</code> returns 73</p> <p><code>to_int('3F')</code> returns 3</p> <p><code>to_int('UK')</code> returns 0</p> <p><code>to_int('F3')</code> returns 0</p>
<code>to_real('argument')</code>	<p>Converts the argument to a 64-bit real number. The argument can be of any data type except real.</p> <p>This function strips any leading white space from the argument, and then scans the remaining string. The scan stops when it encounters a character that cannot be converted to a decimal character, or when it reaches the end of the string, whichever happens first. When the scan stops, the function converts the characters to their decimal value, or returns 0 if it failed to encounter any characters that could be converted to decimal.</p>	<p><code>to_real('7.3')</code> returns 7.300000</p> <p><code>to_real('3F')</code> returns 3.000000</p> <p><code>to_real('UK')</code> returns 0</p> <p><code>to_real('F3')</code> returns 0</p>

Table 41. ObjectServer functions (continued)

Function	Description	Example
<code>to_time(argument [, 'conversion_spec'])</code> <code>to_date(argument [, 'conversion_spec'])</code>	<p>Converts the argument to a time type. The argument can be of any data type except a time type.</p> <p>The comma (,) is required only if a conversion string is specified.</p> <p>If the argument is a string type, you can specify a second argument consisting of a conversion specification to format the output. The default format is EEE MMM dd HH:mm:ss yyyy or in POSIX format %a %b %d %T %Y. Note that the POSIX format with % is deprecated.</p>	<code>update mytab set my_utc_col = to_time('Thu Dec 11 16:00:00 2003')</code>
<code>to_unsigned(argument)</code>	<p>Converts the argument to a 64-bit unsigned integer. The argument can be of any data type except a 64-bit unsigned integer.</p> <p>This function strips any leading white space from the argument, and then scans the remaining string. The scan stops when it encounters a character that cannot be converted to a decimal character, or when it reaches the end of the string, whichever happens first. When the scan stops, the function converts the characters to their decimal value, or returns 0 if it failed to encounter any characters that could be converted to decimal.</p>	<code>to_unsigned('73')</code> returns 73 <code>to_unsigned(73)</code> returns 73 <code>to_unsigned('UK')</code> returns 0 <code>to_unsigned('F3')</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>

Example: Usage of split_multi-byte

```

for each row res_filter in catalog.restrictions where
  res_filter.RestrictionName = rf_users.RestrictionName
begin
  -- Populate master.profile with the new row.
  -- Cut up the filter text into 255 byte chunks
  update master.profiles set HasRestriction = 1,
    Restrict1 = split_multibyte( res_filter.ConditionText, 1, 255),
    Restrict2 = split_multibyte( res_filter.ConditionText, 2, 255),
    Restrict3 = split_multibyte( res_filter.ConditionText, 3, 255),
    Restrict4 = split_multibyte( res_filter.ConditionText, 4, 255)
  Where UID = rf_users.GranteeID;
end;

```

In this example:

Restrict1 is assigned at most 255 bytes from res.filter.ConditionText from byte 1

Restrict2 is assigned at most 255 bytes from res.filter.ConditionText from byte (2-1)
* 255

Restrict1 is assigned at most 255 bytes from res.filter.ConditionText from byte (3-1)
* 255

Restrict1 is assigned at most 255 bytes from res.filter.ConditionText from byte (4-1)
* 255

Related reference

“alerts.status table” on page 321

“master.class_membership table” on page 336

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 entities:

- A quoted string ('Node XB1')
- A number (9)
- A column name (Severity)
- An ObjectServer property (ServerName)
- An environment variable (NCHOME)
- 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 data type constraints. For example, the expression 5 * 'Node XB1' is not valid because you cannot multiply an integer and a string.

Related reference

“Specifying data types for columns” on page 145

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:

- Restriction filters
- The WHERE clause of the SELECT, UPDATE, and DELETE 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)`).

Valid conditions are shown in the following list:

```
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);
```

Related concepts

“Binary comparison operators” on page 161

“List comparison operators” on page 162

“Logical operators” on page 163

“Expressions” on page 171

Related reference

“Creating database triggers (CREATE TRIGGER command)” on page 208

Querying and manipulating data using ObjectServer SQL

You can use data manipulation language (DML) commands to query and modify data in existing tables, views, and files.

ObjectServer SQL provides the following commands for manipulating data.

Table 42. ObjectServer objects and associated DML commands

ObjectServer object	Allowed DML commands
TABLE	SELECT INSERT UPDATE DELETE DESCRIBE SVC
VIEW	SELECT DESCRIBE SVC
FILE	WRITE INTO

Tip: Restriction filters are automatically applied in SELECT, INSERT, UPDATE, and DELETE commands.

Inserting a new row of data into a table (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);
```

Updating the data in table columns (UPDATE command)

Use the UPDATE command to update one or more columns in an existing row of data in a table.

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 is set to TRUE. When the NOMODIFY property is set to TRUE, the value of a column cannot be changed after the initial INSERT command.

If you are updating a single row and you know the value of the primary key for the row that you want 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 quotation marks.

If you include a WHERE clause, only rows meeting the criteria specified in the *condition* are updated. 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';
```

Related concepts

“Conditions” on page 172

Related reference

“Inserting a new row of data into a table (INSERT command)” on page 173

Deleting rows of data from a table (DELETE command)

Use the DELETE command to delete one or more rows of data from an existing table.

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 that you want 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, you must specify the values in order, separated by commas and in parentheses. If the column has a string value, it must be enclosed in quotation marks.

If you include a WHERE clause, only rows meeting the criteria specified in the *condition* are deleted. 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' ;
```

Related concepts

“Conditions” on page 172

Retrieving data from a table or view (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.

You can use the SELECT command to perform the following actions:

- Retrieve data that matches a specified criteria (scalar SELECT)
- Return a single value that is based on a calculation on a number of rows (aggregate SELECT)
- Group all rows that contain identical values in one or more columns, and perform aggregate functions on the columns (group by SELECT)

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 must also include an ORDER BY clause to order (or sort) the selected rows.

Use an asterisk (*) to retrieve all non-hidden columns in the table. Otherwise, you can either specify a comma-separated list of columns that you want to retrieve, or create *virtual columns* using:

- Simple expressions (for example, Severity)
- Complex expressions that contain math or string operators (for example, Severity + Tally)
- Functions (for example, getdate - 60)

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 satisfying the criteria specified in the *condition* are returned.

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 by using the AS keyword, use that alias in any references in the ORDER BY column list rather than the corresponding column name.

Examples

The following example selects all rows of the alerts.status table where the Severity is equal to 4:

```
select * from alerts.status where Severity = 4;
```

The following example selects all rows of the alerts.status table where the Node contains the string terminal followed by any other characters:

```
select * from alerts.status where Node like 'terminal.*';
```

In the preceding example, regular expression syntax is used in the LIKE comparison. For information on regular expression syntax used in the LIKE comparison, see the appendix on regular expressions in the *IBM Tivoli Netcool/OMNIBus User's Guide*.

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 that the virtual column Severity + Tally is renamed Real_Severity.

```
select Severity, Severity + Tally as Real_Severity from alerts.status;
```

Related concepts

“Conditions” on page 172

“Expressions” on page 171

“Functions” on page 165

“Math and string operators” on page 160

Aggregate SELECT

An aggregate SELECT command 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 following aggregate functions (depicted by *aggr_expression* in the syntax) are supported.

Table 43. Aggregate functions

Function	Result returned
<code>max(<i>scalar_column_expr</i>)</code>	This returns the maximum numeric value for the column expression from the rows that satisfy the SELECT condition.
<code>min(<i>scalar_column_expr</i>)</code>	This returns the minimum numeric value for the column expression from the rows that satisfy the SELECT condition.
<code>avg(<i>scalar_column_expr</i>)</code>	This returns the average numeric value for the column expression from the rows that satisfy the SELECT condition.
<code>sum(<i>scalar_column_expr</i>)</code>	This returns the sum (total) of the numeric values for the column expression from the rows that meet the SELECT condition.
<code>count(<i>scalar_column_expr</i>)</code> <code>count(*)</code>	This returns the total number of rows that satisfy the SELECT condition.
<code>dist(<i>scalar_column_expr</i>, <i>value</i>)</code>	This returns the total number of rows for which the column equals the specified value. The result of: <code>dist(<i>scalar_column_expr</i>, <i>value</i>)</code> is equivalent to: <code>SELECT count(<i>scalar_column_expr</i>) FROM <i>table_name</i> WHERE <i>scalar_column_expr</i> = <i>value</i>;</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 satisfying the criteria specified in the *condition* are returned.

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 perform comparisons by 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;
```

Related concepts

“Conditions” on page 172

“Functions” on page 165

Group by SELECT

You can use a SELECT command with a GROUP BY clause to group all rows that have identical values in a specified column or combination of columns, into a single row. You can also 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. An asterisk (*) 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 expression 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.

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.

Related concepts

“Conditions” on page 172

Related reference

“Basic (scalar) SELECT” on page 175

“Aggregate SELECT” on page 177

Logging information to ObjectServer files (WRITE INTO command)

Use the WRITE INTO command to write logging information to ObjectServer files. An ObjectServer file is a logical file, which has a corresponding file or set of files on the physical file system.

Syntax

```
WRITE INTO file_name [ VALUES ] (expression, ...);
```

A carriage return follows each message.

Example

The following command adds a message to the physical file associated with the ObjectServer file file1 each time a user connects to a database.

```
WRITE INTO file1 VALUES  
('User', %user.user_name, 'connected to the system at', getdate );
```

The %user.user_name user variable used in this example is only available in procedures and triggers.

Related concepts

“Files” on page 156

Related reference

“Implicit user variables in procedures and triggers” on page 202

Displaying details of columns in a table or view (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 for this command includes the column name, the data type (returned as the ObjectServer ID), 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

Use the following command to display information about the columns in the catalog.tables table:

```
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

Related reference

“Specifying data types for columns” on page 145

Adding or updating service status data (SVC command)

Use the SVC command to add or update the state of a service status alert in the service.status table for IBM Tivoli Composite Application Manager for Internet Service Monitoring.

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 the following table. If you enter any other value, the service level is set to 3 (unknown).

Table 44. Service status levels

Integer	Service status level
0	Good.
1	Marginal.
2	Bad.

Table 44. Service status levels (continued)

Integer	Service status level
3	Service level is unknown.

Example

```
svc update 'newservice' 2;
```

Sending IDUC notifications to IDUC clients (IDUC FLUSH command)

Use the IDUC FLUSH command to send IDUC notifications to IDUC clients.

Syntax

IDUC FLUSH *destination*

In this command:

- *destination* = *spid*
- *spid* = *integer_expression* (The literal client connection ID)

Example

```
create or replace trigger exmple_trigger
group default_triggers
enabled true
priority 2
on signal iduc_data_fetch
begin
  for each row conn in iduc_system.iduc_stats
  begin
    IDUC FLUSH conn.connectionid;
  end;
end;
go
```

Changing the default and current settings of the ObjectServer (ALTER SYSTEM command)

Use the ALTER SYSTEM command to change the default and current settings of the ObjectServer.

You can use this command to perform the following actions:

- Shut down the ObjectServer
- Set one or more ObjectServer properties
- Drop one or more user connections
- Back up the ObjectServer

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 quotation marks 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.

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 the path to an existing directory where you want to back up the files. This value must be in quotation marks.

Note: The directory cannot be the one in which ObjectServer data files are stored, which is \$NCHOME/omnibus/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 computer with the same operating system as the computer on which they were created.

Examples

```
alter system set 'Auto.StatsInterval' = 15 set 'AlertSecurityModel' = 1;
```

```
alter system shutdown;
```

Related concepts

“Checkpoint file creation” on page 23

Related reference

“ObjectServer properties and command-line options” on page 3

Setting the default database (SET DATABASE and USE DATABASE commands)

Use the SET DATABASE or USE DATABASE command to set a database as the default for an SQL interactive interface **nco_sql** session. These two commands perform the same function.

Restriction: You cannot use this command in triggers or procedures.

After you set the default database with the SET DATABASE or USE DATABASE command, 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;
```

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 your SQL syntax (CHECK STATEMENT command)

The CHECK STATEMENT command parses and checks the syntax of the SQL commands enclosed in quotation marks and returns either a success message or a description of any errors.

Syntax

```
CHECK STATEMENT 'command; command; ...';
```

Because the CHECK STATEMENT command does not run 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 run.

Creating, modifying, and deleting users, groups, and roles

You can use SQL commands to organize collections of users into *groups* and then assign *roles* to each group to control access to ObjectServer objects. You can create, modify, and drop users, groups, and roles.

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 Tivoli 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 triggers, trigger groups, files, SQL procedures, external procedures, and signals. You can then grant that role to a group of administrators who will be creating and updating triggers.

Default groups and roles for network management operators and administrators are defined in the security.sql SQL script. You can also use this script as a

template to create your own groups and roles.

Creating a user (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 a unique user name for the user being added. This name can be up to 64 characters in length. If the user is to be externally authenticated, for example, in a Lightweight Directory Access Protocol (LDAP) repository or by using Pluggable Authentication Modules (PAM), specify the user name that is stored in the external authentication repository.

Note: User names are case-sensitive, and must be enclosed in quotation marks. 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 2147483647.

The *full_user_name* is a text string containing the full name of 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. No password is required for an externally-authenticated user.

To specify that the user is externally authenticated, set PAM to TRUE. The **Sec.ExternalAuthentication** ObjectServer property must also be set to either PAM or LDAP, as appropriate for your authentication system. If PAM is set to FALSE or **Sec.ExternalAuthentication** is set to none, the user cannot be authenticated externally. If you want to store the user name and associated password in the ObjectServer, and to perform ObjectServer authentication, set PAM to FALSE. For more information about PAM or LDAP, see the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide*.

Example

```
create user 'joe' id 1 full name 'Joseph R. User';
```

Related reference

“ObjectServer properties and command-line options” on page 3

Modifying the details of an existing user (ALTER USER command)

Use the ALTER USER command to change the settings, such as the password, for the specified user. You can change more than one setting in a single ALTER USER command.

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 unique user name for the user being modified. This name cannot be changed.

Use the PASSWORD setting to change the password for the specified user. Note that you cannot change the password of a user that is externally authenticated in an LDAP system. You can change the password of a user that is externally authenticated in a PAM system only if the external PAM system has been configured to allow this. If allowed to change the password of a PAM-authenticated user, you must also use the AUTHORIZE PASSWORD keywords to specify the old password.

Use the ENABLED setting to activate (TRUE) or deactivate (FALSE) the specified user. An activated user has login access to the system.

Set PAM to TRUE to enable the user to be externally authenticated. The **Sec.ExternalAuthentication** ObjectServer property must also be set to either PAM or LDAP, as appropriate for your authentication system. If PAM is set to FALSE or **Sec.ExternalAuthentication** is set to none, the user cannot be authenticated externally. If you want to perform ObjectServer authentication, set PAM to FALSE. For more information about PAM or LDAP, see the *IBM Tivoli 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.

Example

```
alter user 'joe' set password 'topsecret';
```

Related concepts

“Restriction filters” on page 155

Related reference

“ObjectServer properties and command-line options” on page 3

Deleting a user (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 unique user name for the user being dropped.

Example

```
drop user 'joe';
```

Creating a group (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 a unique name for the group being created. This name can be up to 64 characters in length.

Note: Group names are case-sensitive, and must be enclosed in quotation marks. Any leading or trailing white space 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 7 are reserved for system groups. Identifiers for other groups can be set to any value between 8 and 2147483647.

Use the optional COMMENT setting to add a description of the group you are creating.

Use the MEMBERS keyword to specify the user names of one or more users that you want to add as group members.

Example

```
create group 'AutoAdmin' id 3 COMMENT 'Group to manage Automations'  
members 'joe', 'bob';
```

Modifying the details of an existing group (ALTER GROUP command)

Use the ALTER GROUP command to change user settings for the specified group. You can change more than one setting in a single ALTER GROUP command.

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 unique name of the group being modified. You cannot change this name.

Use the SET COMMENT setting to modify the description of the group.

Use the ASSIGN or REMOVE RESTRICTION FILTER setting to assign or remove restriction filters that apply to the group. Only one restriction filter per table can be assigned to a group.

Use the ASSIGN or REMOVE MEMBERS setting to assign users as group members, or remove users from the group.

Example

```
alter group 'AutoAdmin' assign members 'sue';
```

Related concepts

“Restriction filters” on page 155

Deleting a group (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 unique name of the group 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.

Example

```
drop group 'LondonAdmin';
```

Creating a role (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 unique name of the role being created. This name can be up to 64 characters in length.

Note: Role names are case-sensitive, and must be enclosed in quotation marks. 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 13 and 2147483647.

Use the optional COMMENT setting to add a description of the role you are creating.

Default roles are created by the security.sql script.

Example

```
create role 'SuperAdmin' id 500
comment 'only users with root access should be granted this role';
```

Related concepts

“Using roles to assign permissions to users”

Modifying the description of a role (ALTER ROLE command)

Use the ALTER ROLE command to modify the description of an existing role.

Syntax

```
ALTER ROLE 'role_name' SET COMMENT 'comment_string';
```

The *role_name* is a text string containing the unique name of the role being modified. This name cannot be changed.

Use the COMMENT setting to modify the description of the role.

Example

```
alter role 'SuperAdmin' set comment 'enhanced description of role';
```

Using roles to assign permissions to users

After you create a role, you must assign permissions to the role using the GRANT command. You can then use the GRANT ROLE command to assign the role to one or more groups. All users who are group members are automatically assigned the permissions defined for that role.

Assigning permissions to roles (GRANT command)

Use the GRANT command to assign system and object permissions to roles. *System permissions* control the commands that can be run in the ObjectServer. *Object permissions* control access to individual objects, such as tables.

Syntax for granting system permissions

```
GRANT system_permission,...
TO ROLE 'role_name',...
[ WITH GRANT OPTION ];
```

The value of *system_permission* can be any of the following subcommands:

```
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

```

The *role_name* is a text string containing the unique name of the role or roles to which you are assigning permissions.

The WITH GRANT OPTION option enables the roles to whom the permission is granted to grant the permission to other roles.

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 for granting system permissions

```
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 ];

```

You can assign one or more permissions to ObjectServer objects. Use *object_permission* to define the SQL commands that authorized users can run on an ObjectServer object of type *permission_object*. The *object_name* is a text string containing the unique name of the object.

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. The following table lists the permissions that the owner has for each object type. The owner can also grant these permissions to other users.

Table 45. Objects and associated permissions

Objects (<i>permission_object</i>)	Permissions (<i>object_permission</i>)
DATABASE	DROP
	CREATE TABLE
	CREATE VIEW

Table 45. Objects and associated permissions (continued)

Objects (<i>permission_object</i>)	Permissions (<i>object_permission</i>)
TABLE	DROP ALTER SELECT INSERT UPDATE DELETE CREATE INDEX DROP INDEX
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 *role_name* is a text string containing the unique name of the role or roles to which the permissions are being assigned.

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 by using the CREATE OR REPLACE syntax, you need ALTER permission to replace an existing object. Some objects can be altered only by 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.

Example for granting object permissions

```
grant drop on database testdb to role 'DDL_Admin';
```

Related reference

“Revoking permissions from roles (REVOKE command)” on page 192

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.

The following table lists the parent of each ObjectServer object.

Table 46. Inheritance of object permissions

Parent object	Child objects
System	DATABASE TRIGGER GROUP FILE SQL PROCEDURE EXTERNAL PROCEDURE SIGNAL RESTRICTION FILTER
DATABASE	TABLE VIEW
TABLE	INDEX
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.

Assigning roles to groups (GRANT ROLE command)

After you have created roles as collections of permissions, you can assign the roles to groups and revoke the roles from groups. Use the GRANT ROLE command to assign roles to groups.

Role assignments 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 unique name of a role being assigned.

Each *group_name* is the name of a group to which the role or roles are being assigned.

Example

```
grant role 'AutoAdmin' to group 'LondonAdministrators';
```

Related reference

“Revoking roles from groups (REVOKE ROLE command)” on page 194

Revoking permissions from roles (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',... ;
```

The following list shows each *system_permission* that can be revoked from a role:

```
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
```

Each *role_name* is a text string containing the unique name of a role from which the permission is being revoked.

Example for revoking system permissions

```
revoke create table from role 'DDL_Admin';
```

Syntax for revoking object permissions

```
REVOKE object_permission,...  
ON permission_object object_name  
FROM ROLE 'role_name',... ;
```

You can revoke one or more permissions for ObjectServer objects. Use *object_permission* to specify the SQL commands that you want to revoke for an ObjectServer object of type *permission_object*. The *object_name* is a text string containing the unique name of the object.

Details of each object and associated permissions that you can revoke are shown in the following table.

Table 47. Objects and associated permissions

Objects (<i>permission_object</i>)	Permissions (<i>object_permission</i>)
DATABASE	DROP CREATE TABLE CREATE VIEW
TABLE	DROP ALTER SELECT INSERT UPDATE DELETE CREATE INDEX DROP INDEX
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

Each *role_name* is a text string containing the unique name of a role from which the permission is being revoked.

Note: The REVOKE command 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, the LondonAdministrators role retains the CREATE TABLE permission.

Example for revoking object permissions

```
revoke drop on database testdb from role 'DDL_Admin';
```

Related reference

“Assigning permissions to roles (GRANT command)” on page 188

Revoking roles from groups (REVOKE ROLE command)

Use the REVOKE ROLE command to revoke roles from groups.

Syntax

```
REVOKE ROLE 'role_name',...  
FROM GROUP 'group_name',... ;
```

Each *role_name* is a text string containing the unique name of a role being revoked.

Each *group_name* is the name of a group that will no longer be assigned 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 SuperAdmin group after SuperAdmin has granted the AutoAdmin role to the LondonAdministrators group, the LondonAdministrators group still has the AutoAdmin role.

Example

```
revoke role 'AutoAdmin' from group 'LondonAdministrators';
```

Related reference

“Assigning roles to groups (GRANT ROLE command)” on page 191

Deleting a role (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 unique name of the role being dropped. When you drop a role, all of the related role permissions are dropped.

Example

```
drop role 'AutoAdmin';
```

Creating, running, and dropping procedures

A procedure is an executable SQL object that can be called to perform common operations.

The types of procedures that you can create are:

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

After you create a procedure in the ObjectServer, you can run the procedure from the SQL interactive interface (**nco_sql1**). You can also run the procedure in a trigger by using the EXECUTE PROCEDURE command.

SQL procedures

An SQL procedure is a set of parameterized SQL commands, or code fragments, with programming language constructs that you can use 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 that 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

Components of an SQL procedure

SQL procedures have the following major components: parameters, local variable declarations, and procedure body.

Parameters are values that are passed into, or out of, a procedure. You declare the parameters of the procedure when you create the procedure, and you specify what values are passed as parameters when you run 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 run is called an *actual parameter*.

The values that you pass to the procedure must be of the same data type as in the parameter declaration.

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 tests conditions and manipulates data in the database.

Related reference

"Creating SQL procedures (CREATE PROCEDURE command)" on page 196

Creating SQL procedures (CREATE PROCEDURE command)

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 that are performed in the 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
```

If there is a possibility that a procedure already exists 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 ObjectServer naming conventions.

Following the *procedure_name*, specify the parameters that can be passed into, or out of, the procedure, within parentheses (). You must include parentheses after 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 that 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. If you do not want to change a variable value within the procedure, use an IN parameter to pass the variable value into the procedure. This parameter is used by 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, you cannot use this type of parameter 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 parameter is useful for variables that 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 ObjectServer naming conventions.

The *parameter_type* defines the type of data that the parameter can pass into, or out of, the procedure. The data type can be any valid ObjectServer data type, except VARCHAR or INCR.

An ARRAY OF *parameter_type* is an array of any valid parameter type.

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. Use semicolons to separate local variable declarations. Variable names must be unique within the procedure and comply with the ObjectServer naming conventions. The *variable_declaration* can include either of the following variable types:

- Simple variables:

variable_name variable_type

- Array variables:

*variable_name variable_type [ARRAY] [**integer**]*

A *variable_type* is any valid ObjectServer data type, except VARCHAR or INCR.

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.

The body of a procedure is enclosed within the keywords BEGIN and END. You can use the SET statement, IF THEN ELSE statement, CASE WHEN statement, FOR EACH ROW loop, and FOR loop in the procedure body.

Example

In the following procedure declaration, the formal parameter is the variable *current_severity*. When you run the procedure, you pass an actual parameter.

```
CREATE PROCEDURE calculate_average_severity
( IN current_severity INTEGER )
```

For example, in the following procedure call, the actual parameter is the value 5, which is assigned to the formal parameter *current_severity*.

```
EXECUTE PROCEDURE calculate_average_severity(5);
```

Example

```
CREATE PROCEDURE add_or_concat
( IN counter INTEGER, IN one_char_string CHAR(1))
```

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 run it. These integers are assigned to an array named *severity_arr*.

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 20 integer values used in the procedure to store node names:

```
DECLARE NodeNameArray INTEGER [20];
```

Related concepts

“Naming conventions for ObjectServer objects” on page 139

“Components of an SQL procedure” on page 195

“Expressions” on page 171

Related reference

“Specifying data types for columns” on page 145

“How to construct an SQL procedure body statement”

How to construct an SQL procedure body statement:

The body of an SQL procedure contains a set of SQL statements and programming constructs that manipulate data in the ObjectServer.

Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name ([ procedure_parameter,... ])
[ DECLARE variable_declaration;...[;] ]
BEGIN
  procedure_body_statement;...[;]
END
```

This topic describes only the entries required for the body of a procedure (*procedure_body_statement*), which is enclosed between the keywords BEGIN and END.

In the body of a procedure, you must separate each statement, except the last one, by a semicolon.

Statements in the procedure can include SQL commands and additional programming constructs.

You can run 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 run the commands in the procedure body.

Attention: 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 in the procedure body:

- SET assignment statement
- IF THEN ELSE statement

- CASE WHEN statement
- FOR EACH ROW loop
- FOR loop

Related reference

“Creating SQL procedures (CREATE PROCEDURE command)” on page 196

“Implicit user variables in procedures and triggers” on page 202

“SET statement”

“IF THEN ELSE statement”

“CASE WHEN statement” on page 200

“FOR EACH ROW loop” on page 200

“FOR loop” on page 201

SET statement:

When constructing the body of an SQL procedure, you can 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.

Note: The value returned by the expression must be of a type compatible with the variable into which you write the value.

Related concepts

“Expressions” on page 171

Related reference

“How to construct an SQL procedure body statement” on page 198

IF THEN ELSE statement:

When constructing the body of an SQL procedure, you can use the IF THEN ELSE statement to perform 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 run 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 run until the next keyword is reached. If an ELSE statement exists, and no previous conditions have been met, the statements following the ELSE statement are run until the END IF statement is reached.

Related concepts

“Conditions” on page 172

Related reference

“How to construct an SQL procedure body statement” on page 198

CASE WHEN statement:

When constructing the body of an SQL procedure, you can use the CASE WHEN statement to perform 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 run 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 run 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 statement are run until an END CASE statement is reached.

Related concepts

“Conditions” on page 172

Related reference

“How to construct an SQL procedure body statement” on page 198

FOR EACH ROW loop:

When constructing the body of an SQL procedure, you can use the FOR EACH ROW loop to perform 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 that are specified in the condition are returned.

A BREAK command exits from the current loop, and the next statement in the procedure starts to run.

A CANCEL command stops the running of a procedure.

Attention: Do not use the CANCEL command when using a desktop ObjectServer in DualWrite mode.

Example

The following example increases 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 runs, the ObjectServer reads each row of the alerts.status table and tests to see if the value in the Severity column is 3. For each row that matches this condition, the statements within the BEGIN and END are run, until all the rows are processed.

Related concepts

“Conditions” on page 172

Related reference

“How to construct an SQL procedure body statement” on page 198

FOR loop:

When constructing the body of an SQL procedure, you can use the FOR loop to perform 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, and the next statement in the procedure starts to run.

A CANCEL command stops the running of a procedure.

Attention: Do not use the CANCEL command when using a desktop ObjectServer in DualWrite mode.

Example

The following 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;

```

Related reference

“How to construct an SQL procedure body statement” on page 198

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.

Tip: You can also use the % helper button to select %user variables.

The following table lists the read-only attributes that are available in procedures and triggers.

Table 48. Implicit user variables

Variable attribute	Data type	Description
%user.user_id	INTEGER	User identifier of the connected user.
%user.user_name	STRING	Name of the connected user.
%user.app_name	STRING	Name of the connected application (such as nco_sql).
%user.host_name	STRING	Name of the connected host.
%user.connection_id	UNSIGNED	Connection identifier. See “Example: Usage of %user.counterpart_id and %user.connection_id” on page 203.
%user.counterpart_id	UNSIGNED	Counterpart connection identifier. See “Example: Usage of %user.counterpart_id and %user.connection_id” on page 203.
%user.is_auto	BOOLEAN	If TRUE, the current action was caused by the execution of an automation (such as a temporal trigger).
%user.is_gateway	BOOLEAN	If TRUE, the current action was caused by a gateway client.
%user.is_eventlist	BOOLEAN	If TRUE, the current action was caused by an event list client.
%user.description	STRING	Descriptive name of the application. Only applicable for ObjectServer gateways or probes.

Example: Usage of %user.user_name

To reference the name of the current user in the body of a procedure or trigger, use the syntax:

`%user.user_name`

Example: Usage of `%user.counterpart_id` and `%user.connection_id`

For gateways, if `%user.connection_id` refers to a gateway writer component, `%user.counterpart_id` is the skip ID of the gateway reader component. This example shows how these variables can be used in automated failover and failback, to disconnect all of the clients (except the gateway) from a backup ObjectServer when the primary ObjectServer comes up. The following code can be added in the `backup_counterpart_up` trigger:

```
For each row connected in catalog.connections where
  (connected.ConnectionID <> %user.connection_id and
   connected.ConnectionID <> %user.counterpart_id and
   connected.AppName == 'GATEWAY') or
   connected.AppName <> 'GATEWAY'
begin
alter system drop connection connected.ConnectionID;
end;
```

Related reference

Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361

External procedures

You can create external procedures to run an executable program on a local or remote system.

Creating external procedures (CREATE PROCEDURE command)

Use the CREATE PROCEDURE command to create external procedures.

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 there is a possibility that a procedure already exists 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 ObjectServer naming conventions.

After the *procedure_name*, include the parameter declaration within parentheses (), to specify the parameters that can be passed into the external procedure. You must include parentheses after the *procedure_name* even if the procedure has no parameters. Each *parameter_name* must be unique within the procedure and must comply with the ObjectServer naming conventions.

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 that the parameter can pass into the procedure. The data type can be any valid ObjectServer data type, except VARCHAR or INCR.

The *executable_name* is the path to an executable program on a local or remote file system.

Tip: On Windows, you must escape the backslash character in file paths or the path will not be interpreted correctly. You can also use the UNIX path separator when specifying paths on Windows.

The *host_name* is the name of the host on which to run the executable program for the procedure.

The *user_id* is the effective user ID under which to run the executable program.

The *group_id* is the effective group ID under which to run the executable program.

The arguments are those passed to the executable. Only spaces can be used to separate arguments. For example: cool tool is interpreted as cool tool, whereas cool'tool or cool"tool is interpreted as cooltool.

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 '$NCHOME/omnibus/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 program. You must enclose strings in quotation marks, and escape the quotation marks with backslashes. All quotation marks in this example are single quotation marks.

Note: To run an external procedure, you must have a process control agent daemon (**nco_pad**) running on the host where the executable command is stored.

Related concepts

“Naming conventions for ObjectServer objects” on page 139

Chapter 7, “Using process control to manage processes and external procedures,” on page 249

Related tasks

“Specifying paths in the SQL interactive interface” on page 140

Related reference

“Specifying data types for columns” on page 145

Running procedures

After you create a procedure, you must run it using the EXECUTE PROCEDURE command for the actions in the procedure to occur. You can do this using the SQL interactive interface (**nco_sql**) or in a trigger or procedure.

Syntax

```
{ EXECUTE | CALL } [ PROCEDURE ] procedure_name  
[ ( expression,... ) | ( [ expression, expression,... ] ,... ) ];
```

Use *procedure_name* to specify the procedure to run.

Each of the expressions passed as actual parameters must resolve to an assignable value that matches the type of the parameter specified when the procedure was created.

Note: If you are passing an array parameter, the square brackets around the expression list, shown in bold type in the preceding syntax description, are not optional.

Example

To run the procedure described in “Creating external procedures (CREATE PROCEDURE command)” on page 203, use the following call in a trigger:

```
execute send_email( critical.Node, critical.Severity, 'Netcool E-mail',  
'root@localhost', critical.Summary, 'localhost');
```

Related concepts

“Components of an SQL procedure” on page 195

Related reference

“Creating external procedures (CREATE PROCEDURE command)” on page 203

Dropping procedures

Use the DROP PROCEDURE command to drop an existing procedure.

You cannot drop a procedure if it is referenced by other objects, such as triggers.

Syntax

```
DROP PROCEDURE procedure_name;
```

Example

```
drop procedure testproc;
```

Configuring automation using triggers

You can use automation to detect changes in the ObjectServer and run automated responses to these changes. This enables the ObjectServer to process alerts without requiring an operator to take action. You can also use automation to manage deduplication, which reduces the quantity of data held in the ObjectServer by eliminating duplicate events.

A set of standard automations is included with Tivoli Netcool/OMNIBus. These automations are created during database initialization.

Creating, modifying, and deleting trigger groups

Every trigger belongs to a trigger group, which is a collection of related triggers.

Creating a trigger group (CREATE TRIGGER GROUP command)

Use the CREATE TRIGGER GROUP command to create a new trigger group.

When you create a trigger, you must assign it to a trigger group.

Syntax

```
CREATE TRIGGER GROUP trigger_group_name;
```

The *trigger_group_name* must be unique within the ObjectServer and comply with the ObjectServer naming conventions.

Example

```
create trigger group update_database_triggers;
```

Related concepts

“Naming conventions for ObjectServer objects” on page 139

Modifying a trigger group (ALTER TRIGGER GROUP command)

Use the ALTER TRIGGER GROUP command to enable or disable an existing trigger group.

Syntax

```
ALTER TRIGGER GROUP trigger_group_name | expression  
SET ENABLED { TRUE | FALSE };
```

A trigger group is enabled by default.

You can specify a trigger group name or an expression with this command. If it is an expression, the name is not evaluated until run time.

Examples

This example disables the update_database_triggers trigger group.

```
alter trigger group update_database_triggers set enabled false;
```

This example disables all triggers except gateway triggers (belonging to the gateway_triggers trigger group) by individually listing the names of all the other trigger groups to be disabled.

```
Create trigger disable_triggers  
Group gateway_triggers  
Priority 1  
on signal gw_counterpart_up  
begin  
alter trigger group trigger_group_name_1 set enabled false;  
...  
alter trigger group trigger_group_name_n set enabled false;  
end;
```

This example uses an expression to disable all triggers except gateway triggers that belong to the gateway_triggers trigger group. At run time, the FOR EACH ROW loop is used to perform the actions in the expression, on each row in the catalog.triggers table.

```
Create trigger disable_triggers  
Group gateway_triggers  
Priority 1
```

```

on signal gw_counterpart_up
begin
for each row tg in catalog.triggers where
  tg.GroupName <> 'gateway_triggers'
begin
  alter trigger group tg.GroupName set enabled false;
end;
end;

```

Deleting a trigger group (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;
```

Creating, modifying, and dropping triggers

You can use database triggers, temporal triggers, and signal triggers in automations.

Database triggers 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: You can create your own deduplication trigger to cause a different action to occur.

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.

Signal triggers fire when a predefined system signal is raised or when a user-defined signal is raised using the RAISE SIGNAL command.

For example, you can send an e-mail to an operator when the ObjectServer starts or stops because system signals are generated.

You do not have to do anything to create or configure system signals. You must explicitly create, raise, and drop user-defined signals.

Creating database triggers (CREATE TRIGGER command)

Use the CREATE TRIGGER command to 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).

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 ]
BEGIN
    trigger_action
END;
```

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

The *trigger_name* value must be unique within the ObjectServer and comply with the ObjectServer naming conventions.

The *group_name* value can be any trigger group already created by 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.

The PRIORITY of a trigger 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.

Use the optional COMMENT keyword to add a comment (*comment_string*) for the trigger.

The BEFORE or AFTER timing keyword specifies whether the trigger runs before or after the database modification that caused the trigger to fire occurs. For example, you can create a BEFORE trigger that evaluates the name of the user before a row in the alerts.status table 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.

A database trigger fires at one of the following levels:

- FOR EACH ROW (known as a *row-level trigger*): Row-level triggers fire once for each row returned as a result of the database modification.
- FOR EACH STATEMENT (known as a *statement-level trigger*): Statement-level triggers fire once for each database modification.

Note: Only row-level triggers can be defined to fire on inserts and reinserts.

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.

Use the optional WHEN clause to test for a particular *condition* before the trigger action runs. If the condition is not met, the trigger action does not run.

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. However, trigger variables are static, so they maintain their value between the times when the trigger runs.

Example

A database signal is raised as a result of the following SQL statement:

```
DELETE FROM alerts.status WHERE Severity = 5;
```

When this statement runs, 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.

Related concepts

“Naming conventions for ObjectServer objects” on page 139

“Conditions” on page 172

Related reference

“Running commands in trigger actions” on page 215

“Best practices for creating triggers” on page 317

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.

The following table shows when the NEW and OLD variables are available depending on the database operation.

Table 49. 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

Note: In a post-reinsert trigger, only the NEW variable is available, and this represents the data from the INSERT statement. For example, if Tally is not specified in the INSERT statement, new.Tally will have a default value of 0 (zero).

Example

The following database trigger uses the NEW variable to update the StateChange column when a row in the alerts.status table is modified to time stamp the change.

```
create trigger SetStateChange
group default_triggers
priority 1
before update on alerts.status
for each row
begin
    set new.StateChange = getdate;
end;
```

Creating temporal triggers (CREATE TRIGGER command)

Use the CREATE TRIGGER command to create temporal triggers that fire at a specified frequency.

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 ]
BEGIN
    trigger_action
END;
```

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

The *trigger_name* value must be unique within the ObjectServer and comply with the ObjectServer naming conventions.

The *group_name* value can be any trigger group already created by 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.

The PRIORITY of a trigger 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.

Use the optional COMMENT keyword to add a comment (*comment_string*) for the trigger.

Within a temporal trigger, you must specify how often the trigger will fire. Specify an *integer* value in seconds (the default unit of time), minutes, or hours.

Use the optional EVALUATE clause 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.

Note: The EVALUATE clause must fully qualify any tables, which are included in the SELECT statement, with a database name. For example, the following syntax is deemed invalid: `evaluate select Node from status...` The correct syntax is: `evaluate select Node from alerts.status...`

An EVALUATE clause can mostly be replaced with a FOR EACH ROW clause. Use an EVALUATE clause only when a GROUP BY clause is required.

Use the optional WHEN clause to test for a particular *condition* before the trigger action runs. If the condition is not met, the trigger action does not run.

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. However, trigger variables are static, so they maintain their value between the times when the trigger runs.

Example

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.

```
create trigger DeleteClears
group my_triggers
priority 1
every 60 seconds
begin
    delete from alerts.status where Severity = 0
        and StateChange < (getdate - 120);
end;
```

Related concepts

“Naming conventions for ObjectServer objects” on page 139

“Conditions” on page 172

Related reference

“Running commands in trigger actions” on page 215

“Best practices for creating triggers” on page 317

Creating signal triggers (CREATE TRIGGER command)

Use the CREATE TRIGGER command to create a signal trigger that fires in response to incidents in the ObjectServer, or that fires in response to a user-defined signal.

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 ]
BEGIN
    trigger_action
END;
```

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

The *trigger_name* value must be unique within the ObjectServer and comply with the ObjectServer naming conventions.

The *group_name* value can be any trigger group already created by 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.

The PRIORITY of a trigger 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.

Use the optional COMMENT keyword to add a comment (*comment_string*) for the trigger.

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.

Note: The EVALUATE clause must fully qualify any tables, which are included in the SELECT statement, with a database name. For example, the following syntax is deemed invalid: `evaluate select Node from status...` The correct syntax is: `evaluate select Node from alerts.status...`

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.

Use the optional WHEN clause to test for a particular *condition* before the trigger action runs. If the condition is not met, the trigger action does not run.

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. However, trigger variables are static, so they maintain their value between the times when the trigger runs.

Related concepts

“Naming conventions for ObjectServer objects” on page 139

“Conditions” on page 172

Related reference

“System signals and their attributes” on page 217

“Best practices for creating triggers” on page 317

Creating a user-defined signal:

Use the CREATE SIGNAL command to create a user-defined signal. When you create a signal, you define a list of data-typed attributes.

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 ObjectServer naming conventions. 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 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;

```

The following user-defined signal trigger, which is triggered by the preceding database trigger, runs an external procedure to send mail notification of the attempted delete operation.

```

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;

```

Related concepts

“Naming conventions for ObjectServer objects” on page 139

Related reference

“Specifying data types for columns” on page 145

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

```
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.

You cannot drop a signal if a trigger references it.

Syntax

```
DROP SIGNAL signal_name;
```

Running commands in trigger actions

The *trigger_action* contains a set of commands that manipulates data in the ObjectServer.

You can run 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 run the commands in the trigger body.

Attention: 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
- The IF THEN ELSE statement
- The CASE WHEN statement
- The FOR EACH ROW loop
- The FOR loop

Related reference

“Creating database triggers (CREATE TRIGGER command)” on page 208

“SET statement” on page 199

“IF THEN ELSE statement” on page 199

“CASE WHEN statement” on page 200

“FOR EACH ROW loop” on page 200

“FOR loop” on page 201

Using trigger variables in trigger conditions and actions

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
```

Tip: You can also use the % helper button to select %trigger variables.

The following table lists the read-only attributes available in the WHEN clause and action section of a trigger.

Table 50. Implicit trigger variables

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.
%trigger.num_zero_rowcount	UNSIGNED	Number of consecutive fires with zero matches in EVALUATE clause.
%rowcount	UNSIGNED	Number of rows that matched the EVALUATE clause when a temporal trigger fires. Note: This variable does not require the trigger keyword prefix. The variable value only holds true if it is checked as the first action in the trigger body. After this, the value is indeterminate.

Note: In a database trigger, the only valid trigger variable is %trigger.previous_condition. All other trigger variables provide the result for an EVALUATE clause, which is not supported for database triggers.

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;

Related reference

Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361

“Implicit user variables in procedures and triggers” on page 202

System signals and their attributes

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 by 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 following syntax:

```
%signal.at
```

Tip: You can also use the % helper button to select %signal variables.

The system signals that can be raised by the ObjectServer or the gateway are as follows:

- “startup signal”
- “shutdown signal” on page 218
- “connect signal” on page 218
- “disconnect signal” on page 218
- “backup_failed signal” on page 219
- “backup_succeeded signal” on page 219
- “login_failed signal” on page 219
- “security_timeout signal” on page 220
- “create_object signal” on page 220
- “alter_object signal” on page 221
- “drop_object signal” on page 222
- “permission_denied signal” on page 223
- “gw_counterpart_down signal” on page 224
- “gw_counterpart_up signal” on page 224
- “iduc_missed signal” on page 224
- “iduc_connect signal” on page 225
- “iduc_disconnect signal” on page 225
- “iduc_data_fetch signal” on page 226
- “resync_lock signal” on page 226
- “resync_unlock signal” on page 226
- “gw_resync_start signal” on page 227
- “gw_resync_finish signal” on page 227

Tip: You can query the catalog.primitive_signals table and the catalog.primitive_signal_parameters table 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;
```

startup signal

The startup signal is raised when the ObjectServer starts. The following table describes the attributes of this signal.

Table 51. startup signal attributes

Attributes	Data type	Description
<i>server</i>	string	Indicates the name of the ObjectServer that started.
<i>node</i>	string	Indicates the computer on which the ObjectServer started.
<i>at</i>	UTC	Indicates the time at which the ObjectServer started.

shutdown signal

The shutdown signal is raised when the ObjectServer shuts down. The following table describes the attributes of this signal.

Table 52. shutdown signal attributes

Attributes	Data type	Description
<i>server</i>	string	Indicates the name of the ObjectServer that shut down.
<i>node</i>	string	Indicates the computer on which the ObjectServer shut down.
<i>at</i>	UTC	Indicates the time at which the ObjectServer shut down.

connect signal

The connect signal is raised when a client connects to the ObjectServer. The following table describes the attributes of this signal.

Table 53. connect signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of client process that 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 that connected to the ObjectServer.
<i>node</i>	string	Indicates the name of the client computer that connected to the ObjectServer.
<i>connectionid</i>	int	Uniquely identifies the connection.
<i>at</i>	UTC	Indicates the time at which the client connected.

disconnect signal

The disconnect signal is raised when a client disconnects from the ObjectServer. The following table describes the attributes of this signal.

Table 54. disconnect signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of process that disconnected from the ObjectServer.

Table 54. *disconnect signal attributes (continued)*

Attributes	Data type	Description
<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 computer that disconnected from the ObjectServer.
<i>connectionid</i>	int	Uniquely identifies the connection.
<i>at</i>	UTC	Indicates the time at which the client disconnected.

backup_failed signal

The backup_failed signal is raised when an attempt to back up the ObjectServer fails. The following table describes the attributes of this signal.

Table 55. *backup_failed signal attributes*

Attributes	Data type	Description
<i>error</i>	string	Indicates a reason why 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 computer from which the backup was run.

backup_succeeded signal

The backup_succeeded signal is raised when the ObjectServer is successfully backed up. The following table describes the attributes of this signal.

Table 56. *backup_succeeded signal attributes*

Attributes	Data type	Description
<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 that the backup took to complete.
<i>node</i>	string	Indicates the name of the computer from which the backup was run.

login_failed signal

The login_failed signal is raised when a client fails to log in to the ObjectServer. The following table describes the attributes of this signal.

Table 57. login_failed signal attributes

Attributes	Data type	Description
<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 computer 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 signal

The security_timeout signal is raised when a login attempt to the ObjectServer times out. The following table describes the attributes of this signal.

Table 58. security_timeout signal attributes

Attributes	Data type	Description
<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 computer 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.

create_object signal

The create_object signal is raised when an object is created in the ObjectServer. The following table describes the attributes of this signal.

Table 59. *create_object* signal attributes

Attributes	Data type	Description
<i>objecttype</i>	string	Indicates the object type, which is one of the following types: <ul style="list-style-type: none"> • CREATE DATABASE • CREATE TABLE • CREATE INDEX • 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 alerts.status table is status.
<i>username</i>	string	Indicates the name of the user that ran the command.
<i>server</i>	string	Indicates the name of the ObjectServer to which the object was added.
<i>node</i>	string	Indicates the name of the computer running the ObjectServer to which the object was added.
<i>hostname</i>	string	Indicates the name of the client computer from which the request to add the object was made.
<i>at</i>	UTC	Indicates the time at which the object was added.

alter_object signal

The *alter_object* signal is raised when an object in the ObjectServer is altered. The following table describes the attributes of this signal.

Table 60. *alter_object* signal attributes

Attributes	Data type	Description
<i>objecttype</i>	string	Indicates the object type, which is one of the following types: <ul style="list-style-type: none"> • ALTER TABLE • ALTER TRIGGER GROUP • ALTER TRIGGER • ALTER PROCEDURE • ALTER RESTRICTION FILTER • ALTER USER SIGNAL • ALTER FILE • ALTER USER • ALTER GROUP • ALTER ROLE <p>Note: The ALTER PROCEDURE, ALTER RESTRICTION FILTER, and ALTER USER SIGNAL permissions are required if a CREATE OR REPLACE command is run against an object of one of these types and the object already exists. You must have the appropriate ALTER permission even though there is no ALTER command for these objects.</p>
<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 alerts.status table is status.
<i>username</i>	string	Indicates the name of the user that ran the command.
<i>server</i>	string	Indicates the name of the ObjectServer in which the object was altered.
<i>node</i>	string	Indicates the name of the computer running the ObjectServer in which the object was altered.
<i>hostname</i>	string	Indicates the name of the client computer from which the request to alter the object was made.
<i>at</i>	UTC	Indicates the time at which the object was altered.

drop_object signal

The drop_object signal is raised when an object in the ObjectServer is dropped. The following table describes the attributes of this signal.

Table 61. *drop_object* signal attributes

Attributes	Data type	Description
<i>objecttype</i>	string	Indicates the object type, which is one of the following types: <ul style="list-style-type: none"> • DROP DATABASE • DROP TABLE • DROP INDEX • 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 alerts.status table is status.
<i>username</i>	string	Indicates the name of the user that ran the command.
<i>server</i>	string	Indicates the name of the ObjectServer from which the object was dropped.
<i>node</i>	string	Indicates the name of the computer running the ObjectServer from which the object was dropped.
<i>hostname</i>	string	Indicates the name of the client computer from which the request to drop the object was made.
<i>at</i>	UTC	Indicates the time at which the object was dropped.

permission_denied signal

The permission_denied signal is raised when permission to perform an operation is denied. The following table describes the attributes of this signal.

Table 62. *permission_denied* signal attributes

Attributes	Data type	Description
<i>username</i>	string	Indicates the name of the user that made the request that caused the permission denied error.
<i>server</i>	string	Indicates the name of the ObjectServer that generated the permission denied error.
<i>node</i>	string	Indicates the name of the computer running the ObjectServer that generated the permission denied error.
<i>hostname</i>	string	Indicates the name of the client computer from which the request that caused the permission denied error was made.

Table 62. *permission_denied* signal attributes (continued)

Attributes	Data type	Description
<i>at</i>	UTC	Indicates the time at which the permission denied error occurred.
<i>sql_cmd</i>	string	Indicates the SQL command that caused the permission denied error.

gw_counterpart_down signal

The gateway raises a *gw_counterpart_down* signal in the backup ObjectServer when it detects that the primary ObjectServer is unavailable. The following table describes the attributes of this signal.

Table 63. *gw_counterpart_down* signal attributes

Attributes	Data type	Description
<i>server</i>	string	Indicates the name of the counterpart ObjectServer that failed, in a failover/failback pair.
<i>node</i>	string	Indicates the name of the computer from which the counterpart ObjectServer was run.
<i>at</i>	UTC	Indicates the time at which the counterpart ObjectServer failed.
<i>gateway_name</i>	string	Indicates the name of the gateway between the primary and backup ObjectServers.

gw_counterpart_up signal

The gateway raises a *gw_counterpart_up* signal in the backup ObjectServer when it detects that the primary ObjectServer is available again. The following table describes the attributes of this signal.

Table 64. *gw_counterpart_up* signal attributes

Attributes	Data type	Description
<i>server</i>	string	Indicates the name of the counterpart ObjectServer that is available again, in a failover/failback pair.
<i>node</i>	string	Indicates the name of the computer from which the counterpart ObjectServer was run.
<i>at</i>	UTC	Indicates the time at which the counterpart ObjectServer failed.
<i>gateway_name</i>	string	Indicates the name of the gateway between the primary and backup ObjectServers.

iduc_missed signal

The *iduc_missed* signal is raised whenever a desktop or gateway client fails to respond to an IDUC prompt from the ObjectServer. The following table describes the attributes of this signal.

Table 65. *iduc_missed* signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of client process that failed to respond to the IDUC prompt from the ObjectServer.
<i>description</i>	string	Contains additional information about the client, where available.
<i>username</i>	string	Indicates the name of the user that is connected to the ObjectServer.
<i>node</i>	string	Indicates the name of the client computer that is connected to the ObjectServer.
<i>connectionid</i>	int	Uniquely identifies the connection.
<i>at</i>	UTC	Indicates the time at which the IDUC cycle was missed.
<i>missed_cycles</i>	int	Indicates the number of consecutively-missed IDUC cycles.

iduc_connect signal

The *iduc_connect* signal is raised when a client establishes an IDUC connection. The following table describes the attributes of this signal.

Table 66. *iduc_connect* signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of client process that 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 that connected to the ObjectServer.
<i>node</i>	string	Indicates the name of the client computer that connected to the ObjectServer.
<i>conn_id</i>	int	Indicates the ID of the connection.

iduc_disconnect signal

The *iduc_disconnect* signal is raised when a client disconnects an established IDUC connection. The following table describes the attributes of this signal.

Table 67. *iduc_disconnect* signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of client 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.

Table 67. *iduc_disconnect* signal attributes (continued)

Attributes	Data type	Description
<i>node</i>	string	Indicates the name of the client computer that disconnected from the ObjectServer.
<i>conn_id</i>	int	Indicates the ID of the connection.

iduc_data_fetch signal

The *iduc_data_fetch* signal is raised whenever an IDUC client retrieves its IDUC changes from the ObjectServer. The following table describes the attributes of this signal.

Table 68. *iduc_data_fetch* signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of client process that requested pending IDUC changes from the ObjectServer.
<i>description</i>	string	Contains additional information about the client, 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 is connected to the ObjectServer.
<i>node</i>	string	Indicates the name of the client computer that is connected to the ObjectServer.
<i>connectionid</i>	int	Uniquely identifies the connection.
<i>at</i>	UTC	Indicates the time at which the client retrieved changes corresponding to the last IDUC notification.

resync_lock signal

The *resync_lock* signal is raised by the ObjectServer when the resync lock is locked. The following table describes the attributes of this signal.

Table 69. *resync_lock* signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of client process that locked the resync lock.
<i>description</i>	string	Contains additional information about the client that locked the resync lock, 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 running the process.
<i>node</i>	string	Indicates the name of the client computer connected to the ObjectServer.
<i>at</i>	UTC	Indicates the time at which the signal occurred

resync_unlock signal

The *resync_unlock* signal is raised by the ObjectServer when the resync lock is unlocked. The following table describes the attributes of this signal.

Table 70. *resync_unlock* signal attributes

Attributes	Data type	Description
<i>process</i>	string	Indicates the type of client process that unlocked the resync lock.
<i>description</i>	string	Contains additional information about the client that unlocked the resync lock, 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 running the process.
<i>node</i>	string	Indicates the name of the client computer connected to the ObjectServer.
<i>at</i>	UTC	Indicates the time at which the signal occurred

gw_resync_start signal

The `gw_resync_start` signal is raised by the gateway to indicate the start of a resynchronization operation. The following table describes the attributes of this signal.

Table 71. *gw_resync_start* signal attributes

Attributes	Data type	Description
<i>gateway_name</i>	string	Indicates the name of the gateway that is starting the resynchronization.
<i>node</i>	string	Indicates the host name of the computer on which the gateway is running.
<i>at</i>	UTC	Indicates the time at which the resynchronization started.
<i>is_master</i>	Boolean	Indicates whether the local ObjectServer is the master or slave of the resynchronization.

gw_resync_finish signal

The `gw_resync_finish` signal is raised by the gateway to indicate the end of a resynchronization operation. The following table describes the attributes of this signal.

Table 72. *gw_resync_finish* signal attributes

Attributes	Data type	Description
<i>gateway_name</i>	string	Indicates the name of the gateway that is finishing the resynchronization.
<i>node</i>	string	Indicates the host name of the computer on which the gateway is running.
<i>at</i>	UTC	Indicates the time at which the resynchronization finished.
<i>is_master</i>	Boolean	Indicates whether the local ObjectServer is the master or slave of the resynchronization.

Related reference

Appendix B, “SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists,” on page 361

Creating triggers for accelerated event notification

To support accelerated event notification, create post-insert, post-update, or post-reinsert triggers that are attached to the alerts.status table. In the triggers, set up conditions to define or identify accelerated events when they are inserted or updated in the alerts.status table, and to forward such events to the relevant Accelerated Event Notification clients.

Two SQL commands are available for use with your triggers: an event fast-track (or accelerated event) command (IDUC EVTFT) and a send message command (IDUC SNDMSG).

Tip: You might find it useful to group triggers that support accelerated event notification within their own trigger group.

Related concepts

Chapter 6, “Configuring accelerated event notification,” on page 239

Activating accelerated event notification (IDUC EVTFT command):

Use the IDUC EVTFT command to activate pop-up notifiers for accelerated events to be sent to clients, and to enable click-across functionality to the desktop event list or the Web GUI Active Event List.

Syntax

IDUC EVTFT *destination*, *action_type*, *row*

The variables in this command can take the following values:

- *destination* = *spid* | *iduc_channel*
- *spid* = *integer_expression* (The literal client connection ID)
- *iduc_channel* = *string_expression* (Channel name)
- *action_type* = INSERT | UPDATE | DELETE
- *row* = *variable* (Variable name reference of a row in the automation)

For example, if you have set up an accelerated event flag within your probe rules file and added a column for this flag to the alerts.status table, you can add a condition within a post-insert trigger to examine the value within this column. If the value is satisfied for accelerated event notification, the event is then forwarded as a pop-up notification to specific Accelerated Event Notification clients. You can define the condition in the trigger by using the following format:

```
begin
  if ( new.accelerated_event_column_name = 1 )
    then
      iduc evtft 'channel_name' , insert , new ;
    end if;
end;
```

In this syntax, *accelerated_event_column_name* is the name of the column that holds accelerated event flag in the alerts.status table, and *channel_name* is the name of a channel over which accelerated event data is broadcast. Note that the channel name is case-sensitive, so ensure that you use the correct case within the syntax.

Example

```
create or replace trigger evtft_insert
group channel_triggers
priority 1
comment 'Fast track critical events from alerts.status'
after insert on alerts.status
for each row
begin
    if ( new.FastTrack = 1 )
    then
        iduc evtft 'FastTrack' , insert , new ;
    end if;
end;
```

Sending messages to AEN clients (IDUC SNDMSG command):

Use the IDUC SNDMSG command to send information messages to an Accelerated Event Notification client.

Syntax

IDUC SNDMSG *destination*, *string_expression*

The variables in this command can take the following values:

- *destination* = *spid* | *iduc_channel*
- *spid* = *integer_expression* (The literal client connection ID)
- *iduc_channel* = *string_expression* (Channel name)
- *string_expression* = Descriptive text to be sent as a message

Example

```
create trigger notify_isqlconn
group default_triggers
priority 1
on signal connect
begin
    if( %signal.process = 'isql' )
    then
        iduc sndmsg 'notif_isql', 'ISQL Connection from ' +
            %signal.node + ' from user ' +
            %signal.username + ' at ' +
            to_char(%signal.at)
    end if;
end;
```

Modifying a trigger (ALTER TRIGGER command)

Use the ALTER TRIGGER command to change the settings of an existing trigger. You can change more than one setting in a single ALTER TRIGGER command.

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 TRUE to activate a trigger or SET ENABLED FALSE to 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.

Example

```
alter trigger mytrig set priority 1;
```

Deleting a trigger (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 dependent objects.

Example

```
drop trigger mytrig;
```

Standard Tivoli Netcool/OMNIBus automations

A set of standard automations is included with Tivoli Netcool/OMNIBus. These automations are created during database initialization.

The standard automations are stored in the location: \$NCHOME/omnibus/etc/automation.sql. You can open the automation.sql file within a text editor and view the syntax of each automation. Comments are included to describe the purpose of the automations. Some of the automations in the automation.sql file are not enabled by default.

You can also use the Netcool/OMNIBus Administrator to browse through these automations by selecting the **Automation** menu button from the Netcool/OMNIBus Administrator window. You might notice that some trigger groups are disabled by default, whereas the triggers belonging to that trigger group have an enabled state. You must enable such trigger groups in order to run the triggers. One such example is the audit_config trigger group, which provides the ability to raise alerts whenever changes are made to the ObjectServer objects. This trigger group can be used as an audit mechanism in conjunction with the audit log files that are written to the \$NCHOME/omnibus/log directory.

The functions performed by some of the standard automations include:

- Backing up the ObjectServer
- Adding alerts to the ObjectServer
- Inserting journal entries
- Removing redundant entries from various tables

The standard automations are listed in the following table.

Table 73. Standard automations

Trigger or procedure name	Description
audit_config_alter_class	Creates an alert indicating that a class has been altered.
audit_config_alter_col_visual	Creates an alert indicating that a column visual has been altered.
audit_config_alter_conv	Creates an alert indicating that a conversion has been altered.
audit_config_alter_menu	Creates an alert indicating that a menu has been altered.
audit_config_alter_object	Creates an alert indicating that an object has been altered.
audit_config_alter_prompt	Creates an alert indicating that a prompt has been altered.
audit_config_alter_property	Creates an alert indicating that a property has been altered.
audit_config_alter_tool	Creates an alert indicating that a tool has been altered.
audit_config_create_class	Creates an alert indicating that a class has been created.
audit_config_create_col_visual	Creates an alert indicating that a column visual has been created.
audit_config_create_conv	Creates an alert indicating that a conversion has been created.
audit_config_create_menu	Creates an alert indicating that a menu has been created.
audit_config_create_object	Creates an alert indicating that an object has been created.
audit_config_create_prompt	Creates an alert indicating that a prompt has been created.
audit_config_create_tool	Creates an alert indicating that a tool has been created.
audit_config_drop_class	Creates an alert indicating that a class has been dropped.
audit_config_drop_col_visual	Creates an alert indicating that a column visual has been dropped.
audit_config_drop_conv	Creates an alert indicating that a conversion has been dropped.
audit_config_drop_menu	Creates an alert indicating that a menu has been dropped.
audit_config_drop_object	Creates an alert indicating that an object has been dropped.
audit_config_drop_prompt	Creates an alert indicating that a prompt has been dropped.
audit_config_drop_tool	Creates an alert indicating that a tool has been dropped.
audit_config_permission_denied	Creates an alert indicating that a permission has been denied.

Table 73. Standard automations (continued)

Trigger or procedure name	Description
automatic_backup	Backs up all ObjectServer memory stores to a sequence of locations dependent on the defined value of a <i>num_backups</i> variable.
automation_disable	Disables the automations that should not be running when the ObjectServer is a backup ObjectServer.
automation_enable	Enables the automations that should be running when the ObjectServer is a primary ObjectServer.
backup_counterpart_down	Enables the automations that should be running when the primary ObjectServer goes down, and the backup ObjectServer is acting as the primary ObjectServer.
backup_counterpart_up	Disables the automations that should not be running in the backup ObjectServer when the primary ObjectServer restarts.
backup_failed	Specifies an action to perform on a failed backup operation.
backup_startup	Disables the automations that should not be running when an ObjectServer designated as a backup, is started.
backup_state_integrity	Ensures that only one record is present in the backup state table by cancelling any other inserts.
backup_succeeded	Specifies an action to perform on a successful backup operation.
clean_details_table	Performs housekeeping cleanup on the alerts.details table. Deletes any entries not found in the alerts.status table.
clean_journal_table	Performs housekeeping cleanup on the alerts.journal table. Deletes any entries not found in the alerts.status table.
connection_watch_connect	Creates an alert when a new client connects. The process or application name identified by the signal is matched against the alerts.application_types table to identify the appropriate severity and event type for the connect. A gateway connection, for example, is treated as a resolution (clearing a disconnect), whereas an event list connect is a Type 1 event, which will be resolved by a disconnect.
connection_watch_disconnect	Creates an alert when a client disconnects. The process or application name identified by the signal is matched against the alerts.application_types table to identify the appropriate severity and event type for the disconnect. A gateway disconnection, for example, is treated as a problem, whereas an event list disconnection is a resolution.
dedup_status_inserts	Counts deduplicated status table inserts.
deduplicate_details	Deduplicates rows on the alerts.details table.
deduplicate_iduc_stats	Deduplicates rows on the iduc_system.iduc_stats table.

Table 73. Standard automations (continued)

Trigger or procedure name	Description
deduplication	Deduplication processing for the alerts.status table. Maintains the deduplication tally and refreshes alert details.
delete_clears	Every 60 seconds, deletes clear alerts that are older than two minutes in the alerts.status table.
details_inserts	Counts details table inserts.
disable_inactive_users	Runs once a day to disable users who have not logged on to the ObjectServer within a defined period.
disable_user	Disables users when they fail to log on after <i>n</i> consecutive failures.
disconnect_iduc_missed	Disconnects real-time clients that have not communicated with the ObjectServer for 100 granularity periods.
escalate_off	Sets Flash field to 0 (not flashing) and SuppressEscl to 0 (not escalated in this example) when an event that has previously had the Flash field set to 1 is Acknowledged or if the event is Cleared (Severity = 0).
expire	Handles the expiration of alerts. Sets the Severity of an alert to 0 if the value of ExpireTime (during which the alert is valid) is exceeded.
flash_not_ack	Sets Flashing on (Flash = 1) for events that are 10 minutes old and Critical (Severity = 5), but which have not yet been acknowledged by a user (Acknowledge = 0). It sets SuppressEscl to 1 as a further indication of the escalation status of the event.
generic_clear	Clears (Severity = 0) all rows in the alerts.status table indicating a down device (Type = 1), where there is a subsequently inserted row indicating that the device has come back up (Type = 2).
iduc_messages_tblclean	Performs housekeeping cleanup on the alerts.iduc_messages table. Runs every 60 seconds and deletes messages older than two minutes.
iduc_stats_insert	Inserts a client entry into the iduc_system.iduc_stats table when the iduc_connect signal is raised.
iduc_stats_update	Updates the LastIducTime field in the iduc_system.iduc_stats table when the iduc_data_fetch signal is raised.
jinsert	Inserts a record into the alerts.journal table. Automations that require journal entries should run this procedure.
journal_inserts	Counts journal table inserts.
mail_on_critical	Send e-mail about critical alerts that are unacknowledged after 30 minutes. Note: This tool is UNIX specific unless an equivalent NT mailer is available.
new_row	Sets default values for new alerts in the alerts.status table.
new_status_inserts	Counts new status table inserts.

Table 73. Standard automations (continued)

Trigger or procedure name	Description
pass_deletes	Deletes from the destination ObjectServer, rows that do not exist in the source ObjectServer, after resynchronization.
profiler_group_report	Writes to the profiler_report file, a row for the sum of the amount of time taken by each distinct application type during the last profiling period.
profiler_report	Writes into the profiler_report file, a row for each connected client with the amount of time taken by that client during the last profiling period.
profiler_toggle	Reports that the profiler has been toggled.
reset_user	Resets the failure count of a user when they log on successfully.
resync_finished	Identifies when resynchronization is complete and sets the ActingPrimary property of the backup ObjectServer to FALSE to define it as the backup.
security_watch_security_failure	Creates an alert when a client fails to authenticate.
service_insert	Service processing for the service.status table.
service_reinsert	Service processing for the service.status table.
service_update	Service processing for the service.status table.
state_change	State change processing for the alerts.status table. Maintains the ObjectServer timestamp of the last insert and update of an alert from any source.
statistics_cleanup	Deletes statistics over an hour old.
statistics_gather	Collects metrics such as the total number of clients that are connected to the ObjectServer, the number of real-time clients, and the number of new inserts into the alerts.status table, and inserts the metrics into the master.stats table. This data can then be viewed using Netcool/OMNIBus Administrator or nco_sql , or can be written to file, or processed by other automations.
stats_reset	Resets the statistics data.
system_watch_shutdown	Creates an alert indicating that the ObjectServer is being shut down.
system_watch_startup	Creates an alert indicating that the ObjectServer has started.
trigger_stats_report	Writes to the trigger_stats.log file, the amount of time each trigger has used in the last profiling period.

Table 73. Standard automations (continued)

Trigger or procedure name	Description
update_service_affecting_events	<p>Runs at a specified frequency to enable service-affected events in Network Manager IP Edition to automatically clear when all their related events are cleared. A service-affected event is an alert that warns operators that a critical customer service has been affected by one or more network events.</p> <p>This automation works only with Tivoli Netcool/OMNIBus V7 .0, or later. Tip: The automation is required only if Network Manager IP Edition is being used, and is used with the precision.entity_service, precision.service_details, and precision.service_affecting_event tables.</p>
webtop_compatibility	<p>Populates the master.profiles table with ObjectServer users for the Web GUI (or Netcool/Webtop) to read. Additionally sets the AllowISQL field for each user who has been granted permission to use the interactive SQL tool in the Web GUI (or Netcool/Webtop).</p>

Automation for service-affected events

A service-affected event (SAE) is an alert that warns operators that a critical customer service has been affected by one or more network events. Service-affected events are generated within IBM Tivoli Network Manager IP Edition.

You can configure Tivoli Netcool/OMNIBus to run an automation at a specified frequency to enable service-affected events in Network Manager IP Edition to automatically clear when all their related events are cleared.

To make this feature operational for an installation of Network Manager IP Edition and Tivoli Netcool/OMNIBus, you must configure Network Manager IP Edition as described in the *IBM Tivoli Network Manager IP Edition Installation and Configuration Guide*, SC27-2760-00. When you install Tivoli Netcool/OMNIBus, the following required Objectserver objects are added, to support SAE operation:

- Database tables for SAE application usage: precision.entity_service, precision.service_details, precision.service_affecting_events
- The NmosEntityId field to the alerts.status table
- An sae trigger group and the update_service_affecting_events trigger
The update_service_affecting_events trigger automation works only with Tivoli Netcool/OMNIBus V7 .0, or later.
- Event list tools for managing service-affected events on UNIX and Windows

From the Tivoli Netcool/OMNIBus event list, you can monitor service-affected events as follows:

- To show the underlying events (for example, linkDowns) that are associated with a service-affected event, select the service-affected event, right-click it and then click **Show SAE Related Events** from the pop-up menu. All the events that are associated with the selected event are displayed in a new window.
- To show the service-affected events to which an event (for example, a linkDown) is related, select the event, right-click it and then click **Show SAE Related Services** from the pop-up menu. A list of all related service-affected events is

displayed in a new window. For example, if you selected a linkDown event in the event list, this window displays all the services that are affected by that linkDown event.

Automation examples

This topic contains examples of some commonly performed automations.

Example: 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;
```

Example: 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;
```

Example: 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;
```

Example: 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 time stamp 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;
```

Example: Trigger to delete clear rows

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;
```

Example: Trigger to send e-mail notifications for 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
enabled false
priority 1
comment 'Send email about critical alerts that are
unacknowledged after 30 minutes. NOTE This tool is
UNIX specific unless an equivalent NT mailer is available.'
every 10 seconds
begin
    for each row critical in alerts.status where critical.Severity = 5 and
        critical.Grade < 2 and critical.Acknowledged = 0 and
        critical.LastOccurrence <= ( getdate() - (60*30) )
    begin
        execute send_email( critical.Node, critical.Severity, 'Netcool Email',
            '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 '$NCHOME/omnibus/utills/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 quotation marks, and the quotation marks must be escaped with backslashes. All quotation marks in this example are single quotation marks.

Example: Procedure to insert a journal entry for triggers

The `jinsert` procedure enables you to insert rows into the `alerts.journal` table. Automations that require journaling entries call the procedure, and pass in the serial number of the row, the user ID of the user making the change (if applicable), the time when the action occurred, and any descriptive text for the action being journaled.

```
create or replace trigger
trigger_name
group default_triggers
priority 10
before delete on alerts.status
for each row
begin
execute jinsert( old.Serial, %user.user_id, getdate(), 'string');
end;
```

In this automation, *trigger_name* is a variable representing the name of the trigger, and *string* is the journal entry text.

Chapter 6. Configuring accelerated event notification

You can configure Tivoli Netcool/OMNIBus for accelerated event notification of events that could present a risk to the system. The Accelerated Event Notification (AEN) system provides a means of accelerating high-priority events to help ensure that systems can continue to run without interruption.

When configuring accelerated event notification, use the following guidelines:

- Determine the conditions under which you want to accelerate events. Consider whether to use the probe rules file to flag events for acceleration, or whether to set up conditions within a post-insert, post-update, or post-reinsert trigger in the ObjectServer.

If the conditions under which events should be accelerated are set up in the rules file, and are complex, you might need to configure the alerts.status table with a dedicated column to receive a flag that identifies an event as an accelerated event.

Typically, you can run a post-insert trigger on inserted events to determine whether they meet the conditions for acceleration, or run a post-update trigger on updated events that have been enriched with information from other sources, or run a post-reinsert trigger to escalate repeating events.

- Determine whether you require a dedicated gateway for forwarding accelerated events. Consider the use of a dedicated gateway if you want to eliminate the possibility of unexpected delays caused by high volumes of events, which may largely be non-critical, or if you want to send accelerated events to a different ObjectServer than that used for normal IDUC updates. If using a dedicated gateway, you must configure that gateway to replicate only accelerated inserts and updates.

You also need to use Netcool/OMNIBus Administrator to configure accelerated event notification, as follows:

- Set up a dedicated event column to flag an event for acceleration, if required.
- Set up channels to define the type of event data to be included in accelerated event notifications, and the recipients of this event data.
- Configure post-insert, post-update, or post-reinsert triggers to act on accelerated events that are inserted or updated in the alerts.status table.

For information on monitoring and managing accelerated events, see the *IBM Tivoli Netcool/OMNIBus User's Guide*.

Configuring a probe to flag events for acceleration

If the conditions for accelerated event notification are complex, determine whether to set up the conditions within the probe rules file.

You might also need to add a dedicated event column to the alerts.status table to flag events for acceleration, and use this field in your probe rules file.

The following sample probe rules file depicts how an event stream can be parsed in order to determine which events are considered high priority. At the top of the rules file, elements (indicated by the \$ symbol) are assigned to ObjectServer fields

(indicated by the @ symbol). The conditional statement uses the \$Summary element to set the AlertKey and FastTrack values in the alerts.status table.

The statement translates to: if the Summary value begins with 'Port failure', then insert the port number value into the AlertKey field in the alerts.status table, and insert a value of 1 into the FastTrack field in the alerts.status table. Otherwise, if the Summary value begins with the string 'Diskspace', insert the concatenated percent value and % full string into the AlertKey field in the alerts.status table.

```
@Manager = "Simnet Probe"
@Class   = 3300
@Node    = $Node
@Agent   = $Agent
@AlertGroup = $Group
@Summary = $Summary
@Severity = $Severity
@Identifier = $Node + $Agent + $Severity + $Group

if (nmatch($Summary, "Port failure"))
{
  @AlertKey = $PortNumber
  @FastTrack = 1
}
else if (nmatch($Summary, "Diskspace"))
{
  @AlertKey = $PercentFull + "% full"
}
```

Configuring a gateway for accelerated event notification

You can choose to use a dedicated gateway to forward accelerated events mainly for performance reasons or to send the events to a particular ObjectServer. If using a dedicated gateway, you must update its table replication definition file so that only accelerated inserts and updates are replicated forward.

To configure a unidirectional ObjectServer gateway for accelerated event notification:

1. Open the following gateway table replication definition file:
\$OMNIHOME/gates/objserv_uni/objserv_uni.reader.tblrep.def
2. Locate the following lines:
REPLICATE ALL FROM TABLE 'alerts.status' USING MAP 'StatusMap';
REPLICATE ALL FROM TABLE 'alerts.journal' USING MAP 'JournalMap';
REPLICATE ALL FROM TABLE 'alerts.details' USING MAP 'DetailsMap';
3. Replace these three lines with the following single line:
REPLICATE FT_INSERT,FT_UPDATE FROM TABLE 'alerts.status' USING MAP 'StatusMap';
4. Save the file.

What to do next

For a bidirectional gateway, you must modify the corresponding files in a similar manner. The gateway table replication definition file for a bidirectional gateway is \$OMNIHOME/gates/objserv_bi/objserv_bi.reader.tblrep.def.

Configuring the alerts.status table to receive the AEN flag

If you have configured your probe rules file with a flag for accelerated event notification, you might need to add a column to support the acceleration of events, to the alerts.status table.

For example, from Netcool/OMNIBus Administrator, add a column with a **Column Name** of FastTrack and a **Data Type** of Integer.

Related tasks

“Adding and editing table columns” on page 124

Related reference

“Altering a table” on page 147

Configuring channels to broadcast event data

When configuring accelerated event notification, you must use channels to define the type of event data to broadcast in the accelerated event notifications, and the recipients of this data. You can set up multiple channels with varied event data and recipients.

Channel administration is permitted only for users with the ChannelAdmin role.

The event data for channels is derived from columns in the ObjectServer tables.

Tip: The columns that you choose for a channel should contain sufficient summary data to help operators interpret critical issues at first glance, when such issues are forwarded to the screen as pop-up notifiers. Operators can then click across to the desktop event list or the Web GUI Active Event List to obtain full details for the event and to manage the event.

Note that click-across functionality from a pop-up notifier to the event list or the Active Event List is available only for events in the alerts.status table.

Creating and editing channels

You must create channels on an ObjectServer from which the accelerated events will be forwarded.

Before you begin

You must be connected to the ObjectServer on which you want to create a channel.

To create or edit a channel:

1. From the Netcool/OMNIBus Administrator window, select the **System** menu button.
2. Click **Channels**. The Channels pane opens.
3. To add a channel, click **Add Channel** in the toolbar. The Channel Details window opens.
4. To edit a channel, select the channel to edit and then click **Edit Channel** in the toolbar. The Channel Details window opens.
5. Define the channel as follows:

Name Type a unique name for the channel. If you are editing a channel, you cannot change the name.

Description

Type meaningful text that summarizes the function of the channel.

6. From the **Columns** tab, specify which columns you want to include in the channel definition. Complete the tab as follows:

Add new Channel Columns

Click this button to add columns to the channel. The Channel Column Details window opens. Complete this window as follows, and then save your changes:

Table From the left list, select an ObjectServer database. From the right list, select a table in that database.

If you are editing channel columns, you cannot change the database or table name.

Restriction: Currently, support is available only for events in the alerts.status table.

Columns: Available

This list is populated with the names of columns that are defined in the selected database table, and which you can assign to the channel. To assign one or more of these columns, use the arrow keys to move the columns to the **Selected** list.

To move all columns to the **Selected** list, click >>. To move a single column or multiple columns to the **Selected** list, select each column and then click >. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.

You can also double-click a column to move it from the **Available** list to the **Selected** list. Columns are added to the end of the **Selected** list.

Columns: Selected

This list contains the columns that are included in the channel definition. To remove columns from the channel definition, use the arrow keys to move the columns to the **Available** list.

To move all columns to the **Available** list, click <<. To move a single column or multiple columns to the **Available** list, select each column and then click <. You can use the SHIFT key for consecutive selections, or the CTRL key for non-consecutive selections.





You can also double-click a column to move it from the **Selected** list to the **Available** list.

Use the arrow buttons to the right of the **Selected** list to specify the position of the column data within the pop-up notifier in the accelerated event notification client. The **Selected** list displays the following default markers to indicate the position of the column data:

- H: Heading
- F: First Line
- S: Second Line
- M: Main Message
- N: Note

To change the position of a column, select it and then click the relevant arrow button. Use the arrow buttons as follows:

Table 74. Arrow buttons

Button	Description
	Moves the selected column to the top of the Selected list.
	Moves the selected column one position up in the Selected list.
	Moves the selected column one position down in the Selected list.
	Moves the selected column to the bottom of the Selected list.

When you return to the **Columns** tab in the Channel Details window, your column selections are shown within a single row. (You cannot add more than one row.)

Edit selected Channel Columns

Click this button to edit the column definition for the channel. Select the row in the **Columns** tab and then click the button. The Channel Column Details window opens. Amend your column selections within this window and then save your changes to return to the **Columns** tab.

Delete selected Channel Columns

Use this button to delete a column definition for the channel. Select the row in the **Columns** table and then click **Delete**. When prompted, confirm the deletion. Your changes are reflected in the **Columns** tab.

- From the **Recipients** tab, specify the user or group of users to whom the channel data should be sent. Complete the tab as follows:

Add new Channel Recipient

Click this button to specify recipients for the channel information. The Channel Recipient Details window opens. Complete this window as follows, and then save your changes:

isGroup

Select this check box to indicate that the channel recipients are a group of users. Clear this check box if the channel recipient is a single user.

Name This list works in conjunction with the **isGroup** check box, and is populated either with a list of all users or a list of all groups in the ObjectServer. Select the name of the user or group who should receive the channel data.

Hostname

Type the name of the connected host. You can use regular expressions to filter on connections that match this value. Leave the field blank for a match on any host name. For example, enter the regular expression `*test*` to match any host with the string test in its name.

Application Name

Type the name of an application that is connected to the ObjectServer. You can use regular expressions to filter on connections that match your entry. For example, `*event*`

indicates match any host with the string event in its name.
Leave this field blank to match on any application name.

Application Description

Type an application description. You can use regular expressions to filter on connections that match your entry. For example, **real time** matches any host with the string *real time* in its application description. Leave this field blank to match on any application description.

When you return to the **Recipients** tab in the Channel Details window, the recipient details are shown in a single row. (You can add further rows of recipients.)

Edit the selected Channel Recipient

Click this button to edit the recipient details for any selected row in the **Recipients** tab. The Channel Recipient Details window opens. Amend the recipient details and save your changes to return to the **Recipients** tab.

Delete the selected Channel Recipient

Use this button to remove recipients from the channel definition. Select the row of recipients that you want to remove and then click this button. When prompted, confirm the deletion. Your changes are reflected in the **Recipients** tab.

8. Save or cancel your changes as follows:

OK Click this button to save the channel details and close the window. New channels are added to the Channels pane.

Cancel Click this button to close the window without saving your changes.

Copying and pasting channels

You can use one channel as a template for another by copying and pasting the channel definition. This is useful if you want to create channels with slight variations in their definitions.

Restriction: You cannot copy and paste between ObjectServers.

To copy and paste a channel:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Channels**. The Channels pane opens.
3. To copy a channel, select the channel from the Channels pane and then click **Copy** in the toolbar.
4. To paste the channel, click **Paste** in the toolbar. The New Channel window opens.
5. Enter a unique name for the channel.
6. Confirm or cancel your actions as follows:
 - Click **OK** to create the new channel with an identical channel definition as the selected channel. The Channel Details window then opens so that you can make the required changes to this new channel.
 - Click **Cancel** to cancel the copy-and-paste action.

Deleting a channel

To delete a channel:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click the **Channels** icon. The Channels window opens.
3. Select the channel that you want to delete and click **Delete** in the toolbar. The channel is deleted.

Sending messages to channel recipients

You can send messages to channel recipients who are currently running the Accelerated Event Notification client.

You can send messages to recipients listening on a single channel, or on multiple channels.

To send messages:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Channels**. The Channels pane opens.
3. From the Channels pane, select one or more channels that are associated with the message to be sent. You can use the Shift key for consecutive selections or the Ctrl key for non-consecutive selections.
4. From the toolbar, click **Send Message**. The Send Message window opens.
5. Type a text message in the **Message Text** field. This field scrolls horizontally to allow for the entry of text.
6. Confirm or cancel your actions as follows:
 - Click **OK** to initiate the send action and then confirm that you want to send the message by clicking **Yes**.

Note: If the **Message Text** field is blank, no message is sent.

- Click **Cancel** to cancel the send action.

Results

The message is displayed in a message box on all relevant Accelerated Event Notification client screens. The message text word wraps at 120 characters. If you sent the message to multiple channels, a client listening on more than one of these channels receives the message once only.

Disconnecting Accelerated Event Notification clients

If you need to perform minor maintenance on the ObjectServer, such as resynchronization, you can remotely disconnect (or sign out) the Accelerated Event Notification clients that are currently running. As part of the disconnect action, you can enter a brief message to users with relevant information.

To disconnect Accelerated Event Notification clients that are listening on one or more channels:

1. From the Netcool/OMNIbus Administrator window, select the **System** menu button.
2. Click **Channels**. The Channels pane opens.

3. From the Channels pane, select one or more channels that are associated with the disconnect action. You can use the Shift key for consecutive selections or the Ctrl key for non-consecutive selections.
4. From the toolbar, click **Disconnect Clients**. The Send Disconnect Command window opens.
5. Enter a text message in the **Reason for Disconnect** field. This field scrolls horizontally to allow for the entry of text.
6. Confirm or cancel your actions as follows:
 - Click **OK** to initiate the disconnect action and then confirm that you want to disconnect by clicking **Yes**.

Note: If the **Reason for Disconnect** field is blank, the clients will disconnect, but no reason for the action is given in the message box.

- Click **Cancel** to cancel the disconnect action.

Results

After you confirm the disconnect action, any message that you entered is displayed in a message box on all relevant Accelerated Event Notification client screens. The message text word wraps at 120 characters. An automatic sign-out then occurs. The status indicator of the Accelerated Event Notification clients reflects the signed out state, although the clients continue to run in the background.

Shutting down Accelerated Event Notification clients

If you need to shut down the ObjectServer, you can remotely shut down the Accelerated Event Notification clients that are currently running. As part of shutting down, you can enter a brief message to users with relevant information.

To shut down Accelerated Event Notification clients that are listening on one or more channels:

1. From the Netcool/OMNIBus Administrator window, select the **System** menu button.
2. Click **Channels**. The Channels pane opens.
3. From the Channels pane, select one or more channels that are associated with the shutdown action. You can use the Shift key for consecutive selections or the Ctrl key for non-consecutive selections.
4. From the toolbar, click **Shutdown Clients**. The Send Shutdown Command window opens.
5. Enter a text message in the **Reason for Shutdown** field. This field scrolls horizontally to allow for the entry of text.
6. Confirm or cancel your actions as follows:
 - Click **OK** to initiate the shutdown action and then confirm that you want to shut down by clicking **Yes**.

Note: If the **Reason for Shutdown** field is blank, the clients will shut down, but no reason for the action is given in the message box.

- Click **Cancel** to cancel the shutdown action.

Results

After you confirm the shutdown action, any message that you entered is displayed in a message box on all relevant Accelerated Event Notification client screens. The

message text word wraps at 120 characters. After five seconds, an automatic sign-out and exit then occurs.

Configuring triggers to support accelerated event notification

To support accelerated event notification, create post-insert, post-update, or post-reinsert triggers that are attached to the alerts.status table. In the triggers, set up conditions to define or identify accelerated events when they are inserted or updated in the alerts.status table, and to forward such events to the relevant Accelerated Event Notification clients.

Two SQL commands are available for use with your triggers: an event fast-track (or accelerated event) command (IDUC EVTFT) and a send message command (IDUC SNDMSG).

Tip: You might find it useful to group triggers that support accelerated event notification within their own trigger group.

Related concepts

“Configuring automation using triggers” on page 205

“Configuring triggers” on page 97

Related reference

“Activating accelerated event notification (IDUC EVTFT command)” on page 228

“Sending messages to AEN clients (IDUC SNDMSG command)” on page 229

Chapter 7. Using process control to manage processes and external procedures

The Tivoli Netcool/OMNIbus process control system performs two primary tasks. It manages local and remote processes, and runs external procedures that are specified in automations.

You can use process control to simplify the management of Tivoli Netcool/OMNIbus components such as ObjectServers, probes, and gateways. You can install process agents on each host and configure them to manage processes. The configured process agents cooperate automatically and understand their own configuration. They start processes and can keep those processes running. You can define processes that are dependent on other processes, and processes that have timed threshold dependencies. If a managed host is restarted, the process agent can be configured to restart local components automatically.

The process control system includes a set of command-line utilities that provide an interface to process management. You can configure and manage process control either from the command line or by using Netcool/OMNIbus Administrator.

How process agents connect

You can set up a process control network system by configuring process control on several Tivoli Netcool/OMNIbus hosts.

The process agents can then communicate with each other and run programs on request. Process agents running on Windows operating systems can communicate with process agents running on UNIX operating systems, and vice versa.

Running process agents in a routing configuration

When several process agents are connected by routing statements, each process agent in the process control network can be made aware of processes in the other process agents. A process agent configuration file is used to define processes, services, and hosts in the process control network, and to define routing statements.

The process control system supports full remote management of your process agents from a single console. You can add, modify, delete, start, and stop services and processes remotely. You can also view the status of both local and remote processes. If using Netcool/OMNIbus Administrator to manage process control, you can optionally save your changes to the configuration files of the process agents.

The process agents support the dynamic addition of routing statements from Netcool/OMNIbus Administrator. You can also add a new process agent to a routing group as follows:

1. Copy and modify the current configuration from an existing process agent to have visibility of the current processes.
2. Update the configuration files for each of the other process agents by adding the new routing entry to the routing definition area in the files.

3. Stop each of the existing process agents and their child processes by running the **nco_pa_shutdown** utility with the `-option STOP` setting. Then restart each process agent in order to pick up the new routing.

Note: Service and process names must be unique within the process control network.

Related tasks

“Creating and starting a process control network system” on page 252

“Defining processes, services, and hosts for process control” on page 264

“Displaying and configuring status information for a process agent” on page 283

Related reference

“Displaying the status of services and processes (nco_pa_status)” on page 273

“Adding a new service or process (nco_pa_addentry)” on page 277

Process control components

The process control system is available as an installable feature during the Tivoli Netcool/OMNIbus installation, and includes a number of standard and configurable components.

Process control consists of:

- Process agents and associated configuration files
- Processes
- Services, in which processes are organized and run
- Process control utilities to help you manage the process agents, processes, and services

Process agents

Process agents are programs that are installed on each host to manage processes in a process control system. Any participating host must have a process agent and an associated configuration file installed.

There can be any number of process agents on any number of hosts. Process agents can manage any number of processes.

Processes

Processes are programs that are run by a process agent on the same workstation, within a process control system. Processes must be defined in a *service*, for ease of management.

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 used. For example, ObjectServers, proxy servers, and ObjectServer 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 Tivoli Netcool/OMNIbus desktop is not PA aware.

Dependent processes

The order in which applications are started can be 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 agent. When the process reaches the point in its startup where it recognizes itself to be running, it sends a message to the process agent. When the process agent receives this message, it starts dependent processes.

Related tasks

“Defining processes, services, and hosts for process control” on page 264

Services

Within a process control system, processes must be grouped together in services. You can group related processes in a service to make them easier to manage.

After a service is correctly configured, it can be managed by process control.

You can configure a service to start automatically when the process agent starts. Alternatively, you can start a service manually.

A service can be configured either as a master service on which other services depend, or as a non-master service. When started automatically by process control, master services are started before non-master services.

Related tasks

“Defining processes, services, and hosts for process control” on page 264

Process control utilities

Command-line utilities are available to help you manage the process agents, processes, and services in a process control system.

The following table lists these command-line utilities.

Table 75. Process control command-line utilities

Utility	Description
nco_pa_status	This utility retrieves and displays the status of services and processes that are being controlled by the process agent.
nco_pa_start	This utility starts a service or process that is located anywhere in the configuration.
nco_pa_stop	This utility stops a service or process that is located anywhere in the configuration.
nco_pa_shutdown	This utility shuts down a process agent.
nco_pa_addentry	This utility adds a service or process entry while a process agent is running.

Related reference

“Displaying the status of services and processes (nco_pa_status)” on page 273

“Starting a service or process (nco_pa_start)” on page 275

“Stopping a service or process (nco_pa_stop)” on page 275

“Shutting down a process agent (nco_pa_shutdown)” on page 276

“Adding a new service or process (nco_pa_addentry)” on page 277

Creating and starting a process control network system

To manage process control, you must first determine your process control configuration requirements and then perform a number of configuration tasks.

A summary of the configuration tasks is as follows:

1. If you are using the default authentication mechanism on UNIX, set up a dedicated user group that you can use to control access to the process control system. Assign users to this group.
On Windows, access to the process control system is governed by the availability of a valid local or domain user account. Any account that is used to log into the computer on which a process agent is running can also be used to connect to the process agent, providing the user has the privilege Access this computer from the network.
2. For each process agent, configure server communication information (that is, a host name and port number) on the host computer and on every computer that needs to connect to the process control network. You must configure the server communication information in the Server Editor (or **nco_xigen**) before starting a process agent.
3. Update the default process agent configuration file for each process agent by defining processes, services, and hosts.

What to do next

When the configuration is complete, start the process agents. You can start a process agent either manually from the command line, or automatically, when the system starts. To start a process agent automatically, you can either use startup scripts on UNIX, or install and configure the process agent to run as a Windows service. Processes automatically run as defined within the configuration file for each process agent, and process agents communicate as configured.

Related tasks

“Configuring and managing process control from the command line” on page 263

Before you configure process control

Before you configure process control on one or more host computers, determine your process control configuration requirements.

The prerequisite tasks are as follows:

- Ensure that the process control function is installed. This function is installed when you select Process Control as an installable feature during the Tivoli Netcool/OMNIBus installation. Additionally, Netcool/OMNIBus Administrator must be installed if you want to manage process control from a graphical user interface. See the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide* for further information about installing process control and Netcool/OMNIBus Administrator.

- Determine which Tivoli Netcool/OMNIbus components are installed and where they are located. Ensure that you have taken into account all components and any failover or backup systems. Tivoli Netcool/OMNIbus desktops are not managed by process control.

Creating UNIX user groups for the process control system

The process control daemon controls who can log in to it. On UNIX, 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.

By default, the process control system uses UNIX user names and passwords to grant access. When running the process agent daemon (**nco_pad**), you can specify other supported authorization modes by using the **-authenticate** command-line option.

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**.

Attention: If using Pluggable Authentication Modules (PAM) for authentication, users do not have to be a member of a UNIX user group such as **ncoadmin**, to gain access to the process control system. With PAM clients, the process control system does not validate users against a UNIX user group, and, as a result, access is not restricted.

Related concepts

“Services” on page 251

Related reference

“Process agent command-line options” on page 256

Windows account requirements for the process control system

On Windows, the process agent must run as an account that is an Administrator on the local computer.

To connect to the process agent from Netcool/OMNIbus Administrator, another process agent, the ObjectServer, or a process control utility, you need one of the following account types:

- A local Windows user account
- A local Windows domain account
- An account in User Principal Name (UPN) format; that is, *username@DNS_domain_name*

For peer-to-peer connections between process agents, be wary of using password policies that lock out Windows accounts after a set number of attempts. Process agents repeatedly attempt to log in if they fail to connect; therefore, the use of a wrong password could cause your Windows accounts to be locked out very

quickly.

Configuring server communication information for process agents

You must use the Server Editor to assign a unique server name to each process agent and specify other communication information, and then make these details available to each host computer in the process control network system. This enables all process agents on all host computers to communicate with each other.

Perform the following actions to configure server communication information for process agents:

1. Start the Server Editor on a host computer and add a server entry for each process agent that you want to include in the process control network system. Save this information.

Tip: The name of a server entry must consist of 29 or fewer uppercase letters and cannot begin with an integer. The naming convention is to also append `_PA` to the name so that you can easily identify the server as a process agent in the Server Editor. For example, if you are configuring the process agent on a host named `sfosys1`, the process agent can be named `SFOSYS1_PA`. By default, the first process agent installed in a configuration is named `NCO_PA`.

2. On UNIX, generate an interfaces file that contains the server communication information. The interfaces file is typically named `$NCHOME/etc/interfaces.arch`, where *arch* is the UNIX operating system name.
In a Windows environment, configure server communications on each Windows computer.
3. On UNIX, distribute the updated interfaces file to all host workstations in the configuration.

What to do next

See the *IBM Tivoli Netcool/OMNIBus Installation and Deployment Guide* for further details on configuring server definitions in the Server Editor, and generating interfaces files.

Updating the default process control configuration file

A process control configuration file is installed for each process agent. This file contains definitions for each process, service, and host within the process control system configuration.

The process control configuration file `nco_pa.conf` is located in the `$NCHOME/omnibus/etc` directory.

Determine which processes should run under process control and identify process dependencies. Manually edit the process control configuration file to set up your process control definitions:

- Create service definitions to group together related or dependent processes. This determines the order in which processes are run.
- Create routing definitions to specify each process agent and its associated host computer that should be included in the configuration.

Results

When the process agent is started, it reads this file to establish configuration settings.

Related tasks

“Defining processes, services, and hosts for process control” on page 264

Manually starting process agents

You can manually start process agents from the command line.

To manually start a process agent, enter the following command on the command line of the host:

```
$NCHOME/omnibus/bin/nco_pad -name process_agent
```

In this command, *process_agent* is the name of the process agent, as defined in the *omni.dat* file (UNIX) or *sql.ini* file (Windows). You can specify additional command-line options with this command.

The process agent daemon (**nco_pad**) runs relative to the *\$NCHOME/omnibus* location.

UNIX The process agent daemon (**nco_pad**) follows UNIX-style quoting rules when deciding whether to expand environment variables inside arguments for external actions. The rules are as follows, in order of precedence:

- Double quotes (" ") inside a substring delimited by single quotes have no effect.
- Single quotes (' ') inside a substring delimited by double quotes have no effect.
- Environment variables inside single quotes are not be expanded.

See the section “Example” for more information.

Note: A new instance of the process agent cannot manage processes that were started by another instance, and which are still running. When the process 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.

Note: If you run the **nco_pad** command on a computer that already contains a process agent that is installed as a Windows service, any commands that are specified for the Windows service are merged with command-line options for running **nco_pad** from the command prompt. If there is a conflict between the options that are specified for the service, and the options that you enter at the command prompt, the options that you enter at the command prompt take precedence. The screen output displays the merged options.

Example

UNIX In the following example, only *\$B* is expanded, because *\$B* is outside of a pair of single quotes:

```
'$A'$B'$C'
```

The following example shows how you can avoid expansion but still output quotes. To obtain single quotes in the output, you must escape the string inside double quotes, as follows:

```
""'$A'""'$B'""'$C'""
```

This string produces the following output:

'\$A'\$B'\$C'

To obtain the output '\$A'\$B'\$C' with no expansion, it is not sufficient to place the string in double quotes because, according to the UNIX quoting rules, the string would be expanded. For no expansion to take place the whole string must be inside single quotes.

Process agent command-line options

When running the process agent with the **nco_pad** command, you can specify a number of command-line options for additional configuration.

The command-line options for the \$NCHOME/omnibus/bin/nco_pad command are described in the following table. UNIX and Linux-specific command-line options, which are not supported on Windows, are flagged in the table.

Table 76. Process agent daemon nco_pad command-line options

Command-line option	Description
<div>UNIX</div> <div>Linux</div> <div>-admingroup <i>string</i></div>	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 ncoadmin. Note: The -admingroup option is applicable only to the UNIX authentication mode.
-apicheck	If specified, Sybase API checking is enabled.

Table 76. Process agent daemon *nco_pad* command-line options (continued)

Command-line option	Description
<code>-authenticate <i>string</i></code>	<p>Specifies the authentication mode to use to verify the credentials of a user or remote process agent daemon.</p> <p>Note: When in FIPS 140-2 mode, only the PAM option can be specified for authentication.</p> <p>On UNIX, the values are:</p> <ul style="list-style-type: none"> • UNIX: This is the default authentication mode, 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 by using the <code>/etc/passwd</code> file, the <code>/etc/shadow</code> shadow password file, NIS, or NIS+. • PAM: If PAM 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. • KERBEROS: If KERBEROS is specified as the authentication mode, Kerberos IV authentication is used to verify user credentials. This is available only on Solaris systems with a Kerberos IV authentication server installed. • HPTCB: If HPTCB is specified as the authentication mode, the HP-UX password protection system is used. This is available only on HP trusted (secure) systems. <p>On Windows, the process agent authenticates against the Windows account. The only value for the <code>-authenticate</code> option is <code>WINDOWS</code>; this is also the default.</p>
<code>-configfile <i>string</i></code>	Use this file name, relative to <code>\$NCHOME/omnibus</code> , as the configuration file, rather than the default file <code>\$NCHOME/omnibus/etc/nco_pa.conf</code> .
<code>-connections <i>integer</i></code>	Sets the maximum number of connections that are available for running external actions. The default is 30.

Table 76. Process agent daemon *nco_pad* command-line options (continued)

Command-line option	Description
<code>-cryptalgorithm <i>string</i></code>	<p>Specifies the cryptographic algorithm to use for decrypting passwords that were encrypted with the nco_aes_crypt utility and then stored in the process agent configuration file. Set the <i>string</i> value as follows:</p> <ul style="list-style-type: none"> • When in FIPS 140–2 mode, use AES_FIPS. • When in non-FIPS 140–2 mode, you can use either AES_FIPS or AES. Use AES only if you need to maintain compatibility with passwords that were encrypted using the tools provided in versions earlier than Tivoli Netcool/OMNIBus V7.2.1. <p>The value that you specify must be identical to that used when you ran the nco_aes_crypt command with the <code>-c</code> setting, to encrypt the passwords in the routing definition section of the file.</p> <p>Use the <code>-cryptalgorithm</code> command-line option in conjunction with the <code>-keyfile</code> option.</p>
<code>-debug <i>integer</i></code>	<p>Enables debugging. The <i>integer</i> value 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.</p> <p>When running at debug level 1, the process agent logs information about processes it is about to start. This information includes the path to the program, each of the command-line arguments, and the effective user ID of the process. If applicable, the log also includes the current working directory, and additionally for UNIX, the effective group ID, and the umask (in octal) of the process.</p> <p>When running at debug level 2, the log contains a message showing which user account the process agent is running under.</p>
<code>-DNS <i>string</i></code>	<p>Specifies a value to override the host name in DNS environments. This must be the same as the entry in the configuration file.</p>
<code>-help</code>	<p>Displays help information about the process agent and exits.</p>
<code>-keyfile <i>string</i></code>	<p>Specifies the path and name of the file that contains the key to be used for decrypting the encrypted passwords that are stored in the process agent configuration file.</p> <p>The key file that you specify must be identical to that used when you ran the nco_aes_crypt utility with the <code>-k</code> setting, to encrypt the passwords in the routing definition section of the file.</p> <p>Use the <code>-keyfile</code> command-line option in conjunction with the <code>-cryptalgorithm</code> option to decrypt passwords.</p>

Table 76. Process agent daemon *nco_pad* command-line options (continued)

Command-line option	Description
<div>UNIX</div> <div>Linux</div> -killprocessgroup	<p>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.</p>
-logfile <i>string</i>	<p>Specifies an alternative log file. On UNIX, the log can be redirected to stderr and stdout. On Windows, the log is always written to a file.</p> <p>The default log file is:</p> <p><code>\$NCHOME/omnibus/log/pa_name.log</code></p> <p>Where <i>pa_name</i> is the name of the process agent specified with the <code>-name</code> option.</p>
-logsize <i>integer</i>	Specifies the maximum log file size in KB. The default is 1024 KB, and the minimum size is 16 KB.
-msgpoolsize <i>integer</i>	Specifies the number of messages that are available to the process control agent.
-name <i>string</i>	Specifies the name of the server for this process agent. If not specified, the default process agent name is <code>NCO_PA</code> .
-newlog	<p>This option is obsolete.</p> <p>The process agent always overwrites the previous log file.</p>
-noautostart	If specified, the process agent does not start any services automatically, even if they are set to start automatically in the <code>nco_pa.conf</code> file.
-noconfig	If specified, the process agent does not read the <code>nco_pa.conf</code> configuration file. This forces process control to start with no configuration information.
<div>UNIX</div> <div>Linux</div> -nodaemon	<p>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.</p>
-password <i>string</i>	Specifies the password that is used to log into other process agents.
<div>UNIX</div> <div>Linux</div> -pidfile <i>string</i>	<p>Specifies the path, relative to <code>\$NCHOME/omnibus</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>\$NCHOME/omnibus/var/pa_name.pid</code>, where <i>pa_name</i> represents the name of the process agent. Provided all process agents are given unique names, there should be no need to change this setting. This makes it possible to run more than one process agent daemon on the same computer.</p> <p>Tip: On Windows, there is no restriction on the number of process agents that can run as Windows services on the same host.</p>

Table 76. Process agent daemon nco_pad command-line options (continued)

Command-line option	Description
<div>UNIX</div> <div>Linux</div> <p><code>-pidmsgpool <i>integer</i></code></p>	Specifies the size of the signal-handling message pool.
<div>UNIX</div> <div>Linux</div> <p><code>-redirectfile <i>string</i></code></p>	<p>Specifies a file to which the stderr and stdout messages of processes started by the process agent are directed. This is useful for troubleshooting purposes.</p> <p>Tip: On Windows, a similar result can be achieved by running the process agent from the command line. Each child process will have its own console window in which the processing output is displayed. Alternatively, if the process agent is running as a Windows service, open the Services window, and specify the following settings in the Properties window for the service: from the Log On tab, select Local System account and then select Allow service to interact with desktop.</p>
<code>-retrytime <i>integer</i></code>	<p>Specifies the number of seconds that a process started by process control must run to be considered a successful start. The default <i>retrytime</i> is 5.</p> <p>The process agent attempts to restart a process if the process exits. If the process exits after <i>retrytime</i> seconds, the process agent attempts to restart the process immediately. If the process exits before <i>retrytime</i> 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 <i>retrytime</i> seconds, the RetryCount (specified in the process definition) for that process is also decremented. If the process runs successfully for at least <i>retrytime</i> seconds, the RetryCount is set back to its original value. If the RetryCount is 0, there is no limit to the number of restart attempts.</p>
<code>-roguetimeout <i>integer</i></code>	Specifies the time in seconds to wait for the process to shut down. The default is 30 seconds and the minimum is 5 seconds.
<code>-secure</code>	<p>If <code>-secure</code> is specified, all clients need to authenticate themselves with a valid user name and password, which are specified with the <code>-user</code> and <code>-password</code> command-line options.</p> <p>In non-FIPS 140-2 mode, login information is automatically encrypted in transmission when the process agent connects to another process agent. In FIPS 140-2 mode, login information is passed as plain text, and SSL must be used if you require encryption during transmission.</p>
<code>-stacksize <i>integer</i></code>	Specifies the size of the thread stack.

Table 76. Process agent daemon *nco_pad* command-line options (continued)

Command-line option	Description
<div>UNIX</div> <div>Linux</div> <code>-ticketdir <i>string</i></code>	Directory for Kerberos tickets if <code>-authenticate</code> is set to KERBEROS.
<code>-traceevtq</code>	Enables tracing of event queue activity.
<code>-tracemsgq</code>	Enables tracing of message queue activity.
<code>-tracemtx</code>	Enables the tracing of mutex locks.
<code>-tracenet</code>	Enables net library tracing.
<code>-user <i>string</i></code>	<p>Specifies the user name that is used to log into another process agent.</p> <p>If the <code>-user</code> option is not specified, the user name that is used to make the connection is the user running the command.</p> <p>This option must be specified if connecting to a process agent that is running in secure mode (using the <code>-secure</code> option).</p> <p>This user name and its associated password (which you specify using <code>-password</code>) are used if no login credentials are specified in the routing section of the process control configuration file.</p>
<code>-version</code>	Displays version information about the process agent and exits.

Automatically starting process agents on UNIX

On UNIX, startup scripts are available to automatically start the process agent when the system starts.

Before you begin

You can modify the startup scripts before you install them, if required.

On Linux operating systems, modify the startup script called **nco** for the version of Linux that you are using. The script contains sections for either the Red Hat and SUSE versions of Linux, and these sections are delimited with comments within the script as follows:

```
### REDHAT ONLY
...
### END REDHAT ONLY

### SUSE ONLY
...
### END SUSE ONLY
```

Note that the script contains a number of separate sections for the Red Hat and SUSE versions of Linux.

Also on Linux, modify the **nco** script if you are using Pluggable Authentication Modules (PAM) for authentication by adding the `-authentication PAM` argument to

the script. For the default UNIX authentication, you do not need to add anything.

These scripts are located in the following directory:

`$NCHOME/omnibus/install/startup`

This directory contains one of the following installation scripts, depending on the operating system:

- **`aix5install`**
- **`hpux11install`**
- **`hpux11hpiainstall`**
- **`solaris2install`**
- **`linux2x86install`**
- **`linux2s390install`**

To use the process agent startup scripts, you must run the appropriate installation script for your operating system. (You might need to make the script executable before running it.)

To install the process agent startup scripts:

1. Run the installation script as the root user. For example, to install the scripts on Solaris, run **`solaris2install`** from the `$NCHOME/omnibus/install/startup` location. The following output 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 output 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 message is displayed:

Enter value for environment variable NETCOOL_LICENSE_FILE,
if required [27000@localhost]:

Note: Although Tivoli Netcool/OMNIBus does not require a license key in order to run, some probes and gateways that have not been through a recent maintenance cycle still require license keys. If you are running these older probe or gateway packages, they will still require the `NETCOOL_LICENSE_FILE` environment variable to be set, and the availability of a Netcool license server.

4. If you do not use a license server, you can safely press Enter to run the script. If you have a license server, either press Enter to accept the default value for the licensing environment variable, or enter another value.

Results

Each installation 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.

What to do next

For information about modifying startup scripts, see your specific operating system documentation.

Automatically starting process agents on Windows

On Windows, you can install the process agent as a Windows service.

Use the Services window in the Control Panel to assign either of the following logon accounts to the service:

- Local system account (LocalSystem).
- An account that belongs to the Administrators group on the local computer.

For further information about installing and configuring a process agent as a Windows service, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

Managing your process control system configuration

After your process control system is set up and your process agents are running, you can choose to make changes to the configuration by running the process control utilities. Any configuration changes that you make apply to the current session only, and are not saved to the configuration file.

You can also use Netcool/OMNIbus Administrator to change the configuration. Configuration changes that you make from Netcool/OMNIbus Administrator can be saved to the configuration file.

Related concepts

“Using Netcool/OMNIbus Administrator to manage process control” on page 281

Related tasks

“Configuring and managing process control from the command line”

Configuring and managing process control from the command line

You can define processes, services, and hosts within the process control configuration file. You can also use command-line utilities to start, stop, and add a service or process, display the status of services and processes, and shut down a process agent.

Before you can manage process control using either of these facilities, you must have created your process control system configuration.

Related concepts

“Using Netcool/OMNIbus Administrator to manage process control” on page 281

Related tasks

“Creating and starting a process control network system” on page 252

Defining processes, services, and hosts for process control

To run under process control, processes, services, and hosts must be defined within a process agent configuration file. When the process agent starts, it reads this file to establish configuration settings.

The default process agent configuration file is:

```
$NCHOME/omnibus/etc/nco_pa.conf
```

The file is made up of definitions, each of which contains attributes and associated values, for each process, service, and host. The definitions are listed in the following order within the file:

1. Process definitions
2. Service definitions
3. Security definitions (optional)
4. Routing definitions

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 process 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 might contain user names and passwords.

Defining processes in the process agent configuration file

Within the process agent configuration file, you must define the list of processes that should be run by the process agents.

Process definition example

An example process definition in the \$NCHOME/omnibus/etc/nco_pa.conf configuration file is as follows:

```
nco_process 'ObjectServer'
{
  Command '$NCHOME/omnibus/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA' run as 0
  Host    = 'sfosys1'
  Managed = True
  RestartMsg = '${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 uses the preceding example to describe the process definition information that is contained in the configuration file.

Table 77. Process definition description

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 agent. If you use the same process name more than once, all, except for the first process definition, are ignored, and a warning message is generated.

Table 77. Process definition description (continued)

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>'\$NCHOME/omnibus/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA' run as 0</pre> <p>Or enter:</p> <pre>'\$NCHOME/omnibus/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA' run as 'root'</pre> <p>In this example:</p> <ul style="list-style-type: none"> • The <code>-pa</code> option specifies the process agent that the ObjectServer uses to run external automations. In this example, the process agent name is specified as <code>SFOSYS1_PA</code>. • The <code>run as</code> option instructs the host computer to run the ObjectServer as the specified user. On UNIX, you can either enter the user ID (typically <code>0</code>), or enter the user name enclosed in single quotation marks (typically <code>root</code>). When a user name is entered, the process agent looks up the user ID to use. If the process agent is not running as <code>root</code>, the <code>run as</code> option is ignored, and the process is run as the user who is running the process agent. <p>On Windows, all processes are run under the same user account as the process agent; always set the <code>run as</code> option to <code>0</code>.</p> <p>Tip: On Windows, you can use <code>%NCHOME%</code>, <code>\$NCHOME</code>, or the expanded form of the environment variable, in the path for the command. It is also acceptable to use slashes (<code>/</code>), backslashes (<code>\</code>), or double backslashes (<code>\\</code>) as separators.</p> <p>You can set the following additional process attributes by adding them to the beginning of the command string:</p> <ul style="list-style-type: none"> • CWD: Set the current working directory to the value specified. On Windows, you can specify the directory in any of these formats: MS-DOS format (for example, <code>C:\temp</code>), UNIX format (for example, <code>/temp/mydir</code>), and UNC format (for example, <code>\\server\share\mydir</code>). Both single and double backslashes can also be used as separators. When you run the process agent from the command line on UNIX and Windows, the working directory for all child processes is the directory from which the process agent was started. When you run the process agent as a UNIX daemon, the working directory for all child processes is <code>\$NCHOME/omnibus</code>. When you run the process agent as a Windows service, the default working directory for the process agent and for any child processes that are spawned by the process agent is <code>%NCHOME%\omnibus\log</code>. • SETGID: Set the group ID of the process to the value specified. This is a UNIX-specific attribute. • UMASK: Set the umask of the process to the value specified. This is a UNIX-specific attribute. <p>The format for specifying each of these attributes is as follows:</p> <pre>Command '[CWD=directory_path]commandpath options' run as user</pre> <pre>Command '[SETGID=groupID]commandpath options' run as user</pre> <pre>Command '[UMASK=permission]commandpath options' run as user</pre> <p>Note: You must specify the attributes as shown, in square brackets, without spaces.</p>

Table 77. Process definition description (continued)

Configuration information	Description
Command	<p>(continued from previous page)</p> <p>Examples (UNIX):</p> <p>Command '[CWD=/opt/netcool/]\$NCHOME/omnibus/bin/nco_objserv -name NCOMS2 -pa NCO_PA' run as 1253</p> <p>Command '[SETGID=ncoadmin]\$NCHOME/omnibus/bin/nco_objserv -name NCOMS2 -pa NCO_PA' run as 1253</p> <p>Command '[UMASK=u=rwx,g=rx,o=rx]\$NCHOME/omnibus/bin/nco_objserv -name NCOMS2 -pa NCO_PA' run as 1253</p> <p>Tip: In the preceding example with the UMASK setting, write permissions are assigned to the current user, but removed for all other users. You can alternatively specify this as [UMASK=022].</p> <p>Command '[UMASK=077]\$NCHOME/omnibus/bin/nco_objserv -name NCOMS2 -pa NCO_PA' run as 1253</p> <p>You can specify one or more of the attributes within the command string. For example:</p> <p>Command '[CWD=/tmp][SETGID=ncoadmin][UMASK=u=rwx,g=,o=]\$NCHOME/omnibus/bin/nco_objserv -name NCOMS2 -pa NCO_PA' run as 1253</p> <p>Example (Windows):</p> <p>Command '[CWD=C:\temp]%NCHOME%\omnibus\bin\nco_objserv -name NCOMS2 -pa NCO_PA' run as 0</p>
Host	The name of the host on which the process should be run. Process control automatically resolves the name of the process agent when required.
Managed	<p>Can have either of these values:</p> <ul style="list-style-type: none"> • True: The process is restarted automatically if it exits. • False: The process is not restarted automatically if it exits.
RestartMsg	Contains the message to be sent to the UNIX syslog or the Windows Event Viewer if the process is restarted. For example, The ObjectServer has been restarted.
AlertMsg	Contains the message to be sent to the UNIX syslog or the Windows Event Viewer if the process exits. For example, The ObjectServer has gone down.
RetryCount	Specifies the number of restart attempts to be made if the process exits in the time specified by the nco_pad -retrytime 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 PaPA_AWARE for PA aware processes and PaNOT_PA_AWARE for processes that are not PA aware.

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 the following table.

Table 78. Expansion keywords

Expansion keyword	Description
\${NAME}	The name of the process.
\${HOST}	The name of the host running the process.
\${EUID}	The effective user ID under which the process is running.
\${COMMAND}	The command that defines the process.

Alert and restart syslog or Event Viewer messages

When an alert or restart message is generated by the process agent daemon `nco_pad`, it is passed to the UNIX syslog or the Windows Event Viewer. Tivoli 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 that is available on the Tivoli Network Management Information Center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp>.

The alert and restart messages are sent to the UNIX syslog or the Windows Event Viewer 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 that is defined in the configuration file for that process.

Defining services in the process agent configuration file

Within the process agent configuration file, you can define services to group together related processes, and configure interdependencies of processes. The processes must already be defined in the list of processes within the file.

Service definition example

An example service definition in the `$NCHOME/omnibus/etc/nco_pa.conf` configuration file is as follows:

```
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 uses the preceding example to describe the service definition information contained in the configuration file.

Table 79. Service definition description

Configuration information	Description
nco_service 'Omnibus'	Defines the name of the service (for example, Omnibus). Note: Each service name must be unique within the process control network.
ServiceType	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 Master or Non-Master.
ServiceStart	This can be set to Auto to start the service as soon as nco_pad has started, and Non-Auto if the service must be started manually with the nco_pa_start command.
process	Each process entry 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. Note: You must include a process only once within any of the service definitions in the configuration file.

Specifying process dependencies

When defining a service, you can use the process attribute 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. A time dependency is always measured from the start of the service. 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.

Restriction: A process cannot be dependent on another process that is time-dependent. If you specify a dependency on a time-dependent process, an error message is added to the process control log file, and the dependent process and any child processes are assigned a status of DEAD. The default log file is \$NCHOME/omnibus/log/pa_name.log, where *pa_name* is the name of the process agent.

The *dependency* type NONE specifies no dependency.

In the preceding service definition example for the Omnibus service, the ObjectServer process starts first because it has no dependencies. Five seconds after the ObjectServer starts, the Sleep process starts. When the ObjectServer is running successfully, Proxy and Probe-1 start. When the proxy server is running, the Probe process starts. If any of the processes was specified as dependent on the time-dependent Sleep process, that process does not start, and is assigned a status of DEAD.

Related reference

“Defining processes in the process agent configuration file” on page 264

Defining secure hosts in the process agent configuration file

You can specify that only certain hosts can connect to process agents by adding a security definition to the process agent configuration file. If you do not create a security definition, any process can connect from any host.

Within the `$NCHOME/omnibus/etc/nco_pa.conf` configuration file, the security definition is inserted between the service definitions and the routing definitions for hosts. You can create a security definition with no hosts specified, as follows:

```
nco_security
{
}
```

When no hosts are specified, only processes that are running on the current host or on any host listed in the routing definition can connect.

Processes running on hosts that are not listed in the routing definition can connect only 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. The process agent 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 must 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 IPv4 address in dotted decimal notation
- An IPv6 address in full, shorthand, or mixed colon-separated notation

An IPv4 address in dotted decimal notation can contain the following wildcards:

- `?` matches one character
- `*` matches any number of characters

You can append `/n` to a specified IPv6 address, where *n* is a number, to represent IPv6 addresses for which the first *n* bits match the stated IP address.

Security definition example

The following example security definition allows connections from processes on the following hosts:

- `alpha`
- `192.9.200.34`
- Any host on the subnet `193.42.52.0`

- Any host with an IPv6 address where the first 10 bits match fe80::203:baff:fe2a:6bf0
- fe80::203:baff:fe2a:6bf0

```
nco_security
{
  host 'alpha'
  host '192.9.200.34'
  host '193.42.52.*'
  host 'fe80::203:baff:fe2a:6bf0/10'
  host 'fe80::203:baff:fe2a:6bf0'
}
```

Defining routing hosts in the process agent configuration file

To specify the hosts that are participating in the process control system, you must define the process agent host names in the process agent configuration file.

Each host entry 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). For each host definition, you can also specify user name and password credentials for connecting to the process agent.

Routing definition example

An example routing definition in the \$NCHOME/omnibus/etc/nco_pa.conf configuration file is as follows:

```
nco_routing
{
  host 'sfosys1' 'SFOSYS1_PA' 'username' 'password'
  host 'sfosys2' 'SFOSYS2_PA' 'username' 'password'
}
```

Note: The *username* and *password* entries are mandatory if you are running the remote process agent in secure mode. If you are not running the remote process agent in secure mode, user names and passwords are optional.

If the process agent is using UNIX authentication (the default on UNIX), the *username* must be an operating system user that is a member of the ncoadmin group (default) or any other administrative group that is created for granting access to the process control system. A process agent daemon that is running in secure mode must be run by the root user.

On Windows, *username* must be the user name of a valid local account, domain account, or UPN account.

Note: To prevent unauthorized users from gaining access, operating system security must be set appropriately for files that contain user names and passwords.

When running the process agent daemon **nco_pad**, you can also specify the user name and password by using the **-user** and **-password** command-line options. This overrides any entries in the nco_pa.conf configuration file.

Encrypting plain text passwords in routing definitions

You can encrypt plain text login passwords that are stored in the nco_pa.conf file.

Password encryption details for running in FIPS 140–2 mode and non-FIPS 140–2 mode are described in the following table.

Table 80. Password encryption in FIPS 140–2 mode and non-FIPS 140–2 mode

Mode	Action
FIPS 140–2 mode	<p>When in FIPS 140–2 mode, passwords can either be specified in plain text or in encrypted format. You can encrypt passwords by using property value encryption, as follows:</p> <ol style="list-style-type: none"> 1. If you do not yet have a key for encrypting the password, create one by running the nco_keygen utility, which is located in <code>\$NCHOME/omnibus/bin</code>. 2. Run the nco_aes_crypt utility to encrypt the password with the key that was generated by the nco_keygen utility. The nco_aes_crypt utility is also located in <code>\$NCHOME/omnibus/bin</code>. Note that you must specify <code>AES_FIPS</code> as the algorithm to use for encrypting the password. 3. Copy the encrypted password into the appropriate routing definition in the configuration file.
Non-FIPS 140–2 mode	<p>In non-FIPS 140–2 mode, you can either use the nco_pa_crypt utility or use property value encryption to encrypt plain text login passwords on UNIX. On Windows, you can use the nco_g_crypt utility or use property value encryption. Perform either of the following actions:</p> <ul style="list-style-type: none"> • To encrypt a password by using the nco_pa_crypt or nco_g_crypt utility, run the command as follows: <ul style="list-style-type: none"> – UNIX: <pre>\$NCHOME/omnibus/bin/nco_pa_crypt plaintext_password</pre> – Windows: <pre>%NCHOME%\omnibus\bin\nco_g_crypt plaintext_password</pre> <p>In these commands, <i>plaintext_password</i> represents the unencrypted form of the password. The encryption utility displays an encrypted version of the password. Copy the encrypted password into the appropriate routing definition in the configuration file.</p> • To encrypt a password by using property value encryption, you require a key that is generated with the nco_keygen utility. You can then run nco_aes_crypt to encrypt the password with the key. Note that you can specify either <code>AES_FIPS</code> or <code>AES</code> as the algorithm for encrypting the password. Use <code>AES</code> only if you need to maintain compatibility with passwords that were encrypted using the tools provided in versions earlier than Tivoli Netcool/OMNIBUS V7.2.1. Copy the encrypted password into the appropriate routing definition in the configuration file.

Note: On UNIX, even if the password is specified on the command line, it does not appear in **ps** command output.

Passwords that are encrypted using **nco_pa_crypt** are decrypted by the remote process control agent.

Passwords that are encrypted using **nco_aes_crypt** are decrypted by the process agent daemon, and are passed to remote process agents as plain text. To decrypt the passwords, you must set the `-cryptalgorithm` and `-keyfile` command-line options when running **nco_pad**. These options specify which algorithm and key file to use for decryption.

For further information about using property value encryption, see the *IBM Tivoli Netcool/OMNIBUS Installation and Deployment Guide*.

Related tasks

“Creating UNIX user groups for the process control system” on page 253

Related reference

“Process agent command-line options” on page 256

Sample: Process agent configuration file

This sample shows the contents of a process agent configuration file
\$NCHOME/omnibus/etc/nco_pa.conf.

```
#####  
#NCO_PA3#  
# Process Agent Daemon Configuration File 1.1  
#  
# Ident: $Id: nco_pa.conf 1.3 2002/05/21 15:28:10  
#  
#  
# List of processes  
#  
nco_process 'NO1_PROXY_ProxyServer'  
{  
  Command '$NCHOME/omnibus/bin/nco_proxyserv -name NETPROXY -server NETOPS1' 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 ${HOST}.'  
  RetryCount = 0  
  ProcessType = PaPA_AWARE  
}  
nco_process 'SFOSYS_ObjectServer'  
{  
  Command '$NCHOME/omnibus/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 ${HOST}.'  
  RetryCount = 0  
  ProcessType = PaPA_AWARE  
}  
nco_process 'Syslog_Probe'  
{  
  Command '$NCHOME/omnibus/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 ${HOST}.'  
  RetryCount = 0  
  ProcessType = PaNOT_PA_AWARE  
}  
nco_process 'Mtttrapd_Probe'  
{  
  Command '$NCHOME/omnibus/probes/nco_p_mtttrapd' 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 ${HOST}.'  
  RetryCount = 0  
  ProcessType = PaNOT_PA_AWARE  
}  
nco_process 'MyScript'  
{  
  Command '$HOME/myscript.sh' run as 0  
  Host = 'objser1'  
  Managed = False  
  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  
}  
#  
# List of Services  
#  
nco_service 'Core'  
{  
  ServiceType = Master  
  ServiceStart = Auto  
  process 'MyScript' NONE  
}
```

```
# ObjectServer started after 20 seconds to allow the script to finish
process 'SFOSYS_ObjectServer' 20
# Proxy server started after the ObjectServer starts
process 'N01_PROXY_ProxyServer' 'SFOSYS_ObjectServer'
# Trapd probe and then Syslog probe started after the proxy server starts
process 'Mtrapid_Probe' 'N01_PROXY_ProxyServer'
process 'Syslog_Probe' 'N01_PROXY_ProxyServer'
}
#
# ROUTING TABLE
#
# 'user' - (optional) only required for secure mode PAD on target host
# 'user' must be member of a UNIX administrative group if using UNIX authentication
# On Windows, 'user' must be the user name of a valid local, domain, or UPN account
# 'password' - (optional) only required for secure mode PAD on target host
# can be plain text, or encrypted using encryption tool provided for specific
# platform and security requirements
nco_routing
{
  host 'objser1' 'OBJSER1_PA'
}
```

Managing process control using the process control utilities

The process control system provides command-line utilities for managing and changing the Tivoli Netcool/OMNIBus configuration. You can use these utilities to start, stop, and add a service or process, display the status of services and processes, and shut down a process agent.

The command-line utilities are as follows:

- **nco_pa_status**
- **nco_pa_start**
- **nco_pa_stop**
- **nco_pa_shutdown**
- **nco_pa_addentry**

Each utility prompts for your password.

Displaying the status of services and processes (nco_pa_status)

You can run the **nco_pa_status** utility to retrieve the status of services in the process control system configuration. For each service, the **nco_pa_status** utility returns a list of defined processes, the status of each process, and the process identifier.

To display the service status, enter the following command:

```
$NCHOME/omnibus/bin/nco_pa_status -server string
```

In this command, *string* is the process agent name. You can optionally run the command with additional command-line options.

Example output is as follows:

```
-----
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
-----
```

The PID value for managed processes is the UNIX process identifier, or the PID as shown in the Windows Task Manager.

The following table describes each of the status levels.

Table 81. Service status descriptions

Status level	Description
RUNNING	The process is running.
STARTING	A start request has been issued.
PENDING	The process is waiting for a time dependency to complete. This status can also indicate that the process has failed to start properly (regardless of any process dependencies).
WAITING	The process is waiting for a dependency to start.
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 by using the **nco_pa_start** utility.

Command-line options for nco_pa_status

Command-line options for the **nco_pa_status** utility are described in the following table.

Table 82. Command-line options for nco_pa_status

Command-line option	Description
-help	Displays help on the command-line options and exits.
-nosecure	Connects to process agents in a non-secure mode that does not encrypt login information during transmission.
-password <i>string</i>	The password to use for the process agent.
-server <i>string</i>	Name of process agent to contact.
-user <i>string</i>	The user name for the process agent. The default is the user running the command.
-version	Displays software version information and exits.

Related reference

“Starting a service or process (nco_pa_start)” on page 275

Starting a service or process (nco_pa_start)

You can run the **nco_pa_start** utility to start a service or process at any location in the process control system configuration.

If the service or process has already been started, the command is ignored.

To start a service or process, enter the command:

```
$NCHOME/omnibus/bin/nco_pa_start command_line_options
```

In this command, *command_line_options* represents one or more command-line options that you can specify for the **nco_pa_start** utility. You can specify only a single service or process.

Command-line options for nco_pa_start

Command-line options for the **nco_pa_start** utility are described in the following table.

Table 83. Command-line options for nco_pa_start

Command-line option	Description
-help	Displays help about the command-line options and exits.
-nosecure	Connects to process agents in a non-secure mode that does not encrypt login information during transmission.
-password <i>string</i>	The password to use for the process agent.
-process <i>string</i>	Name of the process to start.
-server <i>string</i>	Name of process agent to contact.
-service <i>string</i>	Name of the service to start.
-user <i>string</i>	The user name for the process agent. The default is the user running the command.
-version	Displays software version information and exits.

Stopping a service or process (nco_pa_stop)

You can run the **nco_pa_stop** utility to stop a service or process at any location in the process control system configuration.

If the service or process has already been stopped, the command is ignored.

When you stop a service, all the processes that are defined within that service are also stopped. When you stop a process, the status of dependent processes remains unaltered; for example, if an ObjectServer is stopped, the probes that were dependent on it continue to run.

To stop a service or process, enter the following command:

```
$NCHOME/omnibus/bin/nco_pa_stop command_line_options
```

In this command, *command_line_options* represents one or more command-line options that you can specify for the **nco_pa_stop** utility. You can specify only a single service or process.

Command-line options for nco_pa_stop

Command-line options for the **nco_pa_stop** utility are described in the following table.

Table 84. Command-line options for nco_pa_stop

Command-line option	Description
-force	If specified, no warning is output if the process or service is not running.
-help	Displays help about the command-line options and exits.
-nosecure	Connects to process agents in a non-secure mode that does not encrypt login information during transmission.
-password <i>string</i>	The password to use for the process agent.
-process <i>string</i>	Name of the process to stop.
-server <i>string</i>	Name of process 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	Displays software version information and exits.

Shutting down a process agent (nco_pa_shutdown)

You can run the **nco_pa_shutdown** utility to shut down a process agent and optionally stop associated services and processes.

To shut down a process agent, enter the following command:

```
$NCHOME/omnibus/bin/nco_pa_shutdown command_line_options
```

In this command, *command_line_options* represents one or more command-line options that you can specify for the **nco_pa_shutdown** utility.

Command-line options for nco_pa_shutdown

Command-line options for the **nco_pa_shutdown** utility are described in the following table.

Table 85. Command-line options for nco_pa_shutdown

Command-line option	Description
-help	Displays help about the command-line options and exits.
-nosecure	Connects to process agents in a non-secure mode that does not encrypt login information during transmission.

Table 85. Command-line options for `nco_pa_shutdown` (continued)

Command-line option	Description
<code>-option string</code>	Specifies how the shutdown is completed. Can be <code>STOP</code> to shut down all processes that the process agent manages locally, or <code>LEAVE</code> to leave the locally-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 you for the type of shutdown to perform. Tip: To stop a remotely-managed process, you must run the <code>nco_pa_stop</code> utility.
<code>-password string</code>	The password to use for the process agent.
<code>-server string</code>	Name of process agent to shut down.
<code>-user string</code>	The user name for the process agent. The default is the user running the command.
<code>-version</code>	Displays software version information and exits.

Tip: If you are running a process agent as a Windows service, and you stop the service, managed processes are also stopped. This means that all managed processes are stopped in a controlled way when the system is shut down.

Related reference

“Stopping a service or process (`nco_pa_stop`)” on page 275

Adding a new service or process (`nco_pa_addentry`)

You can run the `nco_pa_addentry` utility to add a new service or process while the process agent is running.

Use this utility to:

- Add a new service to a running process agent.
- Start a *fire-and-forget* process. Such processes start automatically and cannot be altered. When setting up a fire-and-forget process, define the process as unmanaged if you want to ensure that it only runs once. You can do this by specifying the `-unmanaged` command-line option when running the `nco_pa_addentry` utility.
- Add a new managed process to a service.

Note: The new service or process is not added to the process agent configuration file unless you choose to update the configuration file when using Netcool/OMNIBus Administrator.

To add a service or process to a running process agent, enter the following command. The square brackets depict optional entries.

```
$NCHOME/omnibus/bin/nco_pa_addentry [-process string | -service string]
command_line_options
```

In this command, *command_line_options* represents one or more command-line options that you can specify for the service or process, or for the `nco_pa_addentry` utility.

Command-line options for nco_pa_addentry

Command-line options for the **nco_pa_addentry** utility are described in the following table.

Important: To ensure that the process or service is properly created, all relevant command-line options must be explicitly specified with an assigned *string* value. This also applies to any default settings that you might want to apply.

Table 86. Command-line options for nco_pa_addentry

Command-line option	Description
-alert_msg <i>string</i>	Specifies the message to send to the UNIX syslog or the Windows Event Viewer if the process exits. On UNIX, enclose the <i>string</i> value in single quotation marks if the text contains spaces. On Windows, enclose the <i>string</i> value in double quotation marks if the text contains spaces.
-auto -nonauto	If -auto is specified, the service or process is started as soon as the process agent is started. By default, the service must be started manually with the nco_pa_start command.
-command <i>string</i>	Specifies the process command line. For example: \$NCHOME/omnibus/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA
-delay <i>string</i>	Specifies the time delay in seconds before the specified process is started.
-depend <i>string</i>	Specifies a process on which the specified process depends.
-help	Displays help on the command-line options and exits.
-host <i>string</i>	Specifies the host on which to run the process.
-managed -unmanaged	If -managed is specified, the process is restarted automatically if it exits. The default is -managed.
-master -nonmaster	If -master is specified, the service type is set to master. The default is -master.
-nosecure	Connects to process agents in a non-secure mode that does not encrypt login information during transmission.
-pa_aware -not_pa_aware	If -pa_aware is specified, the ProcessType is set to PaPA_AWARE. By default, the process is not PA aware.
-parentservice <i>string</i>	Specifies the service to which to add the process. Note: When adding a process to a service, you <i>must</i> define the parent service by using -parentservice.
-password <i>string</i>	Specifies the password to use when connecting to the process agent.
-process <i>string</i>	Specifies the name of the process to add.

Table 86. Command-line options for `nco_pa_addentry` (continued)

Command-line option	Description
<code>-restart_msg string</code>	Specifies the message to send to the UNIX syslog or the Windows Event Viewer if the process is restarted. On UNIX, enclose the <i>string</i> value in single quotation marks if the text contains spaces. On Windows, enclose the <i>string</i> value in double quotation marks if the text contains spaces.
<code>-retrycount integer</code>	Specifies the number of restart attempts to be made if the process exits in the time specified by the <code>nco_pad -retrytime</code> command-line option. If set to 0, there is no limit to the number of restart attempts. The default is 0.
<code>-runas integer</code>	Specifies the user ID to run the process as.
<code>-server string</code>	Specifies the name of the process agent. The default is <code>NCO_PA</code> .
<code>-service string</code>	Specifies the name of the service to add.
<code>-user string</code>	Specifies the user name to use when connecting to the process agent. The default is the user running the command.
<code>-version</code>	Displays software version information and exits.

Example: Using `nco_pa_addentry` to add a fire-and-forget process (UNIX)

1. Enter the following command to add a fire-and-forget process named `simnet1`, which starts automatically and runs only once (as an unmanaged process):

```
./nco_pa_addentry -server TEST_PA -process 'simnet1' -command
'$NCHOME/omnibus/probes/nco_p_simnet' -host 'owl' -retrycount 0
-unmanaged -restart_msg 'test' -alert_msg 'testalert'
```

2. Run the `nco_pa_status` utility to retrieve the status of services in the configuration:

```
$NCHOME/omnibus/bin/nco_pa_status -server TEST_PA
```

Where `TEST_PA` is the process agent name.

When the `nco_pa_status` utility is run, the output will not show a `simnet1` process definition as part of a service entry. However, the `ps -ef` command will show the `simnet1` process as running, although it will not automatically restart if it exits.

Example: Using `nco_pa_addentry` to add a fire-and-forget process (Windows)

1. Enter the following command to add a fire-and-forget process named `simnet1`, which starts automatically and runs only once (as an unmanaged process):

```
"%NCHOME%" \omnibus\bin\nco_pa_addentry -server TEST_PA -process simnet1
-command "%NCHOME%" \omnibus\probes\win32\nco_p_simnet -host owl
-retrycount 0 -unmanaged -restart_msg "Probe restarted" -alert_msg
"Probe stopped" -password secret
```

2. Run the `nco_pa_status` utility to retrieve the status of services in the configuration:

```
"%NCHOME%" \omnibus\bin\nco_pa_status -server TEST_PA
```

Where `TEST_PA` is the process agent name.

When the **nco_pa_status** utility is run, the output will not show a simnet1 process definition as part of a service entry. However, the Windows Task Manager will show the simnet1 process as running, although it will not automatically restart if it exits.

Example: Using **nco_pa_addentry** to add a managed process to a service (UNIX)

1. Enter the following command to add a process named simnet2 to a Core service:

```
./nco_pa_addentry -server TEST_PA -process 'simnet2' -command '$NCHOME/omnibus/probes/nco_p_simnet' -host 'owl' -retrycount 0 -managed -restart_msg 'test' -alert_msg 'testalert' -parents-service 'Core'
```
2. Run the **nco_pa_status** utility to retrieve the status of services in the configuration:

```
$NCHOME/omnibus/bin/nco_pa_status -server TEST_PA
```

Where TEST_PA is the process agent name.

When **nco_pa_status** is run, the output will display a simnet2 process with a status of DEAD, as part of the Core service definition. The simnet2 process will not start automatically because it is part of a service, and must be run using the **nco_pa_start** utility.

Example: Using **nco_pa_addentry** to add a managed process to a service (Windows)

1. Enter the following command to add a process named simnet2 to a Core service:

```
"%NCHOME%" \omnibus\bin\nco_pa_addentry -server TEST_PA -process simnet2 -command "%NCHOME%" \omnibus\probes\win32\nco_p_simnet -host owl -retrycount 0 -managed -restart_msg "test" -alert_msg "testalert" -parents-service "Core"
```
2. Run the **nco_pa_status** utility to retrieve the status of services in the configuration:

```
"%NCHOME%" \omnibus\bin\nco_pa_status -server TEST_PA
```

Where TEST_PA is the process agent name.

When **nco_pa_status** is run, the output will display a simnet2 process with a status of DEAD, as part of the Core service definition. The simnet2 process will not start automatically because it is part of a service, and must be run using the **nco_pa_start** utility.

Related reference

"Starting a service or process (nco_pa_start)" on page 275

Using Netcool/OMNIbus Administrator to manage process control

Netcool/OMNIbus Administrator provides a visual interface from which you can manage process control. You can use Netcool/OMNIbus Administrator to view and manage process agents, processes, and services on your Tivoli Netcool/OMNIbus hosts.

For example, you can view the status of services that are under process control on a host computer, and then start or stop the processes in those services.

You must connect to a process agent in order to manage its services and processes. Configuration changes that you make to services and processes can be saved to the process control configuration file, which is overwritten each time that you save.

Note: You cannot use Netcool/OMNIbus Administrator to specify that only certain hosts can connect to process agents. To define such secure hosts, you must add a security definition to the process agent configuration file manually.

Related tasks

“Configuring and managing process control from the command line” on page 263

“Starting Netcool/OMNIbus Administrator” on page 60

Connecting to a process agent

Before you can connect to a process agent by using Netcool/OMNIbus Administrator, you must ensure that the process agent has started.

You can start the process agent automatically by using the supplied startup scripts on UNIX, or by running the process agent as a Windows service. You can also start the process agent manually by running the `$NCHOME/omnibus/bin/nco_pad` command.

To connect to a process agent:

1. From the Netcool/OMNIbus Administrator window, select the **Reports** menu button.
2. Click the **PA** icon. The Process Agent Report window opens. This window displays all the process agents that were selected when the Import Connections Wizard was last run.

Tip: After you have started Netcool/OMNIbus Administrator, you can select **File > Import** at any time to import new server communication information that is specified in the Server Editor. This information facilitates communication between Tivoli Netcool/OMNIbus server components such as ObjectServers, gateways, process agents, and proxy servers.

3. Select the process agent to which you want to connect and then perform either of the following actions:
 - If you are connecting for the first time or want to enter updated authentication information to be used when connecting, click **Connect As** in the toolbar. The Process Agent Security window opens. Go to step 4.
 - If you want to connect by using previously-specified authentication information, click **Connect**. The Process Agent Security window opens with the previously-specified authentication details. Go to step 5 on page 282.
4. Complete the Process Agent Security window as follows:

Username

Type the user name that is used to log into the process agent.

On UNIX, any user that needs access to the process agent must be a member of a UNIX user group that you identify as an administration group for this purpose. On Windows, the user must be a valid user with a local or domain account.

Password

Type the password that is used to log into the process agent.

Always use for this connection

Select this check box to indicate that the specified user name and password should be saved for automatic reuse on subsequent connection attempts to this process agent. These settings last for the length of the application session.

Use as default

Select this check box if you want the values specified for the user name and password to be automatically filled in the next time this window is displayed. These settings last for the length of the application session.

Note: If you select both check boxes, the **Always use for this connection** setting takes precedence.

5. Verify or cancel the authentication as follows:

OK Click this button to verify the credentials for connection, and close the window. The Service/Process Details pane opens. This pane contains information about the processes and services that are configured for the selected process agent.

Cancel

Click this button to close the Process Agent Security window without attempting to connect to the process agent. You return to the Process Agent Report window.

What to do next

When you first connect to existing process agents with Netcool/OMNIbus Administrator, the remote processes are only visible if they have been defined manually in the configuration of the local process agent. To make remote processes visible to all linked process agents, select each service or process in turn, click the **Edit** button, and close the resulting window without making any changes. Save each configuration file after that process is complete.

Related tasks

“Automatically starting process agents on UNIX” on page 261

“Automatically starting process agents on Windows” on page 263

“Manually starting process agents” on page 255

“Starting Netcool/OMNIbus Administrator” on page 60

Displaying and configuring status information for a process agent

You can view version details for a process agent to which you are connected, and change the logging level for messages that the process agent generates. You can also configure host routing by adding process agents to a routing group.

Before you begin

Before attempting to add a process agent to a routing group, you must have used the **File > Import** option (and the Import Connections Wizard) to import the details of the process agent into Netcool/OMNIbus Administrator.

To display and configure status information for a process agent:

1. If not already connected, connect to the process agent by using Netcool/OMNIbus Administrator. On successful connection, the Service/Process Details pane opens by default.
2. Click the **Info** icon that is displayed to the left of this pane. The PA Status Information pane opens.
3. Complete this pane as follows:

Version

This area displays the version of the process agent.

Note: The version cannot be determined if the process agent is a version that is earlier than V7.0.1.

Debug Level

To change the debug level for the log file, select another value from this drop-down list. Click the tick button to the right of the drop-down list to apply the selected debug level to the current session.

To save the updated debug level, click **Save the Process Agent configuration** in the toolbar.

Defined Hosts (UNIX)/Related Process Agents (Windows)

This area displays a list of process agents that are known to this process agent. The information is read from the process agent configuration file, which may differ from the hosts that are imported from the interfaces file by the Import Connections Wizard.

You can use the buttons to the right to add from a list of known hosts, or to remove hosts.

To save your host definition changes, click **Save the Process Agent configuration** in the toolbar. If you do not save in this way, your changes are lost on exit.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Displaying the processes and services for a process agent

You can use the Service/Process Details pane in Netcool/OMNIBus Administrator to view details of the processes and services that are configured for a process agent, and to manage these processes and services.



To view the processes and services that are configured for a process agent, perform either of the following actions:

- If not already connected, connect to the process agent by using Netcool/OMNIBus Administrator. On successful connection, the Service/Process Details pane opens by default.
- If you are already connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.

Results

In the Service/Process Details pane, the following details are shown for each configured service and process:

- The unique name that is assigned to the service or process
- The status of the service or process
- The host on which the process is running; this column is blank for services
- The process identifier of a running process; this column is blank for services and processes that are not currently running

Within the **Name** column, the  icon shown to the left of a name identifies the entry as a service, and the  icon identifies the entry as a process. Processes are also grouped by the service under which they run, and process names are shown in the following format:

service_name:process_name

For example: Core:MasterObjectServer

Within the **Status** column, the status icon is depicted as a circle, and its color indicates whether the service or process is running:

- Green: The service or process is running.
- Blue: The service is marginal. Not all processes are running.
- Yellow: The process is pending. The process is waiting for a time dependency to complete. This status can also indicate that the process has failed to start properly, regardless of any process dependencies.
- Gray: The process is dead (not running) or the service has stopped.
- Red: This identifies an error status level, which is an indication that a status level cannot be retrieved from the process agent.

From the Service/Process Details pane, you can manage processes and services for the selected process agent in the following ways:

- Create or edit a service
- Create or edit a process
- Delete a selected service or process
- Start a selected service or process
- Stop a selected service or process

- Copy and paste a service or process within the same process agent, or across process agent hosts
- Stop the process agent
- Run an external action
- Send a signal to a process
- Save the process agent configuration file to disk
- Refresh the contents of the pane (by clicking **Refresh** in the toolbar)

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Configuring services for a process agent

You can use Netcool/OMNIbus Administrator to create, edit, delete, start, and stop services that are defined to run under a process agent. When you make configuration changes to a service, you can elect to save your changes to the process control configuration file.

Creating and editing services

You can set up a Tivoli Netcool/OMNIbus service and configure it to start either automatically or manually.

To create or edit a service:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. To add a service, click **New Service** in the toolbar. To edit a service, select the service to edit and then click **Edit** in the toolbar.

The Service Details window opens.

3. Complete this window as follows:

Name Enter the service name. This name must be unique within the process control network. When editing a service, you cannot change the name.

Auto Start Service

Select this check box to specify that the service should start automatically as soon as the process agent has started. Clear this check box if you want to start the service manually by using the **nco_pa_start** command.

Master Service

Select this check box to indicate that this is a master service. A master service is started before other services and is handled as the master service upon which other services depend.

If you define multiple services as master within the same process control configuration file, the master services start in the order in which they appear in the configuration file.

Processes and their order within the configuration file

This list is shown only when editing a service, and shows the processes that are defined to run as part of the service. The order in which the processes are displayed indicates their order in the configuration file. You can use the arrow buttons to change the process order.

4. Save or cancel your changes as follows:

OK Click this button to save the service details and close the window. New services are added to the Service/Process Details pane.

Tip: Your configuration changes are not automatically saved to the process control configuration file. To update the file, click **Write Process Agent Config file** in the toolbar.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Deleting a service

When you delete a service, you remove it from the process control configuration.

To delete a service:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Select the service that you want to delete, click **Delete** in the toolbar, and confirm the deletion. The service is deleted and the Service/Process Details pane is updated.

Results

Tip: Your configuration changes are not automatically saved to the process control configuration file. To update the file, click **Write Process Agent Config File** in the toolbar.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Starting a service

You can only start services that have a Stopped status.

To start a service:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Select the service that you want to start and then click **Start** in the toolbar.

Results

The service and its associated processes are started and the Service/Process Details pane is updated.

Related tasks


“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Stopping a service

You can only stop services that have a Marginal or Running status.

To stop a service:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Select the service that you want to stop and then click **Stop**  in the toolbar.

Results

The service and its defined processes are stopped and the Service/Process Details pane is updated.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Configuring processes

Using Netcool/OMNIBus Administrator, you can create, edit, delete, start, and stop processes within a service that is configured to run under a process agent. You can also send signals to processes.

When you make configuration changes to a process, you can elect to save your changes to the process control configuration file.

Creating and editing processes

You can set up a process and configure it to run as part of a service.

Before you begin

At least one service must already exist before you can configure processes.

To create or edit a process:

1. Perform any of the following actions from Netcool/OMNIBus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. To add a process, click **New Process** in the toolbar. To edit a process, select the process to edit and then click **Edit** in the toolbar.
The Process Details window opens.
3. Define a new process as follows:

Name Type the process name. This name must be unique per process agent. When editing a process, you cannot change the name.

Command

Type 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, with process agent SFOSYS1_PA, type:

```
$NCHOME/omnibus/bin/nco_objserv -name NCOMS -pa SFOSYS1_PA
```

Host Type the name of the host on which the process is run. Process control automatically resolves the name of the process agent, when required.

Service

Select the service under which this process will run. When editing a process, you cannot change the service.

4. From the **Process** tab, specify additional details about the process. Complete the tab as follows:

Managed

Select this check box if you want to the process to restart automatically if it fails.

Run As ID

Type the user ID under which the process is run. This value is typically 0, which corresponds to the root user name.

Note: If the process agent is not running as root, this option is ignored, and the process is run as the user who is running the process agent.

Retry Count

Specify the number of restart attempts to be made if the process exits within the time specified by the process agent `-retrytime` command-line option. If set to 0, there is no limit to the number of restart attempts. The default is 0.

Process Type

From this drop-down list, select PA Aware to make the process aware of process control and enable the use of all process control features, such as dependencies. Select Not PA Aware if the process can be managed by process control, but cannot use all process control features.

Dependency

Use this drop-down list to indicate whether the process has any dependencies. The option that you select here determines the remaining fields that are displayed:

- Select None to specify that the process has no dependency on any other process. No more fields are displayed.
- Select Delay to indicate a time dependency for starting the process. An additional field titled **Start delay** is displayed for you to define the time dependency.
- Select Process to indicate a dependency on another process. An additional field titled **Select Delay from** or **Dependent Name** is displayed for you to specify a process.

Start delay

Specify a time (in seconds) for starting the process. This time is measured from the start of the service.

Select Delay from/Dependent Name

Use this drop-down list to select another PA aware process on which the process being created or edited depends. The process being created or edited will not start until the process on which it depends is already running.

5. From the **Messages** tab, specify message details that should be sent to the UNIX syslog or the Windows Event Viewer when the process is restarted or exits. Complete the tab as follows:

Restart

Type the message to be sent to the UNIX syslog or the Windows Event Viewer if the process is restarted. For example: The ObjectServer has been restarted.

Alert

Type the message to be sent to the UNIX syslog or the Windows Event Viewer if the process exits. For example: The ObjectServer has gone down.

When the process agent generates an alert or restart message, this message is passed to the syslog or Event Viewer. Tivoli Netcool/OMNIBus has a Syslog probe that can monitor these messages and convert them into ObjectServer alerts.

The alert and restart messages are sent to the UNIX syslog or the Windows Event Viewer 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.

You can use the expansion keywords described in the following table in the restart and alert entries that you specify in the **Messages** tab. Expansion keywords act as variables and contain information about the process that has exited or restarted.

Table 87. Expansion keywords

Expansion keyword	Description
\${NAME}	The name of the process.
\${HOST}	The name of the host running the process.
\${EUID}	The effective user ID under which the process is running.
\${COMMAND}	The command that defines the process.

6. Save or cancel your changes as follows:

OK Click this button to save the process details and close the window. New processes are added to the Service/Process Details pane.

Tip: Your configuration changes are not automatically saved to the process control configuration file. To update the file, click **Write Process Agent Config File** in the toolbar.

Cancel

Click this button to close the window without saving your changes.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Related reference

“Defining processes in the process agent configuration file” on page 264

Deleting a process

When you delete a process, you remove it from the service to which it was assigned.

To delete a process:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Select the process that you want to delete, click **Delete** in the toolbar, and then confirm the deletion. The process is removed from the service definition and the Service/Process Details pane is updated.

Results

Tip: Your configuration changes are not automatically saved to the process control configuration file. To update the file, click **Write Process Agent Config File** in the toolbar.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Starting a process

You can only start processes that have a Dead status.

To start a process:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Select the process that you want to start and then click **Start** in the toolbar.

Results

The process is started and the Service/Process Details pane is updated with the status of the process, including the process ID of the running process.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60


“Connecting to a process agent” on page 281

Stopping a process

You can only stop processes that have a Pending or Running status.

When you stop a process, the status of dependent processes remains unaltered; for example, if an ObjectServer is stopped, the probes that were dependent on it continue to run.

To stop a process:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Select the process that you want to stop and then click **Stop**  in the toolbar.

Results

The process is stopped and the Service/Process Details pane is updated.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Sending signals to processes

You can use Netcool/OMNIbus Administrator to send a UNIX signal to a process.

For example, you might want a probe to re-read its rules file. To do this, send a SIGHUP(1) signal to the probe process.

To send a signal to a process:

1. Perform any of the following actions from Netcool/OMNIbus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Select the process to which you want to send a signal and then click **Send Signal** in the toolbar. The Send Signal window opens.
3. Complete this window as follows:

Process Name

Select the process to which the signal should be sent.

Signal Select the signal that you want to send. Valid signals include:

- SIGHUP(1): Hangup signal to stop and restart a process
- SIGINT(2): Interrupt signal
- SIGTERM(15): Terminate signal

On Windows, the SIGINT(2) and SIGTERM(15) signals are supported only on Tivoli Netcool/OMNIbus processes; for example, ObjectServers, proxy servers, and probes. You can alternatively use the other available methods for stopping processes. When a probe is running under process control, the SIGHUP(1) signal can be used to make the probe re-read its rules file.

4. Confirm or cancel your changes as follows:

OK Click this button to close the window and send the signal to the selected process.

Apply Click this button to send the signal to the selected process and keep the window open.

Cancel

Click this button to close the window without sending a signal.

Related tasks

“Starting Netcool/OMNIbus Administrator” on page 60

“Connecting to a process agent” on page 281

Copying and pasting a service or process between process agent hosts

You can copy and paste a selected service and its related processes, or a selected process, either within a process agent, or from one process agent to another.

To copy and paste a service or process:

1. Perform any of the following actions from Netcool/OMNIBus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. From the Service/Process Details pane of the process agent from which you want to copy a service or process, select the relevant service or process, and then click **Copy** in the toolbar. The selected service and its related processes are copied to the clipboard, or the selected process is copied to the clipboard.
3. To paste the contents of the clipboard within the same process agent, click **Paste** in the toolbar of the current Service/Process Details pane.
To paste the contents of the clipboard to another connected process agent, access the Service/Process Details pane for that process agent, and then click **Paste** in the toolbar.
4. If you copied a process, the Process Details pane opens. Process names must be unique per process agent, so rename the process if necessary, and make any other required changes. Click **OK** to close this window and paste the process details to the Service/Process Details pane. Alternatively, click **Cancel** to cancel the paste operation.
5. If you copied a service, the Process Agent Consistency Checker wizard opens. Service names must be unique within the process control network, and process names must be unique per process agent, so renaming is required. Perform the following actions:
 - a. Click **Next** to proceed to the next page, and amend the service or process details as required. You can specify a new name for the service by overwriting its name. To amend the details of each process, double-click the process entry to obtain the Process Details window, and then make the relevant changes. You can additionally specify whether to include or exclude a process in the paste action. On UNIX, click within the **Include in Paste** cell to toggle between the **true** and **false** options. On Windows, right-click over the process entry and then select or deselect the **Include in Paste** option in the pop-up menu.
 - b. Click **Next** to view a summary of the service and processes to be pasted.
 - c. Click **Finish** to close the wizard and paste the details to the Service/Process Details pane. You can also click **Cancel** to cancel the paste operation.

Related tasks

“Starting Netcool/OMNIBus Administrator” on page 60

“Connecting to a process agent” on page 281

Running an external action

You can use Netcool/OMNIBus Administrator to run a command on a host.

To run an external action:

1. Perform any of the following actions from Netcool/OMNIBus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If not already connected, connect to the process agent. On successful connection, the Service/Process Details pane opens.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. To specify the command details, click **Run External Action** in the toolbar. The External Action Details window opens.
3. Complete this window as follows:

Command

Type a valid command that you want to run on the host.

Host Select the host on which to run the command.

4. Confirm or cancel your changes as follows:

OK Click this button to close the window and run the command.

Cancel

Click this button to close the window without running the command.

Related tasks


“Starting Netcool/OMNIBus Administrator” on page 60

“Connecting to a process agent” on page 281

Stopping a process agent

You can stop a process agent and all its managed processes, or stop a process agent and leave its processes running.

To stop a process agent:

1. Perform any of the following actions from Netcool/OMNIBus Administrator:
 - If you are connected to the process agent and the Service/Process Details pane is open, go to the next step.
 - If you are connected to the process agent, but the Service/Process Details pane is not currently on display, click the **Status** icon in the Configuration window for the process agent, to view this pane.
2. Click **Stop the Process Agent**  in the toolbar.
3. When prompted, choose an option to indicate whether you want to stop the process agent and all its managed processes, or stop the process agent and leave its processes running. Click **OK**.

Results

The process agent, and optionally, its processes, are stopped.

Using process control to run external procedures in automations

The process control system runs external procedures that are specified in automations. External procedures are run on a local or remote host.

An automation does not run programs by itself. It sends a request to a process agent. If necessary, the process agent forwards the request to the process agent that is running on the specified host. The remote process agent then runs the requested program.

External procedures can pass between different operating system environments, and process agents in one operating system can run automations sent by process agents in another operating system.

When you run the process agent from the command line on UNIX and Windows, the working directory for all child processes is the directory from which the process agent was started.

When you run the process agent as a UNIX daemon, the working directory for all child processes is `$NCHOME/omnibus`.

On Windows, the default directory for a process agent that is running as a Windows service is `%NCHOME%\omnibus\log`. This directory is also the default working directory of any child processes that are spawned by the process agent.

Tip: When running external procedures, the **PA.Username** and **PA.Password** ObjectServer properties must be set to a valid user name and password combination within the ObjectServer properties file, for authentication purposes. The **PA.Name** ObjectServer property must also be set to the name of the process agent that the ObjectServer uses to run external automations. These settings ensure that connection to the process agent, and the running of the external procedure, are successful.

Related reference

“ObjectServer properties and command-line options” on page 3

Chapter 8. Performance tuning

Tivoli Netcool/OMNIBus performance can be measured in terms of response time, throughput, and availability.

These performance metrics are affected by several factors, which include:

- The number of events, including details and journals, in the ObjectServer
This number has a direct bearing on the volume of data that is transferred during resynchronization operations, and on the number of rows that need to be scanned during SQL queries from clients and triggers.
- The event throughput into the system from probes or inbound gateways
- The event throughput out of the system from outbound gateways
- The number of concurrent Web GUI and desktop display clients, and the number of other connected clients that compete for the ObjectServer's time
- The number and complexity of Web GUI and desktop filters, and the number of connections and complexity of actions in the policies that are submitted by Netcool/Impact clients, where applicable
- The efficiency of the custom ObjectServer automations, and whether they are subjecting the ObjectServer to unnecessary workload
- The granularity settings of the ObjectServer (that is, how often it sends IDUC broadcasts to clients)
- Your hardware and system configuration

To help obtain an optimal level of performance for the ObjectServer, you can monitor the number and throughput of connecting clients, evaluate the efficiency of the ObjectServer automations, and design efficient indexes to support your SQL queries. Also consider the use of a multitiered architecture to support high availability of your Tivoli Netcool/OMNIBus installation.

Tivoli Netcool/OMNIBus key performance indicators

To ensure that a Tivoli Netcool/OMNIBus system is running effectively, you can monitor several key performance indicators (KPIs).

Each KPI description defines the KPI, explains how to set it up, explains usefulness, and gives an indicator of what to monitor, that is, what KPI values indicate good performance, and what values might need to be investigated. For many KPIs, the values given are not absolute; instead you can use these values indicatively. For your own Tivoli Netcool/OMNIBus system, you must establish baseline values on a system that is known to be working effectively, for example on a test environment, and compare the values with the current and future values on the production system.

The KPIs provided do not constitute an exhaustive list. Depending on your Tivoli Netcool/OMNIBus environment, certain KPIs will require special monitoring, while others might not be important.

ObjectServer key performance indicators

Monitor these key performance indicators (KPIs) to establish the effectiveness of the ObjectServer.

The following ObjectServer KPIs can be monitored:

Total time used in granularity

To set up this KPI, you must run the ObjectServer with profiling enabled and run the trigger_stats_report automation. The profiling data shows how much time the ObjectServer has spent profiling queries from clients, including the time spent in automations raised by the client activity. The trigger_stats.log file contains the amount of time each trigger has used in the last profiling period. To demonstrate that the ObjectServer can process all the requests in the available time, the processing time needs to be less than the time of the reporting period. Note that the trend of this KPI over time is more important than its value at a given moment.

Tip: On multiprocessor servers, the time spent processing requests from clients and automations can be greater than the reporting period, because of the multi-threaded nature of the ObjectServer.

CPU usage of the nco_objserv process

To monitor the CPU usage of the nco_objserv process, set up and use an IBM Tivoli Monitoring agent.

The CPU usage of processes is one of the most important metrics for determining performance; for an ObjectServer under heavy load, the performance is most likely bound to the CPU usage.

Tip: When troubleshooting performance, the profile log file and trigger statistics log file are the first place to investigate. Generally, if the total combined time for both clients and triggers is consistently over 60 seconds (the default granularity period), some action needs to be taken. Various operating system metrics can also be useful in identifying whether a system is under stress. The key metrics are the CPU utilization and the process size of the Tivoli Netcool/OMNIbus processes.

Number of rows in the alerts.status table

The performance of unindexed queries is proportional to the number of rows in the alerts.status table. As the number of rows in the alerts.status table increases, so the time needed to perform all queries and execute all triggers against the alerts.status table increases. Additionally, holding row data accounts for most of the memory used by the ObjectServer; the data of the alerts.status table accounts for most of this row data. The number of rows in the alerts.status table fluctuates depending on the networks and applications that are being monitored. However, the number of rows should be stable over a period of weeks. If the number of rows in the alerts.status table is increasing over the weeks then, depending on the rate of the increase, a problem might be developing. The alerts.status table uses more memory than the alert.details table and the alerts.journal table. For more information about the event count in the alerts.status table, see “Other useful information” on page 300.

Number of rows in the alerts.details table

The performance of the queries performed by the Web GUI clients to retrieve alert details is proportional to the number of entries in the alerts.details table. The more entries are contained in the alerts.details table, the slower the queries become. Additionally, holding row data

accounts for most of the memory used by the ObjectServer; the data of the alerts.details table accounts for some of this row data. Each entry in the alerts.details table increases the load on any gateways that are configured to forward this data to other ObjectServers or other applications. The number of rows in the alerts.details table fluctuates depending on the networks and applications that are being monitored, the day of the week, time of day, and so on. However, the number of rows should be stable over a period of weeks. If the number of rows in the alerts.details table is increasing over time, or as new devices or probe rules files are introduced, then, depending on the rate of the increase, a problem might be developing. The alerts.details table uses more memory than the alerts.journal table, but less memory than the alerts.status table.

Tip: Use the `nvp_add` function to move data from alerts.details table entries to the ExtendedAttr field of the alerts.status table. The use of the `nvp_add` function means that the number of inserts performed by a probe is reduced.

For more information about the event count in the alerts.details table, see “Other useful information” on page 300.

Number of rows in the alerts.journal table

The number of rows in the alerts.journal table fluctuates depending on the automation that is processing the events, or the user who is processing the events. However, the number of rows should be stable over a period of weeks. If the number of rows in the alerts.journal table is increasing over the weeks then, depending on the rate of the increase, a problem might be developing. Ensure that an entry is made in the alerts.journal table only when a trigger affects the content of a row, regardless of the frequency of a trigger. Use indicators to make sure that an event is not unnecessarily reprocessed by the same trigger. For more information about the event count in the alerts.journal table, see “Other useful information” on page 300.

Number of inserts in the alerts.status table in the previous *n* seconds

For this KPI, *inserts* refers to new rows and deduplications. To monitor the number of inserts in the alerts.status table, enable the stats_triggers trigger group and the statistics_gather automation. The data gathered by this trigger group and automation is written periodically to the master.stats table, in the StatusInserts column. The performance metric can be derived by comparing the value of the StatusInserts column with the value from the previous report. This KPI enables you to identify large increases in input to the system caused by the monitoring of new devices or applications, and can be used to increase the scalability of your system, and plan capacity. The number of rows in the alerts.status table fluctuates depending on the networks and applications that are being monitored, the day of the week, time of day, and so on. However, the number of rows should be stable over a period of weeks. If the number of rows in the alerts.status table is increasing over the weeks, or as new devices or probe rules files are introduced, then, depending on the rate of the increase, a problem might be developing, for example, a fault or problem with the probe rules file.

Memory usage of the nco_objserv process

To monitor the memory usage of the nco_objserv process, set up and use the IBM Tivoli Monitoring Agent for Tivoli Netcool/OMNIBus. Monitor the memory usage of the nco_objserv process to ensure that the usage does not approach any memory limits, and to help you identify any increases in

memory usage, and determine the cause of the increases. The memory usage of the process increases proportionally to increases in the number of rows in the alerts.status table, alerts.details table, and the alerts.journal table (or any additional tables you have defined) to increases in the number of connections, and increased usage by clients. The memory usage should remain stable over time, and any increases should correspond to increases in the numbers of table rows, or additional clients.

For more information about the IBM Tivoli Monitoring Agent for Tivoli Netcool/OMNIbus, see the IBM Tivoli Monitoring for *Tivoli Netcool/OMNIbus Agent User's Guide*.

Number of connections into the ObjectServer

To monitor the number of connections into the ObjectServer, enable the stats_triggers trigger group and the statistics_gather automation. The data gathered by this trigger group and automation is written periodically to the master.stats table, in the NumClients column. Alternatively, you can run a manual count of the number of rows in the catalog.connections table. Only a finite number of connections can be made to the ObjectServer. When the maximum number of connections is reached, new connections are refused. A refused connection might result in the temporary loss of access to data or loss of input to probes or gateways. The maximum number of connections is 1024. The maximum permitted number of connections is determined by the ObjectServer **Connections** property, with a default of 30. The number of connections varies according to your environment and its usage. However, the number should remain stable when compared over a number of weeks. If the number of connections is increasing over the weeks then, depending on the rate of the increase, a problem might be developing.

Usage of memstore

To monitor the memstore, inspect the content of the catalog.memstores table. For each row, compare the value of the UsedBytes column with the values of the SoftLimit column and the HardLimit column. Memstores are containers that are maintained by the ObjectServer, they contain ObjectServer data and tables in the memory. Memstores have a finite size, and, when full, do not permit any further data to be inserted. Consequently, you must ensure that the memstores do not become full. The usage of the memstores varies according to your environment and its usage. However, the usage should remain stable when compared over a number of weeks. If the usage of the memstores is increasing over the weeks then, depending on the rate of the increase, a problem might be developing.

Other useful information

To monitor the number of rows in the alerts.status table, alerts.details table, and alerts.journal table, enable the stats_triggers group and the statistics_gather automation. To enable triggers, use the **nco_config** command to start Netcool/OMNIbus Administrator, or use the ALTER TRIGGER GROUP command. The data gathered by this trigger group and automation is written periodically to the master.stats table. The default interval is 300 seconds; this value is configurable. The following table describes the column to which the count of the number of rows is written:

Table 88. Columns to which event count is written for ObjectServer tables

Table	Column to which the count is written
alerts.status	EventCount
alerts.details	DetailCount
alerts.journal	JournalCount

Several of the KPIs described in the preceding list can also be monitored by using IBM Tivoli Monitoring agents. Working knowledge of IBM Tivoli Monitoring is required. For information about setting up IBM Tivoli Monitoring agents, see the *IBM Tivoli Monitoring* information center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/index.jsp>.

Related reference

“master.stats table” on page 349

Related information

 IBM Tivoli Monitoring for Tivoli Netcool/OMNIBus Agent

Probe key performance indicators

Probes can be configured to generate ProbeWatch heartbeat events as a self-monitoring mechanism to help monitor performance, diagnose performance problems, and highlight performance bottlenecks before they affect the system.

Tip: For more information about setting up IBM Tivoli Monitoring agents, see the *IBM Tivoli Monitoring* information center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/index.jsp>

The following KPIs can be established to monitor the health of probes:

Number of events received by a probe in previous *n* seconds

The number of events received by the probe in the previous *n* seconds can be derived from the NumEventsProcessed column of the master.probestats table as a delta from the previous reported value for each probe. Probe throughput generates work for the ObjectServer; a flood of events from a specific probe should be investigated. It might highlight a problem with the probe, the probe rules file, or the devices or applications that are being monitored by that probe. Compare the current value against the previous values for this KPI to identify abnormal behaviour.

Probe CPU usage

The CPU usage of the probe is contained in the CPUTimeSec column of the master.probestats table. An IBM Tivoli Monitoring agent installed on the probe computer can also measure the CPU usage of the probe. CPU resources are finite. If the probe process is at maximum CPU, events are queued in the probe until the probe can process them. Consequently, probe input might build up, which can cause delays in processing, or, depending on the probe, can cause loss of data. Contributory factors are the incoming event load and the rules file processing.

Probe memory footprint

The memory footprint of the probe is contained in the ProbeMemory column of the master.probestats table. An IBM Tivoli Monitoring agent installed on the probe computer can also measure the memory usage of the probe. Memory is a finite resource and probe memory should not grow unbounded. Memory usage of a probe process should be relatively stable,

although some increase is expected as caches and buffers build. The memory footprint of a probe will increase when the first SIGHUP signal is sent to the probe to instruct the probe to reread its rules file. This increase is expected as the new rules file is read and parsed before the memory used by the existing rules file is released. This is necessary so that the probe always has a valid rules file. Subsequent SIGHUP signals to reread the rules file should cause only a comparatively small increase in the memory usage. Use of associative arrays might also contribute to increased memory usage of the probe, because the arrays are built up by the events that are processed by the rules file. The memory footprint of the nco_p_mttrapd probe is distinctive because it maintains a large buffer for incoming traps. This can often account for over 50MB of memory growth as the first 2000 traps are received. After the memory for the trap queue buffer has been allocated the memory usage should settle down. Other unexplained unbounded memory growth needs to be investigated.

Average time spent processing rules

The average time spent processing the rules file is contained in the AvgRulesFileTime column of the master.probestats table.

Inefficiencies in the rules file may cause delays in event processing. The time spent processing the rules file is one of the major factors in limiting maximum throughput of a probe. If rules file processing is taking, on average, 5,000 microseconds (millionths of a second) then the probe will only be able to process 200 events per second maximum.

Gateway key performance indicators

Use these key performance indicators (KPIs) to monitor the performance of gateways.

Tip: For more information about setting up IBM Tivoli Monitoring agents, see the *IBM Tivoli Monitoring* information center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/index.jsp>

The following KPIs can be used to monitor gateway performance:

CPU usage of gateway processes

An IBM Tivoli Monitoring agent installed on the gateway computer can measure the CPU usage of the gateway process or processes. CPU resources are finite. If the gateway process is at maximum CPU, the gateway might not be able to process all events, and events might be delayed reaching the target system.

Memory usage of gateway processes

An IBM Tivoli Monitoring agent installed on the gateway computer can measure the memory usage of the gateway process or processes. Memory resources are finite. Unbounded memory growth might cause the abnormal termination of the process or processes. Memory usage of a gateway process should be relatively stable although some increase is expected as caches build, and while the gateway is temporarily storing alert data as the data is passed between the ObjectServer and its destination. Unbounded memory growth needs to be investigated. Memory growth in the ObjectServer Gateway might be due to a problem in the ObjectServer.

For more information about the IBM Tivoli Monitoring Agent for Tivoli Netcool/OMNIBus, see the *IBM Tivoli Monitoring for Tivoli Netcool/OMNIBus Agent User's Guide*.

Best practices for performance tuning

Use these best practice guidelines to help configure your Tivoli Netcool/OMNIbus system for optimal performance. Work through each of the steps in turn to help you identify and resolve changes that adversely affect performance, and to fine tune performance.

For more information about performance tuning for the Web GUI, see the *IBM Tivoli Netcool/OMNIbus Web GUI Administration and User's Guide*.

Run the ObjectServer with profiling enabled

Use profiling to measure the amount of time spent running SQL queries on the ObjectServer and to identify which client connections are using up excessive resources.

To enable profiling on the ObjectServer:

1. Ensure that the **Profile** property is set to TRUE. (This is the default value.)
2. Use the **ProfileStatsInterval** property to specify an interval at which profiling information is written to the profile log file. A default interval of 60 seconds is used if you do not change this value.
3. Ensure that the profiler_triggers trigger group and its triggers (profiler_group_report, profiler_report, profiler_toggle) are enabled for profile logging.

Timing information for running SQL commands from client connections is logged to the catalog.profiles table. You can use Netcool/OMNIbus Administrator to view details that are recorded in the catalog.profiles table. From the Netcool/OMNIbus Administrator window, select the **System** menu button and then click **Databases**. You can use the **Data View** tab on the Databases, Tables and Columns pane to view table data, and use the **Column Definitions** tab to view detailed information about the columns in the table.

Profile statistics are also logged to a profile log file \$NCHOME/omnibus/log/*servername_profiler_report.logn*, where *servername* represents the ObjectServer name and *n* is a number. The profile log file shows a breakdown of the time spent for each client connection and the total time spent by client type, for each granularity period (as set by the **Granularity** property). Each client shown in the log file is identified by a standard default name (for example, GATEWAY or PROBE) and the host on which the client is running. You can use the profile log file to analyze how the ObjectServer spent its time during each granularity period and calculate the percentage of time used. For example, if the granularity period is set to 60 seconds and the total time spent for all the connections during a particular period was 30 seconds, you can calculate that the ObjectServer spent 50% of its available time on running SQL commands from client connections.

The work completed in a report period is output in a summary line for each granularity period. The information presented in the summary line is displayed in the following format: Total time in the report period (profiling period): total time by all clients. The total time by all clients can be greater than the profiling period due to the multi-threaded nature of ObjectServer. This is especially true for multi-CPU systems. If the profiling period is greater than the configured profiling

period it means that ObjectServer is too busy to report the profiling time and might indicate the ObjectServer is overloaded. If the total time by all clients is greater than the profiling period, it indicates the system is under load, but does not necessarily indicate a problem.

Sample output recorded in a profile log file for a granularity period is as follows:

```
[1] Mon Oct 12 17:39:46 2009: Individual user profiles:
[2] Mon Oct 12 17:39:46 2009: 'Administrator' (uid = 0) time on adminhost: 0.000000s
[3] Mon Oct 12 17:39:46 2009: 'isql' (uid = 0) time on omnihost1.ibm.com: 3.770000s
[4] Mon Oct 12 17:39:46 2009: 'PROBE' (uid = 0) time on probehost.ibm.com: 5.010000s
[5] Mon Oct 12 17:39:46 2009: 'e@c0B4D@c0142:11.0' (uid = 0) time on omnihost1.ibm.com: 10.010000s
[6] Mon Oct 12 17:39:46 2009: 'c@xxxxx@xxxxx:11.0' (uid = 45) time on omnihost1.ibm.com: 0.000000s
[7] Mon Oct 12 17:39:46 2009: 'e@c0B4D@c0142:11.0' (uid = 45) time on omnihost1.ibm.com: 9.870000s
[8] Mon Oct 12 17:39:46 2009: 'c@xxxxx@xxxxx:11.0' (uid = 55) time on omnihost1.ibm.com: 0.000000s
[9] Mon Oct 12 17:39:46 2009: 'e@c0B4D@c0142:11.0' (uid = 55) time on omnihost1.ibm.com: 6.020000s
[10] Mon Oct 12 17:39:46 2009: 'GATEWAY' (uid = 0) time on omnihost1.ibm.com: 0.270000s
[11] Mon Oct 12 17:39:46 2009: 'GATEWAY' (uid = 0) time on omnihost1.ibm.com: 0.000000s
[12] Mon Oct 12 17:39:46 2009: 'PROBE' (uid = 0) time on omnihost1.ibm.com: 3.010000s
[13] Mon Oct 12 17:39:46 2009: Grouped user profiles:
[14] Mon Oct 12 17:39:46 2009: Execution time for all connections whose application name is 'PROBE': 8.020000s
[15] Mon Oct 12 17:39:46 2009: Execution time for all connections whose application name is 'GATEWAY': 0.270000s
[16] Mon Oct 12 17:39:46 2009: Execution time for all connections whose application name is 'c@xxxxx@xxxxx:11.0': 0.000000s
[17] Mon Oct 12 17:39:46 2009: Execution time for all connections whose application name is 'e@c0B4D@c0142:11.0': 25.930000s
[18] Mon Oct 12 17:39:46 2009: Execution time for all connections whose application name is 'isql': 3.770000s
[19] Mon Oct 12 17:39:46 2009: Execution time for all connections whose application name is 'Administrator': 0.000000s
[20] Mon Oct 12 17:39:46 2009: Total time in the report period (59.275782s): 29.980000s
```

The line numbers are included in the preceding output to help describe the entries:

- Line [1]: Introduces a list of individual clients that are connected to the ObjectServer.
- Line [2]: Shows the application name for the connected client (Administrator), the associated user for that client (user ID 0), the host computer (adminhost), and the amount of time the client has used in the last profiling period (0.000000s).
- Line [13]: Introduces a list that shows the aggregated time for all clients of the same type.
- Line [14]: Shows that the two connected probes used a combined time of 8.02 seconds.
- Line [17]: Shows that the event lists used 25 seconds. Consider investigating the individual times to see which event list is using the most time.
- Line [20]: Shows that the profiling period as 59.27 seconds and the total time by all clients as 29.98 seconds. The profiling period is approximately the same as the configured profiling period of 60 seconds; this would be expected if the system is not over loaded.

Analyze the profiling statistics in the log file and database table to identify which clients are using the most time and why:

- Determine whether all the client connections are necessary, and drop any redundant client connections; for example, event lists that are left connected after operators have vacated the premises.
- If a desktop event list or a Web GUI client is using a lot of time, focus on the filters that are being used by that client. Analyze the filters both for the number and complexity of the individual queries, with the aim of making them more efficient.
- If the client is a probe, performance degradation might be due to poorly-written rules files that allow unnecessary events to be forwarded to the ObjectServer, the amount of detail information sent per event, or event flooding.
- Increase the granularity period of the ObjectServer to alleviate the effects of heavy client loads. This action slows down the rate at which the ObjectServer sends IDUC broadcasts to its clients, and can lead to improved system

performance. However, incoming events will take longer to reach clients, particularly if the ObjectServer is part of a multitiered architecture.

Related reference

“ObjectServer properties and command-line options” on page 3

“catalog.profiles table” on page 347

Collect statistical information about triggers

Timing information about triggers, including the number of times the trigger has been raised and the number of times the trigger has fired, are saved to the catalog.trigger_stats table.

To collect trigger statistics:

1. Ensure that the **Auto.Enabled** property of the ObjectServer is set to TRUE. This is the default setting, and is used to enable the automation system.
2. Use the **Auto.StatsInterval** property to control the frequency at which the automation system collects and stores statistical information to the catalog.trigger_stats table. A default interval of 60 seconds is used if you do not change this value.
3. Ensure that the trigger_stat_reports trigger group and the trigger_stats_report trigger are enabled.

You can use Netcool/OMNIbus Administrator to view details that are recorded in the catalog.trigger_stats table. From the Netcool/OMNIbus Administrator window, select the **System** menu button and then click **Databases**. You can use the **Data View** tab on the Databases, Tables and Columns pane to view table data, and use the **Column Definitions** tab to view detailed information about the columns in the table.

Trigger statistics are also logged to the file \$NCHOME/omnibus/log/servername_trigger_stats.log*n*, where *servername* represents the ObjectServer name and *n* is a number. The trigger statistics log file shows the amount of time that each trigger has used in the last profiling period. You can use this log file for automation debugging, and to determine which triggers are slow due to slow-running SQL queries. Sample output recorded in a trigger statistics log file is as follows:

```
[1] Mon Oct 12 18:03:56 2009: Trigger Profile Report
[2] Mon Oct 12 18:03:56 2009: Trigger Group 'compatibility_triggers'
[3] Mon Oct 12 18:03:56 2009: Trigger Group 'system_watch'
[4] Mon Oct 12 18:03:56 2009:   Trigger time for 'system_watch_shutdown': 0.000000s
[5] Mon Oct 12 18:03:56 2009:   Trigger time for 'system_watch_startup': 0.000000s
[6] Mon Oct 12 18:03:56 2009: Trigger Group 'sae'
[7] Mon Oct 12 18:03:56 2009:   Trigger time for 'update_service_affecting_events': 0.006790s
[8] Mon Oct 12 18:03:56 2009: Trigger Group 'default_triggers'
[9] Mon Oct 12 18:03:56 2009:   Trigger time for 'deduplication': 0.341918s
[10] Mon Oct 12 18:03:56 2009:   Trigger time for 'deduplication_eval': 0.092659s
[11] Mon Oct 12 18:03:56 2009:   Trigger time for 'service_update': 0.000000s
[12] Mon Oct 12 18:03:56 2009:   Trigger time for 'clean_journal_table': 0.000172s
[13] Mon Oct 12 18:03:56 2009:   Trigger time for 'service_insert': 0.000000s
[14] Mon Oct 12 18:03:56 2009:   Trigger time for 'service_reinsert': 0.000000s
[15] Mon Oct 12 18:03:56 2009:   Trigger time for 'clean_details_table': 0.000083s
[16] Mon Oct 12 18:03:56 2009:   Trigger time for 'state_change': 0.075508s
[17] Mon Oct 12 18:03:56 2009:   Trigger time for 'deduplication_copy': 0.022087s
[18] Mon Oct 12 18:03:56 2009:   Trigger time for 'new_row': 0.002637s
[19] Mon Oct 12 18:03:56 2009:   Trigger time for 'deduplicate_details': 0.000000s
[20] Mon Oct 12 18:03:56 2009: Trigger Group 'connection_watch'
[21] Mon Oct 12 18:03:56 2009:   Trigger time for 'connection_watch_connect': 0.000000s
[22] Mon Oct 12 18:03:56 2009:   Trigger time for 'connection_watch_disconnect': 0.000000s
[23] Mon Oct 12 18:03:56 2009: Trigger Group 'primary_only'
[24] Mon Oct 12 18:03:56 2009:   Trigger time for 'generic_clear': 5.879707s
[25] Mon Oct 12 18:03:56 2009:   Trigger time for 'expire': 0.008233s
[26] Mon Oct 12 18:03:56 2009:   Trigger time for 'delete_clears': 0.007219s
[27] Mon Oct 12 18:03:56 2009:   Trigger time for 'enrich_and_correlate': 23.007219s
```

```

[28] Mon Oct 12 18:03:56 2009: Trigger Group 'security_watch'
[29] Mon Oct 12 18:03:56 2009:   Trigger time for 'disable_user': 0.000000s
[30] Mon Oct 12 18:03:56 2009:   Trigger time for 'reset_user': 0.000000s
[31] Mon Oct 12 18:03:56 2009:   Trigger time for 'security_watch_security_failure': 0.000000s
[32] Mon Oct 12 18:03:56 2009: Trigger Group 'profiler_triggers'
[33] Mon Oct 12 18:03:56 2009:   Trigger time for 'profiler_group_report': 0.065094s
[34] Mon Oct 12 18:03:56 2009:   Trigger time for 'profiler_report': 0.087705s
[35] Mon Oct 12 18:03:56 2009:   Trigger time for 'profiler_toggle': 0.000000s
[36] Mon Oct 12 18:03:56 2009: Trigger Group 'trigger_stat_reports'
[37] Mon Oct 12 18:03:56 2009:   Trigger time for 'trigger_stats_report': 0.198813s
[38] Mon Oct 12 18:03:56 2009: Trigger Group 'iduc_triggers'
[39] Mon Oct 12 18:03:56 2009:   Trigger time for 'disconnect_iduc_missed': 0.000000s
[40] Mon Oct 12 18:03:56 2009:   Trigger time for 'iduc_stats_update': 0.000949s
[41] Mon Oct 12 18:03:56 2009:   Trigger time for 'iduc_messages_tblclean': 0.000089s
[42] Mon Oct 12 18:03:56 2009:   Trigger time for 'deduplicate_iduc_stats': 0.000000s
[43] Mon Oct 12 18:03:56 2009:   Trigger time for 'iduc_stats_insert': 0.000000s
[44] Mon Oct 12 18:03:56 2009: Trigger Group 'automatic_backup_system'
[45] Mon Oct 12 18:03:56 2009:   Trigger time for 'backup_succeeded': 0.000000s
[46] Mon Oct 12 18:03:56 2009:   Trigger time for 'backup_failed': 0.000000s
[47] Mon Oct 12 18:03:56 2009:   Trigger time for 'backup_state_integrity': 0.000000s
[48] Mon Oct 12 18:03:56 2009: Trigger Group 'gateway_triggers'
[49] Mon Oct 12 18:03:56 2009:   Trigger time for 'resync_finished': 0.000000s
[50] Mon Oct 12 18:03:56 2009: Time for all triggers in report period (60s): 29.789663s

```

The line numbers are included in the preceding output to help describe the entries:

- Line [1]: Indicates the start of the report and shows the timestamp for when the report was produced.
- Line [2]: Shows that the report is broken down by trigger group. In this case, the `compatibility_triggers` trigger group does not contain any enabled triggers.
- Line [3]: Shows `system_watch` as the first trigger group that contains enabled triggers.
- Line [4]: Shows the name of the trigger, and the amount of time (in seconds) used by the trigger in the last profiling period.
- Line [27]: Indicates an excessive amount of time for the trigger. If this is a regular occurrence, the trigger would need to be investigated further. For example, the following questions could apply: Are there table scans in a database trigger? Are nested scans being used? Could an index be used to reduce scans? Is the time directly related to the number of events that the system is dealing with? Does the trigger use an `EVALUATE` clause that could be replaced by a `FOR EACH` clause operating directly on the table with the `ACTION` clause?
- Line [50]: Shows the summary line as the last entry in the report. This line shows the time in seconds since the last report was run, and the total amount of time used by the triggers in the reporting period. In this example, 29 seconds out of 60 seconds is a high percentage, so further investigation might be necessary to determine the cause, particularly if this value is a regular occurrence.

Analyze the trigger statistics in the log file and database table to determine whether any workload is causing a degradation in performance:

- If a trigger is identified as using the majority of the granularity period, investigate the cause.
- Review your custom ObjectServer automations to assess their efficiency and to reduce the workload on the ObjectServer.
- Make sure that trigger execution time is kept to a minimum, because no other writes can occur while a trigger is being executed.

Tip: When troubleshooting performance, the profile log file and trigger statistics log file are the first place to investigate. Generally, if the total combined time for both clients and triggers is consistently over 60 seconds (the default granularity

period), some action needs to be taken. Various operating system metrics can also be useful in identifying whether a system is under stress. The key metrics are the CPU utilization and the process size of the Tivoli Netcool/OMNIBus processes.

Related reference

“catalog.trigger_stats table” on page 350

“Standard Tivoli Netcool/OMNIBus automations” on page 230

Review and revise your system architecture

Use the multitiered customizations provided with Tivoli Netcool/OMNIBus to deploy your installation in a one-, two-, or three-tiered architecture, in which the components sit within collection, aggregation, and display layers.

At a minimum, set up a virtual pair of failover ObjectServers in the aggregation layer, and then add collection or display components, as required.

If inbound probe traffic is causing problems, consider implementing a collection layer (if not already in place) with collection ObjectServers that are dedicated to handling the incoming events before passing them up to the aggregation layer.

If display client traffic is causing problems such as slow response times for users, consider implementing a display layer (if not already in place) with display ObjectServers that are dedicated to handling client requests.

If both inbound probe traffic and display client traffic are causing problems, consider implementing the three-tier architecture with a collection layer, aggregation layer, and display layer.

Also configure Tivoli Netcool/OMNIBus for high availability to reduce resynchronization time between the failover pair of ObjectServers, minimize event loss, and improve data integrity.

Enable the stats_triggers trigger group

In the default ObjectServer configuration, the stats_triggers group uses triggers to gather several statistics and metrics. You can enable the trigger group in a production environment.

The triggers in the stats_triggers trigger group count the number of inserts to the alerts.status table, the number of inserts to the alerts.details table, the number of inserts to the alerts.journal table, and the number of deduplications. These counts are stored in the master.activity_stats table and are aggregated by the statistics_gather trigger.

To monitor the number of rows in the alerts.status table, alerts.details table, and alerts.journal table, enable the stats_triggers group and the statistics_gather automation. To enable triggers, use the **nco_config** command to start Netcool/OMNIBus Administrator, or use the ALTER TRIGGER GROUP command. The data gathered by this trigger group and automation is written periodically to the master.stats table. The default interval is 300 seconds; this value is configurable. The following table describes the column to which the count of the number of rows is written:

Table 89. Columns to which event count is written for ObjectServer tables

Table	Column to which the count is written
alerts.status	EventCount

Table 89. Columns to which event count is written for ObjectServer tables (continued)

Table	Column to which the count is written
alerts.details	DetailCount
alerts.journal	JournalCount

Related concepts

Chapter 4, “Using Netcool/OMNIBus Administrator to configure ObjectServers,” on page 59

Related reference

“Modifying a trigger group (ALTER TRIGGER GROUP command)” on page 206

“master.stats table” on page 349

Review and revise your probe configuration files

Carefully review the probe properties file and rules file settings to ensure that neither introduce any inefficiency to incoming event processing.

In general, keep both configuration files as simple as possible. If high event throughput is expected, consider the use of multiple probes to gather the incoming events - this is more likely to minimize probe slowdown and dropped events.

Configure event flood detection

Configure your probes to detect when they are subject to an event flood or other anomalous event rates, and to perform remedial actions.

Sample rules files are provided in the `$NCHOME/omnibus/extensions/eventflood` directory, which you can use for this configuration.

Manage the volume of information in the alerts.details table

When a high volume of alert information is stored in the alerts.details table, ObjectServer performance significantly deteriorates.

Use the following guidelines to manage the volume of information that is stored in this table.

- Ensure that the `clean_details_table` automation is enabled on both your primary and backup ObjectServers. This automation performs housekeeping cleanup on the alerts.details table, and deletes any entries not found in the alerts.status table.
- Avoid using `details($*)` in probe rules files, to add all the alert information to the alerts.details table. For each alert in the alerts.status table, multiple rows might be added to the alerts.details table because the `details($*)` command in the rules file records each token as one row. After using `details($*)` for long periods of time, the ObjectServer tables become very large and the performance of the ObjectServer suffers. Only use `details($*)` when you are debugging or writing rules files.

If you need to add more information for the alert, use the `details` statement to add specific elements or use a regular expression to extract specific elements from the details. For example, `details($a,$b)` adds the elements `$a` and `$b` to the alerts.details table.

Note also that each `details` entry requires a separate `INSERT` statement, so for an event with 20 `details` entries, 21 inserts will be made to the ObjectServer. An event with no details will, however, be a single insert.

- From the SQL interactive interface, use the DELETE SQL command to clear all the records in the alerts.details table. For example:
`delete from alerts.details;`

Note: Manually deleting data from the alerts.details table is a temporary solution because the number of details will increase again to a high volume if the first two guidelines for managing the alerts.details table are not followed. Instead of using alerts.details, consider using the ExtendedAttr column with the **nvp_add rules** file function, and the **nvp_get** or **nvp_set** SQL functions to store those fields that have been put in details.

Use a monitoring agent to monitor and manage Tivoli Netcool/OMNIbus resources

An IBM Tivoli Monitoring agent is available for monitoring Tivoli Netcool/OMNIbus health and performance, automation triggers, and event activity and distribution. This monitoring agent includes a set of automations that add further instrumentation to the ObjectServer.

The IBM Tivoli Monitoring for Tivoli Netcool/OMNIbus agent is available for download as part of the Tivoli Netcool/OMNIbus base installation package. The prerequisite IBM Tivoli Monitoring software is also available for download.

For information about installing and configuring the monitoring agent, go to the IBM Tivoli Network Management Information Center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp>. From the navigation pane on the left, locate and expand the relevant *Tivoli Netcool/OMNIbus* version node, and then go to the *IBM Tivoli Monitoring for Tivoli Netcool/OMNIbus* node. A PDF publication is also available in the *PDF documentation set* node.

After setting up the monitoring agent, monitor the system resources, and perform any remedial actions that are required to improve performance or availability.

Review and amend your SQL queries, and create a selection of well-designed, efficient indexes

It is useful to understand how SQL queries are optimized so that you can construct efficient SQL queries. Review your existing SQL queries and amend them to benefit from optimization and indexing.

Read through the SQL query and indexing guidelines provided.

When designing and creating indexes, it is also useful to understand the characteristics of the ObjectServer database tables and columns, to help you gauge which indexed columns might help to enhance performance. Integer comparisons are faster than string comparisons, so if your event data contains strings that are constants, consider using integers to represent the strings within your rules files, automations, and filters, and then use conversions to display the strings to users. For example, the Class column in the ObjectServer is an integer data type, but is displayed as a string in event lists. You can take the following actions to pass event data as integers, which are then displayed as strings:

- Revise your probe rules files to set integer values that map to corresponding strings.
- Revise your automations (and Netcool/Impact policies if applicable) to use the integer values.

- Revise your filters to use the integer values in WHERE clauses.
- Add conversions that map the integer values to the string values that will be displayed to users.

You can use the CREATE INDEX SQL command to create indexes, and the DROP INDEX command to delete redundant indexes. Details about the indexes that you create are stored in the catalog.indexes table.

After revising your SQL queries and creating indexes, temporarily set the **MessageLevel** property to debug so that the execution time of individual SQL queries, and indexing details, will be logged. Allow a suitable period of processing activity.

Examine the ObjectServer log file `$NCHOME/omnibus/log/server_name.log` to:

- Determine which SQL queries negatively impacted performance; check to see which queries took too long, or were executed repeatedly.
- Determine which indexes were used.
- Analyze the response times for SQL queries that are frequently used, and assess whether the benefits are significant or marginal.

The following steps describe how to determine the length of time a specific SQL query takes to run:

1. Look for an ObjectServer log message that is similar to the following sample:

```
2010-03-19T14:39:27: Debug: D-0BJ-105-010: Client language command on connection ID N: [user1][isql][][hostname.ibm.com] [SQL statement].
```

where *SQL statement* is the SQL statement you want to analyze.
2. Make a note of the connection ID number *N*.
3. Look for an ObjectServer log message that is similar to the one shown below, which contains the same value for *N*:

```
2010-03-19T14:39:27: Information: I-0BJ-104-016: Profiler timing submitted from connection ID N: time in seconds
```

where the *time in seconds* is the length of time taken for the SQL query to run.

Using the following ObjectServer log output as an example:

```
2010-03-19T14:39:27: Debug: D-0BJ-105-010: Client language command on connection ID 1: [user1][isql][][hostname.ibm.com] [select * from catalog.indexes;]. 2010-03-19T14:39:27: Information: I-0BJ-104-016: Profiler timing submitted from connection ID 1: 0.000265
```

You can see that it has taken 0.000265 seconds for the select * from catalog.indexes command to complete.

Note: Setting ObjectServer logging to debug mode can be performed without taking the ObjectServer offline. This setting can adversely affect performance, so you must switch off debug logging after the data has been collected.

Related concepts

“Indexes” on page 151

Related reference

“catalog.indexes table” on page 348

“SQL query guidelines”

“Indexing guidelines” on page 314

“Best practices for creating triggers” on page 317

Track the performance trends at regular intervals

As time progresses, various factors could affect the performance of the ObjectServer. Establish a performance baseline and periodically monitor performance to compare the metrics against those collected previously. Investigate significant differences above or below the baseline and fine tune as required.

SQL query guidelines

When an SQL query is passed to the ObjectServer, query optimization and query plans are used to evaluate the available methods for accessing or modifying the data, and to select the most efficient way to run the query.

The evaluation determines whether index scans or primary keys can be used, or whether a full table scan must be performed instead. If index scans are used, indexes are automatically applied to SQL queries that reference indexed columns.

Optimization rules for SQL queries

A set of optimization rules are applied to SQL queries to determine the most efficient way to execute the queries. When optimization occurs, all of the optimization rules are applied to the condition in the query and then the predicates are reordered.

The following WHERE clauses are optimized:

- The WHERE clause in DML operations (SELECT, UPDATE, DELETE, GROUP BY, and aggregate SELECT)

The WHERE clause is concatenated with the restriction filter (where present) and then optimized. For a view, the WHERE clause is concatenated with the WHERE clause of the view.

- The WHERE clause of a FOR EACH ROW loop
- The WHERE clause of an IF THEN ELSE statement
- The WHERE clause of an EVALUATE statement (GROUP BY, SELECT, and aggregate SELECT)

The following types of queries are not optimized:

- A HAVING clause in a GROUP BY command
- Column in array

Tip: Query optimization is enabled by default. To review the optimization results, use the **MessageLevel** ObjectServer property to set the logging level to debug. Optimized and unoptimized clauses are logged to the default `$NCHOME/omnibus/log/servername.log` file, where *servername* is the ObjectServer name. Review this information to see which queries are being rewritten by the ObjectServer to be more efficient, and use it as an aid to improve the efficiency of the SQL queries that you write.

AND and OR optimization, and the reordering of predicates are applied to SQL queries as follows.

AND optimization

If a WHERE clause consists of multiple predicates that are connected by the AND operator, optimization occurs only if the same column and comparison operator are used in every predicate, and the comparison operator is LIKE or equal to (=).

The acceptable format for AND optimization is:

where column operator expr1 AND column operator expr2 AND column operator expr3...

In this WHERE clause, *operator* is LIKE or = only.

This format is optimized into an ALL list as follows:

where column operator ALL (expr1, expr2, expr3, ...)

Example 1

Original SQL query:

where Node like 'ibm' and Node like 'com'

Optimized query:

where Node like all('ibm','com')

Example 2

Original SQL query:

where Node like 'ibm' and Node like all('com','uk')

Optimized query:

where Node like all('ibm','com','uk')

Example 3

Original SQL query:

where Node like all('ibm','com') and Node like all('uk','london')

Optimized query:

where Node like all('ibm','com','uk','london')

OR optimization

If a WHERE clause consists of multiple predicates that are connected by the OR operator, optimization occurs only if the same column and comparison operator are used in every predicate, and the comparison operator is LIKE or equal to (=).

The acceptable format for OR optimization is:

where column operator expr1 OR column operator expr2 OR column operator expr3...

In this WHERE clause, *operator* is LIKE or = only.

This format is optimized into an ANY list as follows:

where column operator ANY (expr1, expr2, expr3, ...)

Example 1

Original SQL query:

where Node like 'London' or Node like 'Copenhagen'

Optimized query:
where Node like any('London', 'Copenhagen')

Example 2

Original SQL query:
where Severity = 1 or Severity = any(2,3)

Optimized query:
where Severity = any(1,2,3)

Example 3

Original SQL query:
where Severity = any(1,2) or Severity = any(3,4)

Optimized query:
where Severity = any(1,2,3,4)

Reordering of predicates

The optimizer reorders the evaluation of the predicates in a WHERE clause according to their assigned execution cost.

If the first predicate (that is, the cheapest) in an OR optimization evaluates to TRUE, the more expensive predicates (that is, any that follow) do not have to be evaluated. Similarly, in an AND optimization, if the first predicate evaluates to FALSE, the more expensive predicates do not have to be evaluated.

The assigned execution cost, from lowest to highest, is:

1. True/False
2. Integer comparison
3. String comparison
4. Integer ANY/ALL/IN
5. String ANY/ALL/IN
6. Subselect - that is a nested SELECT statement

Example: AND optimization (a AND b AND c)

Original SQL query:
where Summary like 'tool' and Serial in (1, 2, 3, 4, 5) and Severity > 2

Optimized reordered query:
where Severity > 2 and Summary like 'tool' and Serial in (1, 2, 3, 4, 5)

Example: OR optimization (a OR b OR c)

Original SQL query:
where Summary like 'tool' or Serial in (1, 2, 3, 4, 5) or Severity > 2

Optimized reordered query:
where Severity > 2 or Summary like 'tool' or Serial in (1, 2, 3, 4, 5)

Indexing guidelines

Indexing can affect the performance of your SQL queries. Without indexing, a full database table scan is typically performed when an SQL query runs. Use indexing to limit the number of rows that are examined.

Tivoli Netcool/OMNIBus supports hash and tree index structures. The hash index supports equality comparisons in SQL queries. The tree index is an ordered index that stores column values in a sorted structure, and allows a wider range of comparisons, including equality, in SQL queries. Consequently, a tree index can be used in range queries and in queries with an ORDER BY clause.

Indexes are rebuilt whenever the ObjectServer is restarted, and these indexes use up a small amount of memory rather than physical disk space.

You can create indexes on all ObjectServer tables except the tables in the system databases, such as the catalog and security databases.

Although there is no limit on the number of indexes that you can create on a table, you must use indexes sparingly. Indexes incur a performance overhead because they are updated when insert, update, or delete operations are performed on the table on which they are based. For tables such as alerts.status, which are updated frequently, creating a large number of indexes can adversely affect the overall performance of the ObjectServer. Evaluate the tradeoff between indexing for fast retrieval of data and the performance degradation during insert, update, and delete operations.

Avoid indexing tables that contain only small amounts of data. Also avoid indexing columns that contain data values which frequently change.

The following columns are considered good candidates for indexing:

- Columns that are searched or sorted against frequently; that is, columns typically used in ORDER BY clauses
- Columns that are frequently used in WHERE clauses that contain the predicate formats supported for indexing; see Table 90 on page 315
- Columns with data that contains few duplicate values

Columns that are defined as primary keys are, by default, uniquely indexed. These special, implicit indexes are not stored in the catalog.indexes table. The Serial column in the alerts.status table is indexed by default.

Indexing restrictions on columns are as follows:

- Only one index per column is allowed.
- A column that is defined as Boolean cannot have a tree index.

During SQL processing, both the restriction filter for the table and the WHERE clause in each SELECT, UPDATE, DELETE, FOR EACH ROW, and EVALUATE statement is examined to determine whether an index scan should be performed instead of a full table scan. An index scan is performed when one or more predicates fulfill the following conditions:

- The predicate uses the equality operator (=) in the format *ColumnName* = *ConstantExpression*, where *ColumnName* is an indexed column.
- The predicate uses the less than (<), less than or equal to (<=), greater than (>), or greater than or equal to (>=) operator, providing *ColumnName* is an indexed column of type tree.

- The predicate is not connected to another predicate by an OR operator. For example, if the Severity or Serial field is indexed, an index is not used in the following SQL query:

```
select Summary from alerts.status where Severity > 3 or Serial = 102;
```

The following table summarizes which predicate formats are supported for hash and tree indexes.

Table 90. Predicate formats for hash and tree indexes

Predicate format	Hash index	Tree index
<i>ColumnName = ConstantExpression</i>	Yes	Yes
<i>ColumnName < ConstantExpression</i>	No	Yes
<i>ColumnName > ConstantExpression</i>	No	Yes
<i>ColumnName <= ConstantExpression</i>	No	Yes
<i>ColumnName >= ConstantExpression</i>	No	Yes
<i>ColumnName %= ConstantExpression</i>	No	No
<i>ColumnName %< ConstantExpression</i>		
<i>ColumnName %> ConstantExpression</i>		
<i>ColumnName %<= ConstantExpression</i>		
<i>ColumnName %>= ConstantExpression</i>		

Related concepts

“Indexes” on page 151

Related reference

“Example usage of indexes with SQL queries”

“Example usage of indexes with triggers or procedures” on page 316

Example usage of indexes with SQL queries

These examples show how indexes can be applied to SQL queries.

Example 1

If the Severity or Serial field is indexed, the index on the Severity field (providing it is a tree index) can be used in the following SQL query because all the rows will have to meet the expression `Severity > 3`. The index on the Serial field is not used because an OR operator is used to connect two predicates.

```
select Summary from alerts.status where
    Severity > 3 and (Serial = 102 or ServerName = 'NCOMS');
```

Example 2

If a tree index is created on the Severity field, the index on the Severity field can be used in the following SQL query. However, an index on LastOccurrence cannot be used because of the OR operator between the `LastOccurrence > getdate() - 360` predicate and the `Summary like 'LinkUp'` predicate. Note, however, that the expression `getdate() - 360` is considered constant for the duration of the query.

```
select Summary, Severity, Node from alerts.status where
    Severity > 1 and (LastOccurrence > getdate() - 360 or Summary like 'LinkUp')
```

Example 3

Consider the following query:

```
select Summary from alerts.status where Severity > 0;
```

For a comparison operator like $>$, $>=$, $<$, or $<=$ to be used with an index, a tree index (which is an ordered index) is required. If only 100 rows out of 20,000 have Severity 0, such an index will reduce the number of rows examined by only 0.5%, and will not provide a significant performance benefit. Therefore, the actual row data must be taken into account to decide which column to index.

Example 4

If a hash index is created on the Node column, when the following SQL query runs, only three hash lookups are performed for tool, bar, and toolbar instead of examining each row for equality of Node and one of the three values.

```
select Identifier from alerts.status where Node in ('tool', 'bar', 'toolbar');
```

Example 5

If a tree index is created on the Severity column, when the following SELECT statement is processed, only Severity values 2 and 3 are searched for and returned.

```
select * from alerts.status where Severity > 1 and Severity < 4;
```

If an ORDER BY clause includes more than one column, an index is used for the first column, if available.

```
select Identifier, Serial from alerts.status order by Severity;
```

Example 6

If there are 20,000 rows in the alerts.status table, and an index on the ServerSerial field is applied to the following query, only two rows are examined instead of 20,000:

```
select Summary from alerts.status where ServerSerial in (102,103);
```

Example usage of indexes with triggers or procedures

This example shows how indexes can be applied to triggers or procedures. The example is a correlation between two types of events, Type 14 and Type 15, such that if they both occur on the same host, they are cleared.

```
create procedure correlation
begin
  for each x in alerts.status where Type = 14
  begin
    for each y in alerts.status where y.Node = x.Node
      and y.AlertGroup = x.AlertGroup and Type = 15
    begin
      update alerts.status set Severity = 0
      where Identifier in (y.Identifier, x.Identifier)
    end
  end
end;
```

In this example, any event of Type 14 and 15 will be cleared if both exist for the same node and AlertGroup.

If there are 10,000 events of Type 14 and 10,000 events of Type 15, and there are, on average, 10 events per unique Node, the following results are possible:

- Without indexing, the inner WHERE clause will scan over 10,000 * 20,000 rows; that is, 200 million rows. This will be slow, and is the reason why nested FOR EACH ROW statements are not advisable without good indexing.
- With an index on the Type column, the inner WHERE clause will scan 10,000 * 10,000 rows; that is, 100 million rows. This will be slow, but is half the number of rows scanned when indexing is not used.
- With an index on the Node column, the inner WHERE clause will scan 10,000 * 10 rows; that is, 100,000 rows. This will be performant.

Note: The scan performed by the UPDATE statement will always use the primary key.

Best practices for creating triggers

The overriding goal when creating or modifying triggers should be to make the triggers as efficient as possible, with the shortest possible execution time.

A trigger has exclusive access to the ObjectServer database for the duration of its execution. By minimizing the execution time of a trigger, you can free up time for other triggers or clients that require access to the database. It is particularly important to reduce the execution time of database triggers because they interrupt the execution of a database operation, thereby slowing down the operation. For example, a pre-insert trigger on the alerts.status table will fire for every new event, so if an event flood occurs, the trigger will be executed multiple times. The degree of efficiency of the trigger will affect the ability of the system to cope.

The ObjectServer records the amount of time that each trigger uses during each granularity period and saves the details in the `$NCHOME/omnibus/log/servername_trigger_stats.logn` file. You can use this file to identify which triggers are using the most time, to prioritize which triggers to review, and to monitor the system to ensure that it is running as expected. In general, if a single trigger is using more than 3 seconds of time every 60 seconds (that is, the default granularity period), the trigger should be reviewed.

Whenever you update your triggers, review the log file to verify that your changes do not cause a degradation in performance.

Use the following guidelines to improve the performance of your triggers.

Avoid table scans in database triggers

Table scans are expensive operations and can occur when SQL statements such as FOR EACH ROW are applied to a database table. When such statements are included in a database trigger, the cost can be particularly significant if the trigger is invoked frequently, and if the table being scanned has a large number of rows. For example, if the deduplication trigger on the alerts.status table is modified so that every time the trigger fires it scans alerts.status for rows matching a set of criteria, this will limit the scalability of the system because the database trigger will take increasing amounts of time as the number of rows in the table being scanned increases. Also avoid nested scans.

You can use the following techniques to avoid the table scan in database triggers:

- Perform the scan in a temporal trigger that is written so that one scan can match many rows. See the generic_clear trigger in \$NCHOME/omnibus/etc/automation.sql for an example.
- If using a lookup table to enrich events, access the lookup table by using its primary key, as described further on. The use of the primary key results in a direct lookup of the row rather than a scan (V7.2, or later). You can also limit the size of the lookup table. The number of rows that are acceptable for a lookup table is site specific, and will depend on factors such as how often the lookup table is accessed, and hardware performance.
- Access a lookup table by using an index.

Avoid using the EVALUATE clause

When a trigger contains an EVALUATE clause, a temporary table is created to hold the results of the SELECT statement in the EVALUATE clause. The amount of time and resources that this temporary table consumes depends on the number of columns being selected and the number of rows matched by the condition in the WHERE clause.

In most cases, you can replace the EVALUATE clause with a FOR EACH ROW clause, which cursors over the data and does not incur the overhead of creating a temporary table.

A suitable use for an EVALUATE clause is when a GROUP BY clause is being applied to an SQL query.

Avoid excessive use of the WRITE INTO statement for logging out to file

The WRITE INTO statement is very useful for several purposes; in particular, for debugging triggers during development. However, when a trigger is being deployed in a production environment, it is advisable to comment out or remove WRITE INTO statements, because the quantity of data that is logged during debugging can create a bottleneck.

Determine what is suitable for your system. For example, if the logging is infrequently called, there is probably no issue. However, if logging is called multiple times per INSERT statement (for example, within a nested loop), there could be a bottleneck.

Where possible, use the primary key when modifying rows

If the primary key of a database table is used in the WHERE clause of an UPDATE statement, the row is accessed by using direct lookup, rather than a table scan. For example:

```
update alerts.status where Identifier = tt.Identifier set Severity = Severity + 1;
```

Note: The VIA keyword is no longer required in V7.2, or later. The following command (which uses VIA) is equivalent to the preceding command:

```
update alerts.status VIA Identifier = tt.Identifier set Severity = Severity + 1;
```

Use indexes when using lookup tables

In V7.2, or later, the ObjectServer uses an index to access rows in a table if the primary key is used in a FOR EACH ROW statement.

This is most useful where an ObjectServer table is being used as a lookup table, perhaps to enrich events. In such a case, the table and triggers that access the lookup table should be designed to access the lookup table by its primary keys to prevent costly full table scans. For example:

```
create table alerts.iplookup persistent
(
  IpAddr  varchar(32) primary key,
  HostName varchar(8),
  Owner   varchar(40)
);

create or replace trigger set_hostname
group madeup_triggers
priority 10
before insert on alerts.status
for each row
begin
  -- Access the lookup table using the primary key
  for each row tt in alerts.iplookup where tt.IpAddr = new.Node
  begin
    set new.Hostname = tt.HostName;
  end;
end;
```

Review and modify triggers produced from migrating from V3.6

If you have migrated from V3.6 to V7.2.1, the migration tool in V7.2.1 will produce best-effort replications of the V3.6 triggers. When you then upgrade to V7.3.1, the triggers, which were initially migrated from V3.6, will be functionally correct, but are unlikely to be the most performant implementation of the triggers. Review and modify these triggers as follows:

- The V3.6 ObjectServer supported only temporal triggers, while V7.0 or later, includes database and signal triggers. The processing that is performed by a temporal trigger in V3.6 might be better suited to a database trigger in V7.0 or later. However, the migration tool converts triggers only on a like-for-like basis, so you must review the migrated triggers to identify which triggers can be better implemented by using the new trigger types.
- Where V3.6 triggers have the condition `select *`, the migration tool implements the condition as an `EVALUATE` clause, where all the columns in the `alerts.status` table are selected. Where possible, replace the `EVALUATE` clause with a `FOR EACH ROW` statement.
- When migrating from V3.6, the migration tool also creates generic clear triggers that work in the same manner as in V3.6. However, the triggers supplied in V7.0, or later, are more efficient. Therefore, it is advisable to use the V7.0 or later triggers, which are by default disabled, instead of using the triggers migrated from V3.6.

Use the generic_clear trigger as a basis for correlation type triggers

The standard `generic_clear` trigger (see `$NCHOME/omnibus/etc/automation.sql`) correlates resolution events with their related problem events. After this trigger runs, all matched rows have their severity set to 0, in readiness for removal by the `delete_clears` automation.

The `generic_clear` trigger has several key design features that should be duplicated if a different type of correlation trigger is required. These features include the manner in which the events are identified and how they are updated.

The standard generic_clear trigger does not use the EVALUATE clause to select the events; instead it uses the FOR EACH ROW construct to loop over the events to populate a temporary table with the problem events. Because this temporary table contains only a subset of the events in the alerts.status table, the cost of the update operation that is applied to relate the problems with resolutions is reduced. Additionally, because the identifier of the problem event is stored in the temporary table, the problem events can be updated directly in alerts.status by using the UPDATE VIA command to perform a direct lookup on the row; this takes advantage of the Identifier field being a primary key.

Use deduplication to clear events where possible

The deduplication trigger can be used to clear problem events with the incoming resolution event when there is a one-to-one mapping between the problem and resolution. The following modification is required to your existing system:

- Write the probe rules so that the problem and resolution events have the same identifier.
- Modify the deduplication trigger so that when it fires, it checks the Type field. If the type of the incoming event is set to 1 (resolution), the severity of the existing event should be set to 0.

The benefit of this approach is that it reduces the amount of processing that the generic_clear trigger has to perform, leaving it with the task of resolving the cases where a single resolution event will clear many problem events.

Notes on creating automations

Use the following best practice guidelines to help create any new automations:

- Confirm whether or not an automation currently exists before attempting to create a new automation with the same function.
- In any WHERE conditions within the automation, use the guidelines for reordering of predicates in SQL queries. For example, compare integers, compare characters, and then compare regular expressions. For more information, see “Optimization rules for SQL queries” on page 311.
- Ensure that the automation trigger does not acquire events which have previously been processed. This is especially important for external scripts.
- For temporal triggers, set the firing interval of different triggers to prevent them from being activated together.
- Add a description to all newly created automations.
- Automations can update the journal entry if they modify events in the ObjectServer database.

Test your changes

After new triggers are developed and validated, test the performance of the triggers as follows:

1. Ensure that the data on which you run the tests is representative of the production system.
2. Ensure that the number of rows in any table that the trigger accesses is representative of the production system.
3. Measure the effect on system performance by using profiling and by collecting trigger statistics.

Appendix A. ObjectServer tables

The ObjectServer database contains the following tables: alerts tables, service tables, system catalog tables, statistics tables, client tool support tables, desktop tools tables, desktop ObjectServer tables, security tables, IDUC channel tables, and service-affected events tables.

The ObjectServer database tables are stored in \$NCHOME/omnibus/db on UNIX systems and %NCHOME%\omnibus\db on Windows systems.

Alerts tables

Alert information is forwarded to the ObjectServer from external programs such as probes and gateways. This information is stored and managed in database tables, and displayed in the event list.

alerts.status table

The alerts.status table contains status information about problems that have been detected by probes.

The following table describes the columns in the alerts.status table.

Table 91. Columns in the alerts.status table

Column name	Data type	Mandatory	Description
Identifier	varchar(255)	Yes	<p>Controls ObjectServer deduplication. The Identifier field controls the deduplication feature of the ObjectServer, and also supports compatibility with the GenericClear automation by ensuring resolution events are properly inserted into the ObjectServer and not deduplicated with their respective problem events.</p> <p>The following identifier correctly identifies repeated events in a typical environment:</p> <pre>@Identifier=@Node+ " "+@AlertKey+" "+@AlertGroup+ " "+@Type+ " "+@Agent+ "+@Manager</pre> <p>Additional information might need to be appended to the Identifier field to ensure correct deduplication and compatibility with the GenericClear automation. For example, if an SNMP specific trap contains a status enumeration value in one of its variable bindings, the specific trap number and the value of the relevant varbind must be appended to the Identifier field as follows:</p> <pre>@Identifier=@Node + " "+ @AlertKey+" "+@AlertGroup+ " "+@Type+ " "+@Agent+ "+@Manager+ " "+\$specific-trap+ "+\$2</pre>
Serial	incr	Yes	The Tivoli Netcool/OMNIBus serial number for the row.

Table 91. Columns in the *alerts.status* table (continued)

Column name	Data type	Mandatory	Description
Node	varchar(64)	Yes	<p>Identifies the managed entity from which the alarm originated. This could be a device or host name, service name, or other entity.</p> <p>For IP network devices or hosts, the Node column contains the resolved name of the device or host. In cases where the name cannot be resolved, the Node column must contain the IP address of the device or host.</p> <p>For non-IP network devices or hosts, alarms must contain similar information to the IP device or host. That is, the Node column must contain the name of the device or host which allows direct communication, or can be resolved to allow direct communication, with the device or host.</p>
NodeAlias	varchar(64)	No	<p>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 must be the IP address.</p> <p>For non-IP devices or hosts, there are several addressing schemes that could be used. When selecting a value for the NodeAlias field, the value should allow for direct communication with the device or host. For example, a device managed by TL-1. The NodeAlias field may be populated by a lookup table or Netcool/Impact policy, with the IP address and port number of the terminal server through which the TL-1 device can be reached.</p>
Manager	varchar(64)	Yes	<p>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. Ideally this is set in the properties file of the probe, however the rules file should check to ensure it is set correctly, and modify if necessary.</p> <p>For example, the following syntax can be used to define the Manager field:</p> <pre>@Manager="MTTrapd Probe on" + hostname()</pre>

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
Agent	varchar(64)	No	<p>The descriptive name of the sub-manager that generated the alert.</p> <p>Probes which process SNMP traps must set the Agent field to either the name of the vendor or the standards body which defined the trap, and provide a description of the MIB, or MIB Definition Name, where the trap is defined. It must be presented in the following format: vendor-MIB description</p> <p>For example::</p> <p>Cisco-Accounting Control, Cisco-Health Monitor, IETFBRIDGEMIB, ATMF-ATM-FORUM-MIB</p> <p>Optionally, vendor-specific information, such as device model numbers, can be appended to the Agent field for vendor-specific implementations of standard traps.</p> <p>The Syslog probe should set the Agent field to the name of the vendor which defined the received message, and provide any logical description for the family of messages to which the received message belongs.</p> <p>For example, Cisco defines messages received from IOS-based devices in separate documentation from messages received from the PIX Firewall. The format of the messages is also slightly different. Therefore the Syslog messages received from Cisco will have the Agent field set to either Cisco-IOS or Cisco- PIX Firewall.</p> <p>The TL-1 TSM should set the Agent field to the name of the vendor which defined the received message, and provide any logical description for the family of messages to which the received message belongs.</p>
AlertGroup	varchar(255)	No	<p>The descriptive name of the failure type indicated by the alert. For example:</p> <p>Interface Status or CPU Utilization).</p> <p>The AlertGroup field must contain the same value for related problem and resolution events.</p> <p>For example, SNMP trap 2 (linkDown) and trap 3 (linkUp) must both contain the same AlertGroup value of Link Status.</p> <p>The AlertGroup field for a TL-1 message will be set to the value of the message's alarm type.</p>

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
AlertKey	varchar(255)	Yes	<p>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.</p> <ul style="list-style-type: none"> SNMP Trap-related probes: Probes that process SNMP traps should set the AlertKey field to one of the following values (in order of preference): <ul style="list-style-type: none"> The SNMP instance of the managed object which is represented by the alarm. This is normally obtained by extracting the instance from the OID of one of the variable bindings of the trap. Additionally, it might also be contained in a combination of one or more of the trap's variable binding values. For example, the first variable binding of a linkDown trap contains the ifIndex value (interface number) of the interface which failed. The AlertKey can be set with either of the following: <pre>@AlertKey = extract(\$OID1, "\.([0-9]+)\$") @AlertKey = \$1</pre> A textual description of the instance derived from the trap name or trap description. For example, a device with two power supplies (A and B) might be able to send two separate specific traps, without variable bindings, to indicate the failed status of either power supply. The appropriate power supply instance would need to be derived from the trap definitions of the MIB and then encoded in the rules file: <pre>switch(\$specific-trap) { case "1": @AlertKey = "A" case "2": @AlertKey = "B" default: }</pre> A mixed combination of variable binding values and information derived from the trap name or trap description. Therefore, any instance information that is not available in the previous two methods, but is required to ensure correct deduplication and GenericClear compatibility.

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
			<ul style="list-style-type: none"> The Syslog Probe: The Syslog Probe should set the AlertKey to a textual description of the instance derived from the log message text. Ideally this is a textual name of the same managed entity. For example: Nov 20 13:12:57 device.customer.net 195.180.208.193 19986: 37w0d: %LINK-3-UPDOWN: Interface FastEthernet0/13, changed state to down In the previous example, the AlertKey would be set to FastEthernet0/13 using the following syntax: @AlertKey = extract(\$Details, "Interface (.*), changed") TL-1 TSM: Typically the AlertKey field for a TL-1 message is set to the value of the message's alarm location.

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
Severity	integer	Yes	<p>Indicates the alert severity level, which indicates 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:</p> <p>0: Clear. The Clear severity level indicates the clearing of one or more previously reported alarms. The alarms have either been cleared manually by a network operator, or automatically by a process which has determined the fault condition no longer exists. Automatic processes, for example the GenericClear Automation process, typically clear all alarms for a managed object (the AlertKey) that have the same Alarm Type and/or probable cause (the Alert Group).</p> <p>1: Indeterminate. The Indeterminate severity level indicates that the severity level cannot be determined. Additionally, all problem resolving alarms are initially defined as indeterminate until they have been correlated with problem indicating alarms (for example by the GenericClear Automation), when all correlated alarms are set to Clear.</p> <p>2: Warning. The Warning severity level indicates the detection of potential or impending service affecting faults. If necessary, a further investigation of the fault should be made to prevent it from becoming more serious.</p> <p>3: Minor. The Minor severity level indicates the existence of a non-service affecting fault condition. Corrective action should be taken to prevent it from becoming a more serious fault. This severity level may be reported, for example, when the detected alarm condition is not currently degrading the capacity of the managed object.</p> <p>4: Major. The Major severity level indicates that a service affecting condition has developed and corrective action is urgently required. This severity level may be reported, for example, when there is a severe degradation in the capability of the managed object, and its full capability must be restored.</p> <p>5: Critical. The Critical severity level indicates that a service affecting condition has occurred, and corrective action is immediately required. This severity level may be reported, for example, when a managed object is out of service, and its capability must be restored.</p>

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
Summary	varchar(255)	Yes	<p>Contains text which describes the alarm condition and the affected managed object instance.</p> <ul style="list-style-type: none"> You must ensure that the information presented in the Summary field is concise and sufficiently detailed. The Summary field must contain, in parenthesis, a description of the managed object instance provided by the available alarm data. For example, a linkDown trap from a Cisco device will contain the ifDescr value in the 2nd variable binding. The text summary of such an event would be similar to: "Link Down (FastEthernet0/13)" For alarms that relate to thresholds containing the compared or threshold values, you should select one of the following formats based on the available data: <ul style="list-style-type: none"> No values provided: "Link Utilization High (BRI2/0:1)" Compared value name provided: "Link Utilization High: inOctets Exceeded Threshold (BRI2/0:1)" Compared value name and value provided: "Link Utilization High: inOctets, 7100, Exceeded Threshold (BRI2/0:1)" Threshold name provided: "Link Utilization High: inOctetsMax Exceeded (BRI2/0:1)" Threshold Value provided: "Link Utilization High: inOctetsMax, 7000, Exceeded (BRI2/0:1)" Compared value and threshold value provided: "Link Utilization High: 7100 Exceeded 7000 (BRI2/0:1)" Both names and values provided: "Link Utilization High: inOctets, 7100, Exceeded inOctetsMax,7000 (BRI2/0:1)"
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 January 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 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
Type	integer	No	<p>The type of alarm, where type refers to the problem or resolution state of the Alarm. This field is important for the correct correlation of events by the GenericClear Automation. The following values are valid for the Type field:</p> <p>0: Type not set</p> <p>1: Problem</p> <p>2: Resolution</p> <p>3: Netcool/Visionary problem</p> <p>4: Netcool/Visionary resolution</p> <p>7: Netcool/ISMs new alarm</p> <p>8: Netcool/ISMs old alarm</p> <p>11: More Severe</p> <p>12: Less Severe</p> <p>13: Information</p> <p>Some scenarios cannot be categorized as either a Problem or Resolution. For example, events which are increasingly becoming an issue but do not currently represent a failure, and events which are becoming less of an issue but do not currently indicate the failure has been completely resolved. In which case, the Type field must be set to Problem, More Severe or Less Severe to maintain compatibility with the GenericClear Automation.</p> <p>For example, the following rule file logic is used for handling traps associated with BGP Peer Connection Status:</p> <pre>switch (\$bgpPeerState) { case "1": ### idle @Severity = 4 @Type = 1 case "2": ### connect @Severity = 2 @Type = 12 case "3": ### active @Severity = 2 @Type = 12 case "4": ### opensent @Severity = 2</pre>

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
			<pre>@Type = 12 case "5": ### openconfirm @Severity = 2 @Type = 12 case "6": ### established @Severity = 1 @Type = 2 default: @Severity = 2 @Type = 1 }</pre>
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.
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.
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(255)	No	The event ID (for example, SNMPTRAP-link down). Multiple events can have the same event ID. The event ID is populated by the probe rules file and used by IBM Tivoli Network Manager IP Edition.
ExpireTime	integer	Yes	The number of seconds from the time this alert was last received by the ObjectServer (LastOccurrence) until it is cleared automatically. Used by the Expire automation.

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
ProcessReq	integer	No	Indicates whether the alert should be processed by IBM Tivoli Network Manager IP Edition. This is populated by the probe rules file and used by IBM Tivoli Network Manager IP Edition.
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.
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 the event that is suppressing the current event. Populated by IBM Tivoli Network Manager IP Edition.
NmosObjInst	integer	No	Populated by IBM Tivoli Network Manager IP Edition during alert processing.
NmosCauseType	integer	No	The alert state, populated by IBM Tivoli Network Manager IP Edition as an integer value: 0: Unknown 1: Root cause 2: Symptom

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
NmosDomainName	varchar(64)	No	<p>The name of the IBM Tivoli Network Manager IP Edition domain that is managing the event.</p> <p>By default, this column is populated only for events that are generated by IBM Tivoli Network Manager IP Edition polls. To populate this column for other event sources such as probes, you must modify the rules files.</p>
NmosEntityId	integer	No	<p>A unique numerical ID that identifies the IBM Tivoli Network Manager IP Edition topology entity with which the event is associated.</p> <p>This column is similar to the NmosObjInst column, but is more granular. For example, the NmosEntityId value can represent the ID of an interface within a device.</p>
NmosManagedStatus	integer	No	<p>The managed status of the network entity for which the event was raised. Can apply to events from IBM Tivoli Network Manager IP Edition and from any probe.</p> <p>You can use this column to filter out events from interfaces that are not considered relevant.</p>
NmosEventMap	varchar(64)	No	<p>Contains the required IBM Tivoli Network Manager IP Edition V3.9 or later, eventMap name and optional precedence for the event, which indicates how IBM Tivoli Network Manager IP Edition should process the event.</p> <p>The optional precedence number can be concatenated to the end of the value, following a period (.). If the precedence is not supplied, it is set to 0. The following examples show the configuration for an event map with an explicit event precedence of 900, and another where the precedence defaults to 0:</p> <ul style="list-style-type: none"> • ItnmLinkdownIfIndex.900 • PrecisionMonitorEvent
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 91. Columns in the alerts.status table (continued)

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.
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.

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
ExtendedAttr	varchar(4096)	No	<p>Holds name-value pairs (of Tivoli Enterprise Console® extended attributes) or any other additional information for which no dedicated column exists in the alerts.status table.</p> <p>Use this column only through the nvp_get, nvp_set, and nvp_exists SQL functions.</p> <p>An example of a name-value string is: Region="EMEA";host="sf01392w"; Error="errno=32: ""Broken pipe"""</p> <p>In this example, the Region attribute has a value of EMEA, the host attribute has a value of sf01392w, and the Error attribute has a value of errno=32: "Broken pipe".</p> <p>Notice that quotation marks are escaped by doubling them, as shown with the Error attribute value.</p> <p>In name-value pairs, the value is always enclosed in quotation marks (" ") and embedded quotation marks are escaped by doubling them. The separator between name-value pairs is a semicolon (;). No whitespace is allowed around the equal sign (=) or semicolon.</p> <p>Note: The column can hold only 4096 bytes, so there will be fewer than 4096 characters if multi-byte characters are used.</p>
OldRow	integer	No	<p>Maintains the local state of the row in each ObjectServer during resynchronization in the failover pair. This column must not be added to the gateway mapping files.</p> <p>The value of OldRow is changed to 1 in the destination ObjectServer for the duration of resynchronization if the Gate.ResyncType property of the gateway is set to Minimal.</p> <p>The default is 0.</p>
ProbeSubSecondId	integer	No	<p>For those alerts that a probe sends within the same one-second interval, and which therefore have the same LastOccurrence value, an incremental value, starting at 1, is added to the ProbeSubSecondId field to differentiate the LastOccurrence time. The default is 0.</p>
MasterSerial	integer	No	<p>Identifies the master ObjectServer if this alert is being processed in a desktop ObjectServer environment.</p> <p>This column is added when you run the database initialization utility nco_dbinit with the -desktopserver option.</p> <p>Note: MasterSerial must be the last column in the alerts.status table if you are using a desktop ObjectServer environment.</p>

Table 91. Columns in the alerts.status table (continued)

Column name	Data type	Mandatory	Description
BSM_Identity	varchar(1024)	No	The unique identifier of the resource from where the event originates, and is used to correlate the event to that resource in IBM Tivoli Business Service Manager (TBSM).

Note: You can display only 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.

Related concepts

“Functions” on page 165

alerts.details table

The alerts.details table contains the detail attributes of the alerts in the system.

The following table describes the columns in the alerts.details table.

Table 92. Columns in the alerts.details table

Column name	Data type	Description
KeyField	varchar(255)	<p>Internal sequencing string for uniqueness.</p> <p>The Keyfield value is composed of an Identifier value plus four # plus a sequence number starting at a count of 1; for example:</p> <p><i>Identifier####1</i></p> <p>Where <i>Identifier</i> is a data type of varchar(255), which is used to relate details to entries in the alerts.status table.</p> <p>If the Identifier value is over a certain length, there is a possibility that the Keyfield value could exceed its defined 255 limit, resulting in truncation of the sequence number. Keyfield values could therefore no longer be unique, and the unintended duplication could cause inserts into the alerts.details table to fail.</p> <p>Tip: To prevent an overflow in KeyField (and ensure uniqueness), the length of the Identifier value must be sufficiently less than 255 to allow the four # and a sequence number (of one or more digits) to be appended.</p>
Identifier	varchar(255)	<p>Identifier to relate details to entries in the alerts.status table.</p> <p>The Identifier is used to compute the Keyfield value, and is required to be less than a certain length to ensure that each computed Keyfield value remains unique. For guidelines on the maximum length of the Identifier value, see the tip in the preceding KeyField row.</p>
AttrVal	integer	Boolean; when false (0), just the Detail column is valid. Otherwise, the Name and Detail columns are both valid.
Sequence	integer	Sequence number, used for ordering entries in the event list Event Information window.
Name	varchar(255)	Name of attribute stored in the Detail column.
Detail	varchar(255)	Attribute value.

alerts.journal table

The alerts.journal table provides a history of work performed on alerts.

The following table describes the columns in the alerts.journal table.

Table 93. Columns in the alerts.journal table

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.
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.

alerts.iduc_messages table

The alerts.iduc_messages table is required for multicultural support and is used to send insert, delete, update, or control (IDUC) client messages. This table ensures that characters transferred across varying encodings are converted into recognizable formats.

The following table describes the columns in the alerts.iduc_messages table.

Table 94. Columns in the alerts.iduc_messages table

Column name	Data type	Description
MsgID	integer	Primary key for table.
MsgText	varchar(4096)	Message text sent to an event list by the nco_elct utility. This utility enables you to open a customized, transient event list.
MsgTime	time	Time the message was sent.

alerts.application_types table

The alerts.application_types table contains details about application types that cause connection watch messages to be generated when the applications connect and disconnect. Use this table to configure the severity of connection and disconnection events by application type.

For example, a gateway connection is treated as a resolution (clearing a disconnect), whereas an event list connect is a Type 1 event, which is resolved by a disconnection. A gateway disconnection is treated as a problem, whereas an event list disconnection is a resolution. You can use the alerts.application_types table to configure a gateway to generate a Type 1 event (warning) on disconnection and a Type 2 event (disconnection cleared) on connection, and configure an event list to generate a Type 1 event on connection and a Type 2 event (clear) on disconnection.

The alerts.application_types table is read by the connection_watch_connect and connection_watch_disconnect triggers.

You can add a new application type to this table by adding a row, if required. You can also change the behavior of an application by updating its row.

The following table describes the columns in the alerts.application_types table.

Table 95. Columns in the alerts.application_types table

Column name	Data type	Description
application	varchar(64)	Primary key for the table. This is the internal application name specified as a regular expression for efficient string matching.
description	varchar(64)	Descriptive name for the event.
connect_type	int	Event type for the connection.
connect_severity	int	Event severity for the connection.
disconnect_type	int	Event type for the disconnection.
disconnect_severity	int	Event severity for the disconnection.
expire_time	int	Number of seconds until the alert is cleared automatically. Used by the Expire automation.
discard	Boolean	Set to TRUE if the event is to be suppressed.

master.class_membership table

The master.class_membership table supports the mapping of Tivoli Enterprise Console classes to Tivoli Netcool/OMNIBus classes, and stores class membership information. This table is used with the instance_of() SQL function.

The following table describes the columns in the master.class_membership table.

Table 96. Columns in the master.class_membership table

Column name	Data type	Description
Class	integer	Primary key for table. Class number as stored in the alert.status table.
ClassName	varchar(255)	Name of class.

Table 96. Columns in the master.class_membership table (continued)

Column name	Data type	Description
Parent	integer	Primary key for table. Parent ID of class. The root class has a parent ID of -1. If a class has multiple parents, a row exists for each parent.

The master.class_membership table does not permit duplicate mappings of class names to class numbers. The table also does not permit multiple entries with either the same class name or class number. If a duplicate entry is inserted into the table, a warning message of the following format is written to the ObjectServer log:

Warning: W-0BJ-103-002: Class name '*ClassName*' and number '*class_number*' must be unique, row discarded

Related concepts

“Functions” on page 165

Service tables

The service table contains information about IBM Tivoli Composite Application Manager for Internet Service Monitoring.

service.status table

The service.status table is used to control the additional features required to support IBM Tivoli Composite Application Manager for Internet Service Monitoring.

The following table describes the columns in the service.status table.

Table 97. Columns in the service.status table

Column name	Data type	Description
Name	varchar(255)	Name of the service.
CurrentState	integer	Indicates the state of the service: 0: Good 1: Bad 2: Marginal 3: Unknown
StateChange	time	Indicates the last time the service state changed.
LastGoodAt	time	Indicates the last time the service was Good (0).
LastBadAt	time	Indicates the last time the service was Bad (1).
LastMarginalAt	time	Indicates the last time the service was Marginal (2).
LastReportAt	time	Time of the last service status report.

System catalog tables

The catalog database contains the system tables that are 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.memstores table

The catalog.memstores table stores information about memstores, including the hard and soft limits of the memstore size, and how many bytes are currently being used.

The following table describes the columns in the catalog.memstores table.

Table 98. Columns in the catalog.memstores table

Column name	Data type	Description
StoreName	varchar(40)	Name of the memstore.
HardLimit	unsigned	Maximum size of the store in memory.
SoftLimit	unsigned	Current maximum size of the store; can be extended to the hard limit.
UsedBytes	unsigned	The amount of memory (in bytes) being used by the memstore.
IsProtected	Boolean	Used internally.

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).

The following table describes the columns in the catalog.databases table.

Table 99. Columns in the catalog.databases table

Column name	Data type	Description
DatabaseName	varchar(40)	Name of the database.
NumTables	unsigned	Number of base tables and views in the database.
IsSystem	Boolean	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.

The following table describes the columns in the catalog.tables table.

Table 100. Columns in the catalog.tables table

Column name	Data type	Description
TableName	varchar(40)	Name of the table.
DatabaseName	varchar(40)	Name of the parent database.

Table 100. Columns in the catalog.tables table (continued)

Column name	Data type	Description
Status	integer	Current status of the table: 0: Valid 1: Invalid 2: Compile failed
NumDependents	unsigned	Number of dependents.
TableID	integer	Table identifier.
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.

The following table describes the columns in the catalog.base_tables table.

Table 101. 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.
UsedBytes	unsigned	Size (in bytes) of the table. Updated on creation, alteration, and at a default interval of 60 seconds.

catalog.views table

The catalog.views table stores information about views. The CreationText column contains the SQL used to create the view.

The following table describes the columns in the catalog.views table.

Table 102. Columns in the catalog.views table

Column name	Data type	Description
ViewName	varchar(40)	Name of the view.
DatabaseName	varchar(40)	Name of the parent database.
CreationText	varchar(16384)	The CREATE VIEW text used to create the view.
StorageKind	integer	Type of storage: 1: Persistent 4: Transient
IsRecovered	Boolean	TRUE if this is a successfully recovered view after restart.
IsDmlEnabled	Boolean	TRUE if all of the table's primary keys are in the view, and therefore DML actions can be performed on the view.
IsAggregate	Boolean	TRUE if this is created from an aggregate SELECT 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.

The following table describes the columns in the catalog.files table.

Table 103. Columns in the catalog.files table

Column name	Data type	Description
FileName	varchar(40)	Name of the ObjectServer file.
FilePath	varchar(1028)	Full path to the file on the file system.
MaximumFiles	unsigned	Maximum number of files.
MaximumSize	unsigned	Maximum file size.
IsEnabled	Boolean	TRUE if information is being logged to this file.
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.

The following table describes the columns in the catalog.restrictions table.

Table 104. 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 CREATE RESTRICTION text used to create the restriction filter.

catalog.columns table

The catalog.columns table stores information about table columns, including their data types.

The following table describes the columns in the catalog.columns table.

Table 105. Columns in the catalog.columns table

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.
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.

The following table describes the columns in the catalog.primitive_signals table.

Table 106. Columns in the catalog.primitive_signals table

Column name	Data type	Description
SignalName	varchar(40)	Name of the signal.
IsSystem	Boolean	TRUE if this is a system signal.

Table 106. Columns in the catalog.primitive_signals table (continued)

Column name	Data type	Description
CommentBlock	varchar(1024)	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.

The following table describes the columns in the catalog.primitive_signal_parameters table.

Table 107. Columns in the catalog.primitive_signal_parameters table

Column name	Data type	Description
ParameterName	varchar(40)	Name of the parameter.
SignalName	varchar(40)	Name of signal with this parameter.
DataType	integer	Parameter data type.
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 the trigger group is enabled.

The following table describes the columns in the catalog.trigger_groups table.

Table 108. 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.

The following table describes the columns in the catalog.triggers table.

Table 109. Columns in the catalog.triggers table

Column name	Data 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.

Table 109. Columns in the catalog.triggers table (continued)

Column name	Data type	Description
IsEnabled	Boolean	TRUE if the trigger is enabled.
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.

The following table describes the columns in the catalog.database_triggers table.

Table 110. Columns in the catalog.database_triggers table

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
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.

The following table describes the columns in the catalog.signal_triggers table.

Table 111. 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.

The following table describes the columns in the catalog.temporal_triggers table.

Table 112. 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.

The following table describes the columns in the catalog.procedures table.

Table 113. Columns in the catalog.procedures table

Column name	Data type	Description
ProcedureName	varchar(40)	Name of the procedure.
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.

The following table describes the columns in the catalog.sql_procedures table.

Table 114. 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.

The following table describes the columns in the catalog.external_procedures table.

Table 115. 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.

The following table describes the columns in the catalog.procedure_parameters table.

Table 116. 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.

The following table describes the columns in the catalog.connections table.

Table 117. Columns in the catalog.connections table

Column name	Data type	Description
ConnectionID	integer	Connection identifier.
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 uses IDUC. Desktops and gateways use IDUC and are real-time connections.
ConnectTime	time	Amount of time the client is connected.

catalog.properties table

The catalog.properties table contains information about ObjectServer properties.

The following table describes the columns in the catalog.properties table.

Table 118. 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 Auto or Store . 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 IBM Software 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.

The following table describes the columns in the catalog.security_permissions table.

Table 119. 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.

Table 119. Columns in the catalog.security_permissions table (continued)

Column name	Data type	Description
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 running SQL commands from client connections.

SQL profile statistics are also logged to the file `$NCHOME/omnibus/log/servername_profiler_report.logn`, at the interval specified in the **ProfileStatsInterval** property or `-profilestatsinterval` command-line option.

The following table describes the columns in the catalog.profiles table.

Table 120. Columns in the catalog.profiles table

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 running time, in seconds, for this client.
MaxSQLTime	real	Longest running time, in seconds, for this client.
PeriodSQLTime	real	Amount of time, in seconds, that the application has spent running SQL since the last profile report.
TotalSQLTime	real	Total time, in seconds, for running all SQL commands for this client.
LastTimingAt	time	Last time an SQL profile was taken for this client.
NumSubmits	integer	Number of submissions for this client. A single submission can contain multiple SQL commands, run with the go command.
TotalParseTime	real	Records the total amount of time spent parsing commands for this client.
TotalResolveTime	real	Records the total amount of time spent resolving commands for this client.
TotalExecTime	real	Records the total amount of time spent running commands for this client.

catalog.indexes table

The catalog.indexes table stores information about indexes, including the column and database table on which the index is based, and the index type.

The following table describes the columns in the catalog.indexes table.

Table 121. Columns in the catalog.indexes table

Column name	Data type	Description
IndexName	varchar(40)	Name of the index.
DatabaseName	varchar(40)	Database name of the indexed column.
TableName	varchar(40)	Table name of the indexed column.
ColumnName	varchar(40)	Name of the indexed column.
IndexKind	integer	Type of index: 1: Hash 2: Tree
IsValid	Boolean	TRUE if the index is valid. FALSE if the index is not valid; for example, due to a memory allocation failure. To make an index valid again, you can rebuild the index by restarting the ObjectServer.

Statistics tables

Statistics tables contain timing information.

The catalog.profiles table contains timing information for the running 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 running SQL commands from client connections. SQL profiling is enabled by using the **Profile** property or -profile command-line option.

SQL profile statistics are also logged to the file \$NCHOME/omnibus/log/servername_profiler_report.logn, at the interval specified in the **ProfileStatsInterval** property or -profilestatsinterval command-line option.

The following table describes the columns in the catalog.profiles table.

Table 122. Columns in the catalog.profiles table

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.

Table 122. Columns in the catalog.profiles table (continued)

Column name	Data type	Description
ProfiledFrom	time	Time at which profiling began.
LastSQLTime	real	Duration, in seconds, of the last SQL command.
MinSQLTime	real	Shortest running time, in seconds, for this client.
MaxSQLTime	real	Longest running time, in seconds, for this client.
PeriodSQLTime	real	Amount of time, in seconds, that the application has spent running SQL since the last profile report.
TotalsQLTime	real	Total time, in seconds, for running all SQL commands for this client.
LastTimingAt	time	Last time an SQL profile was taken for this client.
NumSubmits	integer	Number of submissions for this client. A single submission can contain multiple SQL commands, run with the go command.
TotalParseTime	real	Records the total amount of time spent parsing commands for this client.
TotalResolveTime	real	Records the total amount of time spent resolving commands for this client.
TotalExecTime	real	Records the total amount of time spent running commands for this client.

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.

The following table describes the columns in the master.stats table.

Table 123. Columns in the master.stats table

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 alerts.status table.
JournalCount	integer	The current number of entries in the alerts.journal table.
DetailCount	integer	The current number of entries in the alerts.details table.
StatusInserts	integer	The total number of inserts into the alerts.status table.
StatusNewInserts	integer	The number of new inserts into the alerts.status table.
StatusDedups	integer	The number of reinserts into the alerts.status table.
JournalInserts	integer	The number of inserts into the alerts.journal table.

Table 123. Columns in the master.stats table (continued)

Column name	Data type	Description
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 \$NCHOME/omnibus/log/servername_trigger_stats.logn.

The following table describes the columns in the catalog.trigger_stats table.

Table 124. Columns in the catalog.trigger_stats table

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 run.
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.

catalog.channel_stats table

The catalog.channel_stats table holds an entry for each channel currently known within the ObjectServer. Each entry within the table presents details about the channel in two parts: statistics for the current statistical collection period, and the statistical total since the ObjectServer was started.

The following table describes the columns in the catalog.channel_stats table.

Table 125. Columns in the catalog.channel_stats table

Column name	Data type	Description
ChannelName	varchar(64)	The name of the channel.

Table 125. Columns in the catalog.channel_stats table (continued)

Column name	Data type	Description
LastTimingAt	time	The time at which the last statistics update was made; stored as a UTC.
PeriodNumMsgs	int	The number of times this channel has been used in the last profile window.
PeriodTime	real	The time taken to send messages to all clients in the last profile period window.
PeriodMaxTime	real	The maximum time taken to send messages to all clients in the last profile period window.
PeriodClientNum	int	The number of clients that were sent messages in the last invocation.
PeriodMaxClientNum	int	The maximum number of clients sent messages in the last profile period window.
NumMsg	int	The number of times this channel has been used since the server was started.
TotalTime	real	The total time taken to send messages to all clients since the server was started.
MaxTime	real	The maximum time taken to send messages to all clients since the server was started.
MaxClientNum	int	The maximum number of clients sent messages since the server was started.
TotalClientNum	int	The total number of clients that were sent messages since the server was started.

Client tool support tables

The client tool support tables are used by the desktop GUIs to display alert information.

alerts.resolutions table

The alerts.resolutions table is used to maintain the **Resolutions** option in the event list.

The following table describes the columns in the alerts.resolutions table.

Table 126. Columns in the alerts.resolutions table

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.
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.

The following table describes the columns in the alerts.conversions table.

Table 127. 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.

The following table describes the columns in the alerts.col_visuals table.

Table 128. Columns in the alerts.col_visuals table

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.
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.

The following table describes the columns in the alerts.colors table.

Table 129. Columns in the alerts.colors table

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.
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.

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.

The following table describes the columns in the tools.actions table.

Table 130. Columns in the tools.actions table

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.

Table 130. Columns in the tools.actions table (continued)

Column name	Data type	Description
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.
ExternalEffect1	varchar(255)	The first line of the external procedure.
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 operating system that 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 window to be opened when the tool runs.

tools.action_access table

The tools.action_access table is used to control access to desktop tools.

The following table describes the columns in the tools.action_access table.

Table 131. 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.menus table

The tools.menus table is used to control desktop tool menus.

The following table describes the columns in the tools.menus table.

Table 132. Columns in the tools.menus table

Column name	Data type	Description
MenuID	integer	Primary key field for the menus 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.

tools.menu_items table

The tools.menu_items table is used to control desktop tool menu items.

The following table describes the columns in the tools.menu_items table.

Table 133. Columns in the tools.menu_items table

Column name	Data type	Description
KeyField	varchar(32)	A key field for this menu item. Created from the menu_id (in the tools.menus table) and the menu_item_id.
MenuID	integer	The unique menu identifier taken from the tools.menus 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.
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 the InvokeType column.
Position	integer	Indicates the position (order) of this item on the menu.
Accelerator	varchar(32)	Indicates the keyboard shortcut of this menu item.

tools.prompt_defs table

The tools.prompt_defs table is used to store all prompt definitions.

The following table describes the columns in the tools.prompt_defs table.

Table 134. 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.

Table 134. Columns in the tools.prompt_defs table (continued)

Column name	Data type	Description
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.

The following table describes the columns in the tools.menu_defs table.

Table 135. Columns in the tools.menu_defs table

Column name	Data type	Description
Name	varchar(64)	The name of the menu. This must match the name of the type 7 prompt definition.
DatabaseName	varchar(64)	The database used to build the menu items.
TableName	varchar(64)	The table used to build the menu items.
ShowField	varchar(64)	The field in the table to show as the menu pull-down list.
AssignField	varchar(64)	The actual field used to enter a value into the prompt.
OrderByField	varchar(64)	The field used to order the menu.
WhereClause	varchar(255)	The filter (condition) to show a subset of menu items.
Direction	integer	The order of the menu items: 0: Ascending 1: Descending

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.

master.national table

The master.national table identifies the master ObjectServer and the dual-write mode in a desktop ObjectServer architecture.

The following table describes the columns in the master.national table.

Table 136. Columns in the master.national table

Column name	Data type	Description
KeyField	incr	Primary key column for table.
MasterServer	varchar(29)	Name of the master ObjectServer in a desktop ObjectServer architecture.

Table 136. Columns in the master.national table (continued)

Column name	Data type	Description
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: enabled 0: disabled

master.servergroups table

The master.servergroups table is used to load-balance desktop connections.

The following table describes the columns in the master.servergroups table.

Table 137. Columns in the master.servergroups table

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.
Weight	integer	The priority for each desktop ObjectServer. Higher values attract proportionally more connections. For example, an ObjectServer with a Weight of 2 attracts twice the number of connections as one with a Weight of 1. Load balanced connections are not redirected to ObjectServer with a Weight of 0.

Security tables for backward compatibility

In Netcool/OMNIBus V3.6, 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 provided the user restriction information. These tables are only required for compatibility with previous versions of the desktop.

IDUC tables

The iduc_system database stores all the IDUC application support tables for accelerated event notification, sending informational messages, and IDUC update times.

iduc_system.channel table

The iduc_system.channel table defines the set of known IDUC channels within the ObjectServer.

The following table describes the columns in the iduc_system.channel table.

Table 138. Columns in the iduc_system.channel table

Column name	Data type	Description
Name	varchar(64)	The channel name.
ChannelID	int	A key that is added for a more efficient reference to details of the channel stored in the associated tables.
Description	varchar(2048)	The channel description.

iduc_system.channel_interest table

The iduc_system.channel_interest table stores the channel interest entries for a given channel. There can be multiple interest entries per channel.

The following table describes the columns in the iduc_system.channel_interest table.

Table 139. Columns in the iduc_system.channel_interest table

Column name	Data type	Description
Element Name	varchar(64)	The name of the channel recipients. This is a user or group name.
IsGroup	int	An indication of whether the recipients are a group of users.
Hostname	varchar(255)	The host name of the IDUC client.
AppName	varchar(255)	The application name of the IDUC client.
AppDescription	varchar(255)	The application description of the IDUC client.
ChannelID	int	The channel ID.

iduc_system.channel_summary table

The iduc_system.channel_summary table is used only for an Event Fast Track (or accelerated event) IDUC client command. Rows from any table in the ObjectServer can be forwarded as accelerated events. This table enables a channel to be associated with multiple tables from which events can be accelerated.

The following table describes the columns in the iduc_system.channel_summary table.

Table 140. Columns in the iduc_system.channel_summary table

Column name	Data type	Description
DatabaseName	varchar(64)	The database to which the table belongs.
TableName	varchar(64)	The name of the table.
SummaryID	int	An integer key that is added for a more efficient reference to the summary columns table.
ChannelID	int	The channel ID.

iduc_system.channel_summary_cols table

The iduc_system.channel_summary_cols table stores details on the exact columns that make up the actual summary for a given table.

The following table describes the columns in the iduc_system.channel_summary_cols table.

Table 141. Columns in the iduc_system.channel_summary_cols table

Column name	Data type	Description
ColumnName	varchar(64)	The name of a column that is part of a given summary definition.
Position	int	The position of the column in the summary order.
SummaryID	int	The summary ID.

iduc_system.iduc_stats table

The iduc_system.iduc_stats table stores details about the last time at which IDUC changes were passed to an IDUC client.

The following table describes the columns in the iduc_system.iduc_stats table.

Table 142. Columns in the iduc_system.iduc_stats table

Column name	Data type	Description
ServerName	varchar(40)	The name of the ObjectServer to which the IDUC client connects. Primary key.
AppName	varchar(40)	The application name of the IDUC client.
AppDesc	varchar(128)	The application description of the IDUC client. Primary key.
ConnectionId	int	Uniquely identifies the connection. Primary key.
LastIducTime	UTC	The time at which IDUC changes were last passed to the IDUC client. This column is updated whenever the gateway fetches data from the ObjectServer.

Service-affected events tables

The service-affected events tables provide support for service-affected events that are generated in Network Manager IP Edition. A service-affected event is an alert that warns operators that a critical customer service has been affected by one or more network events.

precision.service_affecting_event table

The precision.service_affecting_event table stores the identifier of a service-affected event that is generated in Network Manager IP Edition.

The following table describes the column in the precision.service_affecting_event table.

Table 143. Column in the precision.service_affecting_event table

Column name	Data type	Description
ServiceEntityId	integer	Primary key column for table. The ID of a service-affected event.

precision.service_details table

The precision.service_details table stores the details of a service-affected event that is generated in Network Manager IP Edition.

The following table describes the columns in the precision.service_details table.

Table 144. Columns in the precision.service_details table

Column name	Data type	Description
ServiceEntityId	integer	Primary key column for table. The ID of a service-affected event.
Type	varchar(255)	The type of service.
Name	varchar(255)	The service name.
Customer	varchar(255)	The associated customer.

precision.entity_service table

The precision.entity_service table maps the identifiers of service-affected events to the numerical IDs that uniquely identify the Network Manager IP Edition topology entities with which the events are associated.

The following table describes the columns in the precision.entity_service table.

Table 145. Columns in the precision.entity_service table

Column name	Data type	Description
NmosEntityId	integer	Primary key column for table. A numerical ID that identifies the Network Manager IP Edition topology entity with which a service-affected event is associated.
ServiceEntityId	integer	Primary key column for table. The ID of a service-affected event.

Appendix B. SQL commands, variable expressions, and helper buttons in tools, automations, and transient event lists

You can use a number of SQL commands, variable expressions, and helper buttons 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.

The following table lists the SQL commands, variable expressions, and helper buttons.

Table 146. SQL commands, variable expressions, and helper buttons in tools, triggers, procedures, and the transient event list






Command/ variable expression	Button	Usage
select_command insert_command update_command delete_command use_command service_command		Click this button to select an SQL command from the pop-up menu. Based on the command that you select, complete the resulting window as follows: <ul style="list-style-type: none"> • Select: Select the database and table on which to run the SELECT command. Then, choose the table columns to select. • Insert: Select the database and table on which to run the INSERT command. Then, select the table columns in which to insert values. For each selected column, enter the value to insert. For insert statements, you must include the primary key. Primary keys are indicated with an asterisk (*). • Update: Select the database and table on which to run the command. Then, select the table columns to update. For each selected column, enter the new value. For update statements, you must exclude the primary key. Primary keys are indicated with an asterisk (*). Note: For inserts and updates to the alerts.status table, any existing conversions appear in the drop-down lists. • Delete: Select the table to delete. • Use: Select the database to use. • Service: Select a service name and a value. Values can be Good, Marginal, or Bad.
column_name @column_name		Click this button to select a table column name to add to the command. The column name is substituted for the corresponding event list row value when the tool runs. 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: RemoteNodeAlias = '@LocalNodeAlias'
conversion_name		Click this button to select from a list of available conversions.
N/A		Click this button to bring up a list of keywords that complete the entered SQL.
N/A		Click this button to check the validity of the entered SQL syntax.

Table 146. SQL commands, variable expressions, and helper buttons in tools, triggers, procedures, and the transient event list (continued)



Command/ variable expression	Button	Usage
%internal_value		<p>Click this button to select from a list of internal values known to the current instance of the event list. For example, to run the transient event list and specify the ObjectServer to connect to using the -server command-line option, specify: -server "%server"</p> <p>The following internal values are available for tools and as a parameter to the transient event list:</p> <ul style="list-style-type: none"> %display: The current display running the application (UNIX only). %password: The password of the user running the application. %encrypted_password: The encrypted password of the user running the application (UNIX only). In FIPS 140-2 mode, the password is passed as plain text when used in tools, but is hidden when specified as a parameter from the command line. %server: The name of the ObjectServer to which the tool is currently connected. %desktopserver: The name of the desktop ObjectServer to which the tool is currently connected. %uid: The ObjectServer user identifier of the user running the application. %username: The ObjectServer user name of the user running the application. <p>The following internal value is available for procedures and triggers:</p> <ul style="list-style-type: none"> %user: Used to specify user variables and access information about connected users. <p>The following internal value is available for triggers only:</p> <ul style="list-style-type: none"> %trigger: Used to specify trigger variables and access information about the current and previous executions of triggers. <p>The following internal value is additionally available for signal triggers only:</p> <ul style="list-style-type: none"> %signal: Used to specify signal variables and access information about signals raised.
\$prompt. prompt_name		<p>Click this button to select the name of the prompt to use when querying the user. For example, to run the transient event list and prompt the user to enter their password using the Password prompt, specify: -password \$prompt.Password</p> <p>You can use prompts in tools and as a parameter to the transient event list.</p>
\$selected_rows. column_name	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>

Table 146. SQL commands, variable expressions, and helper buttons in tools, triggers, procedures, and the transient event list (continued)

Command/ variable expression	Button	Usage
<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 by using the <code>-elf</code> command-line option, such as:</p> <pre>-elf "\$(NCHOME)/omnibus/ini/tool.elf</pre> <p>To run the tool on Windows, enclose the environment variable, such as <code>\$(NCHOME)</code>, in double quotation marks. If there is a space in the path name, it will not be interpreted correctly.</p>

Tip:

- When typing SQL commands within the Tivoli Netcool/OMNIBus SQL editor panels, you can type one or more characters and then press Ctrl+F1 to obtain a dialog box with a list of keywords that might match your entry. Select the required keyword and click **OK** to complete your entry. If only one keyword matches your typed characters, the keyword is automatically completed for you. If you press Ctrl+F1 after typing a database-related keyword, the dialog box provides a list of possible ObjectServer databases from which you can select. If you press Ctrl+F1 after typing a database name followed by a dot (for example: `alerts.`), you can press Ctrl+F1 again to view and select from a list of tables in the database.
- You can click the **To Clipboard** button to copy the command in a text format to the clipboard.

Related tasks

- “Creating and editing tools” on page 91
- “Creating and editing database triggers” on page 99
- “Creating and editing signal triggers” on page 101
- “Creating and editing temporal triggers” on page 103
- “Creating and editing SQL procedures” on page 107

Related reference

- “Implicit user variables in procedures and triggers” on page 202
- “Using trigger variables in trigger conditions and actions” on page 215
- “System signals and their attributes” on page 217

Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

Trademarks

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

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



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

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

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

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

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

Index

A

- accelerated event notification
 - configuring 239
 - configuring alerts.status 241
 - configuring channels 241
 - configuring gateway 240
 - configuring probe rules 239
 - configuring triggers 247
 - disconnecting clients 245
 - sending messages to channel recipients 245
 - shutting down clients 246
- accessibility x
- ADD COLUMN 148
- adding
 - columns 124, 148
 - separators to menus 87
 - submenus to menus 87
 - table columns 124, 148
 - tools to menus 87
- aggregate SELECT 177
- alert processing 1
- alerts
 - sending to ObjectServer 26
- alerts tables 321
- alerts.application_types table 336
- alerts.col_visuals table 352
- alerts.colors table 352
- alerts.conversions table 352
- alerts.details table 334
- alerts.iduc_messages table 335
- alerts.journal table 335
- alerts.resolutions table 351
- alerts.status table 321
- ALTER COLUMN 150
- ALTER FILE 158
- ALTER GROUP 186
- ALTER ROLE 188
- ALTER SYSTEM 181
- ALTER TABLE 147
- ALTER TRIGGER 229
- ALTER TRIGGER GROUP 206
- ALTER USER 185
- altering
 - columns 150
 - files 158
 - groups 186
 - roles 188
 - table columns 150
 - tables 147
 - triggers 229
 - users 185
- arch
 - operating system directory x
- assigning permissions
 - roles 188
- assigning roles
 - groups 191
- audience vii
- authorization
 - description 71

- automations
 - configuring 97
 - service-affected events 235
 - standard 230

B

- best practices
 - triggers 317
- binary comparison operators 161

C

- CASE WHEN 200
- catalog.base_tables table 339
- catalog.channel_stats table 350
- catalog.columns table 341
- catalog.connections table 346
- catalog.database_triggers table 343
- catalog.databases table 338
- catalog.external_procedures table 345
- catalog.files table 340
- catalog.indexes table 348
- catalog.memstores table 338
- catalog.primitive_signal_parameters table 342
- catalog.primitive_signals table 341
- catalog.procedure_parameters table 345
- catalog.procedures table 344
- catalog.profiles table 348
- catalog.properties table 346
- catalog.restrictions table 341
- catalog.security_permissions table 346
- catalog.signal_triggers table 344
- catalog.sql_procedures table 344
- catalog.tables table 338
- catalog.temporal_triggers table 344
- catalog.trigger_groups table 342
- catalog.trigger_stats table 350
- catalog.triggers table 342
- catalog.views table 340
- channels
 - configuring 131
 - copying 244
 - creating 241
 - deleting 245
 - editing 241
 - pasting 244
- CHECK STATEMENT 183
- checkpoint files 23
- checkpointing
 - checkpoint files 23
 - nco_check_store 24
 - ObjectServer 23
- classes
 - creating 121
 - deleting 121
 - editing 121
- client tool support tables 351
- column visuals
 - creating 120
 - deleting 121
 - editing 120
- columns
 - adding 124, 148
 - altering 150
 - data types 145
 - deleting 126, 148
 - dropping 148
 - editing 124, 150
 - optional properties 146
- command-line options
 - ObjectServer 3
- components
 - process control 250
- conditions 172
- configuring
 - accelerated event notification 239, 240, 241, 247
 - channels 131
 - databases 122
 - files 128
 - groups 76
 - procedures 107
 - prompts 93
 - properties 127
 - restriction filters 84
 - roles 71
 - signals 115
 - tools 90
 - triggers 97
 - users 79
- connecting
 - ObjectServer 62
 - process agent 64, 281
- conventions, typeface x
- conversions
 - creating 118
 - deleting 118
 - editing 118
- copying
 - channels 244
 - processes 293
 - services 293
- CREATE DATABASE 142
- CREATE FILE 156
- CREATE GROUP 186
- CREATE INDEX 152
- CREATE PROCEDURE 196, 203
- CREATE RESTRICTION FILTER 155
- CREATE ROLE 187
- CREATE SIGNAL 213
- CREATE TABLE 144
- CREATE TRIGGER 208, 210, 212
- CREATE TRIGGER GROUP 206
- CREATE USER 184
- CREATE VIEW 154
- creating
 - channels 241
 - classes 121

- creating (*continued*)
 - column visuals 120
 - conversions 118
 - database triggers 99, 208
 - databases 122, 142
 - external procedures 110, 203
 - files 156
 - groups 77, 186
 - indexes 152
 - ObjectServer files 128
 - processes 288
 - prompts 94
 - restriction filters 84, 155
 - roles 74, 187
 - services 285
 - severity colors 119
 - signal triggers 101, 212
 - SQL procedures 107, 196
 - tables 123, 144
 - temporal triggers 103, 210
 - tools 91
 - trigger groups 98, 206
 - user-defined signals 115, 213
 - users 80, 184
 - views 154
- customizing
 - menus 86

D

- data types
 - columns 145
- database triggers
 - creating 99, 208
 - editing 99
- databases
 - configuring 122
 - creating 122, 142
 - deleting 126, 143
 - dropping 143
 - system-initialized 143
- defining
 - routing hosts 270
 - secure hosts 269
- DELETE 175
- deleting
 - channels 245
 - classes 121
 - column visuals 121
 - columns 126, 148
 - conversions 118
 - databases 126, 143
 - groups 79, 187
 - ObjectServer files 130
 - procedures 115
 - processes 290
 - prompts 97
 - restriction filters 86
 - roles 76, 194
 - separators from menu 90
 - services 286
 - submenus from menu 90
 - table columns 126, 148
 - tables 126
 - tools 93
 - tools from menu 90
 - trigger groups 107, 207

- deleting (*continued*)
 - triggers 107, 230
 - user-defined signals 117
 - users 83
- DESCRIBE 180
- desktop tables 356
- desktop tools tables 353
- displaying
 - firewall bridge server property value 56
- DROP COLUMN 148
- DROP DATABASE 143
- DROP FILE 158
- DROP GROUP 187
- DROP INDEX 153
- DROP PROCEDURE 205
- DROP RESTRICTION FILTER 156
- DROP ROLE 194
- DROP SIGNAL 214
- DROP TABLE 151
- DROP TRIGGER 230
- DROP TRIGGER GROUP 207
- DROP USER 186
- DROP VIEW 155
- dropping
 - columns 148
 - databases 143
 - files 158
 - indexes 153
 - procedures 205
 - restriction filters 156
 - table columns 148
 - tables 151
 - user-defined signals 214
 - users 186
 - views 155

E

- editing
 - channels 241
 - classes 121
 - column visuals 120
 - columns 124, 150
 - conversions 118
 - database triggers 99
 - external procedures 110
 - groups 77
 - menus 88, 89
 - ObjectServer files 128
 - processes 288
 - prompts 94
 - restriction filters 84
 - roles 74
 - services 285
 - severity colors 119
 - signal triggers 101
 - SQL procedures 107
 - table columns 124, 150
 - tables 147
 - temporal triggers 103
 - tools 91
 - trigger groups 98, 206
 - user-defined signals 115
 - users 80
- education
 - see Tivoli technical training x

- encrypting
 - ObjectServer passwords 17
 - passwords 141
- environment variables, notation x
- events
 - sending to ObjectServer 26
- examples
 - nco_postmsg 32
- EXECUTE PROCEDURE 205
- exiting
 - Netcool/OMNIBus Administrator 71
 - SQL interactive interface 141
- expressions 171
- external actions
 - running 294
- external editors for procedures 113
- external editors for triggers 105
- external procedures
 - creating 110, 203
 - editing 110

F

- files
 - altering 158
 - configuring 128
 - creating 156
 - dropping 158
- firewall bridge
 - standard setup 44
- firewall bridge failover configuration
 - failover 47
- firewall bridge server
 - command-line options 50
 - nco_bridgeserv 49
 - overview 43
 - properties 50
 - starting 49
 - starting manually 49
 - starting using process control 49
 - starting using services 49
- firewall bridge server data flows
 - listing 56
- firewall bridge server properties
 - listing 55
- firewall bridge server property value
 - displaying 56
- firewall bridge server shutdown
 - shutdown 57
- FOR 201
- FOR EACH ROW 200
- functions 165

G

- GET CONFIG; 55
- GET PROP; 56
- GRANT 188
- GRANT ROLE 191
- Granularity property 20
- group by SELECT 178
- groups
 - altering 186
 - assigning roles 191
 - configuring 76
 - creating 77, 186

groups (*continued*)
 default 76
 deleting 79, 187
 editing 77

H

helper buttons 361

I

IDUC 20
 specifying port 20
 update interval 20
IDUC EVTFT 228
IDUC FLUSH 181
IDUC SNDMSG 229
IDUC tables 357
iduc_system.channel table 358
iduc_system.channel_interest table 358
iduc_system.channel_summary table 358
iduc_system.channel_summary_cols
 table 359
iduc_system.iduc_stats table 359
IF THEN ELSE 199
implicit user variables 202
implicit variables 209
index details
 viewing 153
indexes
 creating 152
 dropping 153
indexing guidelines 314
INSERT 173
isql 135, 136
 command-line options 136

K

key performance indicators
 gateways 302
 ObjectServer 298
 probes 301

L

list comparison operators 162
listing
 firewall bridge server data flows 56
 firewall bridge server properties 55
Logging level
 specifying 57
logical operators 163

M

manuals viii
master.class_membership table 336
master.national table 356
master.servergroups table 357
master.stats table 349
math operators 160
memstores
 ObjectServer data storage 22
 table_store 25

menus
 customizing 86
 editing 88, 89
 previewing structure 90
monitoring
 ObjectServer
 connections 130
 ObjectServer connections 130
multicultural support
 Netcool/OMNibus Administrator 59
 ObjectServer 21
multiple firewall bridge
 multiple setup 46

N

naming conventions
 ObjectServer objects 139
nco_aes_crypt 17, 39
nco_bridgeserv 49
nco_check_store 24
nco_config 60
 command-line options 60
 properties 60
nco_g_crypt 17, 39
nco_objserv 1
nco_pa_addentry 277
nco_pa_shutdown 276
nco_pa_start 275
nco_pa_status 273
nco_pa_stop 275
nco_pad 255
 command-line options 256
nco_postmsg 26
 command-line options 29
 examples 32
 properties 29
nco_proxyserv 36
nco_sql 135, 136
 command-line options 136
nco_store_resize 22
 command-line options 25
Netcool/OMNibus Administrator
 aligning columns 68
 command-line options 60
 configuring syntax colors 69
 copying objects 69
 exiting 71
 filtering rows 68
 hiding columns 68
 multicultural support 59
 nco_config 60
 pasting objects 69
 properties 60
 selecting columns 68
 selecting ObjectServer objects 67
 selecting rows to display 68
 setting preferences 68
 sorting results tables 68
 SQL syntax colors 69
 starting 60
 Web browser for online help 70

O

object permissions 188
 inheritance 191
ObjectServer
 alert processing 1
 ALTER SYSTEM 1
 changing properties 127
 checkpointing 23
 command-line options 3
 configuring automations 97
 connecting 62
 data recovery 24
 file creation sequence 128
 Granularity property 20
 IDUC 20
 IDUC update interval 20
 maintaining database table files on
 disk 23
 memstores 22
 monitoring connections 130
 multicultural support 21
 naming conventions for objects 139
 nco_aes_crypt 17
 nco_g_crypt 17
 operator precedence 165
 properties 3
 properties file 1
 reserved words 158
 secure mode 17
 specifying command-line options 1
 specifying properties 1
 SQL 135
 system tables 151
 viewing properties 127
ObjectServer files
 creating 128
 deleting 130
 editing 128
ObjectServer tables
 alerts tables 321
 alerts.application_types 336
 alerts.col_visuals 352
 alerts.colors 352
 alerts.conversions 352
 alerts.details 334
 alerts.iduc_messages 335
 alerts.journal 335
 alerts.resolutions 351
 alerts.status 321
 catalog.base_tables 339
 catalog.channel_stats 350
 catalog.columns 341
 catalog.connections 346
 catalog.database_triggers 343
 catalog.databases 338
 catalog.external_procedures 345
 catalog.files 340
 catalog.indexes 348
 catalog.memstores 338
 catalog.primitive_signal_param. 342
 catalog.primitive_signals 341
 catalog.procedure_parameters 345
 catalog.procedures 344
 catalog.profiles 348
 catalog.properties 346
 catalog.restrictions 341
 catalog.security_permissions 346

- ObjectServer tables (*continued*)
 - catalog.signal_triggers 344
 - catalog.sql_procedures 344
 - catalog.tables 338
 - catalog.temporal_triggers 344
 - catalog.trigger_groups 342
 - catalog.trigger_stats 350
 - catalog.triggers 342
 - catalog.views 340
 - client tool support tables 351
 - desktop tables 356
 - desktop tools tables 353
 - IDUC tables 357
 - iduc_system.channel 358
 - iduc_system.channel_interest 358
 - iduc_system.channel_sum*_cols 359
 - iduc_system.channel_summary 358
 - iduc_system.iduc_stats 359
 - master.class_membership 336
 - master.national 356
 - master.servergroups 357
 - master.stats 349
 - overview 321
 - precision.entity_service 360
 - precision.service_affecting_event 359
 - precision.service_details 360
 - security tables 357
 - service tables 337
 - service-affected events tables 359
 - service.status 337
 - statistics tables 348
 - system catalog tables 338
 - tools.action_access 354
 - tools.actions 353
 - tools.menu_defs 356
 - tools.menu_items 355
 - tools.menus 355
 - tools.prompt_defs 355
 - online publications viii
 - operating system directory
 - arch x
 - operator precedence 165
 - operators 160
 - binary comparison 161
 - list comparison 162
 - logical 163
 - math 160
 - string 160
 - optimization rules
 - AND optimization 311
 - for SQL queries 311
 - OR optimization 311
 - reordering of predicates 311
 - optional properties
 - columns 146
 - ordering publications viii
- P**
- password encryption 17, 39, 141
 - pasting
 - channels 244
 - processes 293
 - services 293
 - performance tuning 297
 - best practices 303
 - alerts.details table 308
 - performance tuning (*continued*)
 - best practices (*continued*)
 - collecting trigger statistics 305
 - event flood 308
 - gateway KPIs 302
 - monitoring agent 309
 - ObjectServer KPIs 298
 - ObjectServer profiling 303
 - performance tracking 311
 - probe configuration files 308
 - probe KPIs 301
 - SQL queries 309
 - statistics triggers 307
 - system architecture 307
 - indexing guidelines 314
 - SQL query examples 315
 - SQL query guidelines 311
 - trigger examples 316
 - permissions
 - description 71
 - ports
 - IDUC 20
 - precision.entity_service table 360
 - precision.service_affecting_event table 359
 - precision.service_details table 360
 - preferences
 - Netcool/OMNIBus Administrator 68
 - probes
 - connecting to proxy server 39
 - procedures
 - configuring 107
 - configuring external editors 113
 - creating 107, 110
 - creating in external editors 114
 - deleting 115
 - dropping 205
 - editing 107, 110
 - editing in external editors 114
 - implicit user variables 202
 - running 205
 - SQL 195
 - process agent
 - connecting 64
 - shutting down 276
 - process agents
 - starting automatically 261, 263
 - starting manually 255
 - stopping 294
 - process control
 - adding processes 277
 - adding services 277
 - command-line options 256
 - components 250
 - configuration file 254
 - configuring server
 - communication 254
 - connecting to process agent 281
 - copying processes 293
 - copying services 293
 - creating network 252
 - creating processes 288
 - creating services 285
 - creating UNIX user groups 253
 - defining dependencies 267
 - defining processes 264
 - defining routing hosts 270
 - process control (*continued*)
 - defining secure hosts 269
 - defining services 267
 - deleting processes 290
 - deleting services 286
 - displaying process status 273
 - displaying processes 284
 - displaying service status 273
 - displaying services 284
 - editing processes 288
 - editing services 285
 - logging level 283
 - nco_pa_addentry 277
 - nco_pa_shutdown 276
 - nco_pa_start 275
 - nco_pa_status 273
 - nco_pa_stop 275
 - nco_pad 255
 - overview 249
 - pasting processes 293
 - pasting services 293
 - process agents 249
 - process control
 - host routing 283
 - running external procedures 295
 - sending signals 292
 - shutting down process agent 276
 - starting 252
 - starting processes 275, 291
 - starting services 275, 287
 - status information for process agent 283
 - stopping process agent 294
 - stopping processes 275, 291
 - stopping services 275, 287
 - updating the configuration file 254
 - utilities 251, 273
 - Windows user accounts 253
 - process dependencies 267
 - processes
 - copying 293
 - creating 288
 - deleting 290
 - editing 288
 - pasting 293
 - sending signals 292
 - starting 291
 - stopping 291
 - prompts
 - configuring 93
 - creating 94
 - deleting 97
 - editing 94
 - properties
 - configuring 127
 - ObjectServer 3
 - proxy server
 - command-line options 36
 - connecting probes 39
 - nco_aes_crypt 39
 - nco_g_crypt 39
 - nco_proxyserv 36
 - overview 35
 - properties 36
 - secure mode 39
 - starting 35
 - starting manually 36

- proxy server (*continued*)
 - starting using process control 35
 - starting using services 36
- publications viii

R

- RAISE SIGNAL 214
- raising
 - user-defined signals 214
- reserved words 158
- restriction filters
 - configuring 84
 - creating 84, 155
 - deleting 86
 - dropping 156
 - editing 84
- REVOKE 192
- REVOKE ROLE 194
- roles
 - altering 188
 - assigning permissions 188
 - configuring 71
 - creating 74, 187
 - default 71
 - deleting 76, 194
 - editing 74
 - revoking from groups 194
 - revoking permissions 192
- routing hosts
 - process control 270
- running
 - external actions 294
 - procedures 205

S

- scalar SELECT 175
- secure hosts
 - process control 269
- secure mode
 - ObjectServer 17
 - proxy server 39
 - SQL interactive interface 141
- security tables 357
- SELECT 175
- SELECT (aggregate) 177
- SELECT (group by) 178
- SELECT (scalar) 175
- sending alerts
 - nco_postmsg 26
- sending events
 - nco_postmsg 26
- sending signals
 - processes 292
- separators
 - adding to menus 87
 - deleting from menu 90
- service tables 337
- service-affected events automations 235
- service-affected events tables 359
- service.status table 337
- services
 - copying 293
 - creating 285
 - deleting 286

- services (*continued*)
 - editing 285
 - pasting 293
 - starting 287
 - stopping 287
- SET 199
- SET DATABASE 182
- SET LOG LEVEL TO 57
- severity colors
 - creating 119
 - editing 119
- SHOW DATAFLOWS 56
- SHOW PROPS; 55
- shutdown
 - firewall bridge server 57
- SHUTDOWN 57
- signal triggers
 - creating 101, 212
 - editing 101
- signals 217
 - configuring 115
- specify
 - logging level 57
- SQL commands 361
- SQL helper buttons 361
- SQL interactive interface
 - command line 135
 - command-line options 136
 - exiting 141
 - GUI mode 131
 - redirecting text files 140
 - running commands 138
 - secure mode 141
 - specifying paths 140
 - starting 136
 - syntax notation 138
- SQL procedures 195
 - body statement 198
- CASE WHEN 200
- components 195
 - creating 107, 196
 - editing 107
- FOR 201
- FOR EACH ROW 200
- IF THEN ELSE 199
- SET 199
- SQL query guidelines 311
 - AND optimization 311
 - optimization rules 311
 - OR optimization 311
 - reordering of predicates 311
- SQL syntax colors
 - configuring 69
- SQL syntax notation 138
- SSL connections 66
 - validating server certificates 66
- starting
 - firewall bridge server 49
 - Netcool/OMNIbus Administrator 60
 - process agents 255, 261, 263
 - processes 291
 - proxy server 35, 36
 - services 287
 - SQL interactive interface 136
- statistics tables 348
- stopping
 - process agent 294

- stopping (*continued*)
 - processes 291
 - services 287
- string operators 160
- submenus
 - adding to menus 87
 - deleting from menu 90
- support information x
- SVC 180
- system catalog tables 338
- system permissions 188
- system signals 217
- system tables 151
- system-initialized databases 143

T

- table columns
 - adding 124, 148
 - altering 150
 - deleting 126, 148
 - dropping 148
 - editing 124, 150
- table_store memstore 25
- tables
 - altering 147
 - creating 123, 144
 - deleting 126
 - deleting rows 175
 - dropping 151
 - editing 147
 - inserting data rows 173
 - retrieving data 175
 - updating columns 174
- temporal triggers
 - creating 103, 210
 - editing 103
- Tivoli software information center viii
- Tivoli technical training x
- tools
 - adding to menus 87
 - configuring 90
 - creating 91
 - deleting 93
 - deleting from menu 90
 - editing 91
- tools.action_access table 354
- tools.actions table 353
- tools.menu_defs table 356
- tools.menu_items table 355
- tools.menus table 355
- tools.prompt_defs table 355
- training, Tivoli technical x
- trigger groups
 - creating 98, 206
 - deleting 107, 207
 - editing 98, 206
- triggers
 - accelerated event notification 228
 - altering 229
 - best practices 317
 - configuring 97
 - configuring external editors 105
 - creating 99, 101, 103
 - creating in external editors 106
 - deleting 107, 230
 - editing 99, 101, 103

- triggers (*continued*)
 - editing in external editors 106
 - implicit user variables 202
 - implicit variables 209
 - running commands 215
 - variables 215
- typeface conventions x

U

- UPDATE 174
- USE DATABASE 182
- user-defined signals
 - creating 115, 213
 - deleting 117
 - dropping 214
 - editing 115
 - raising 214
- users
 - altering 185
 - configuring 79
 - creating 80, 184
 - default 79
 - deleting 83
 - dropping 186
 - editing 80
 - viewing connections 83

V

- variables 202, 209
- variables, notation for x
- viewing
 - index details 153
- views
 - creating 154
 - description 153
 - dropping 155

W

- Web browser
 - Netcool/OMNIBus Administrator 70
- WRITE INTO 179



Printed in the Republic of Ireland

SC14-7605-00

