Netcool®/OMNIbus™ Version 7

Virtual Workshop For ObjectServer Administration

Version 2.0

01 November 2004

Micromuse Ltd
Disraeli House
90 Putney Bridge Road
London SW18 1DA
Tel: +44 (0)20 8875 9500

Document Control Page

This document has been prepared by the following Micromuse personnel:

Authors: Don Wildman

Date: 01 November 2004

Document Revision: 2.0

Status: Draft

Revision History:

Date:	Authors:	Revision:	**Comments:**
31 October 2003	DW	0.1	Initial outline
28 November 2003	DW	0.2	Detail added
19 December 2003	DW	0.3	Revised labs and gateway information added
15 January 2004	DW	0.4	Additional detail on authentication and desktop enhancements
2 February 2004	DW	0.5	Add licensing details for beta training
23 February 2004	DW	0.6	Review following beta training
6 April 2004	DW	0.7	Further feedback from beta program
27 April 2004	DW	0.8	Additional feedback from User Group
12 May 2004	DW	1.0	Issue
19 October 2004	DW	1.1	Draft from v7 with initial updates for v7.0.1
29 October 2004	DW	1.2	Additional review comments added
1 November 2004	DW	2.0	Issued with GA of 7.0.1

Review History:

Date:	**Reviewed by:**
December 2003	Harry Manley Netcool/OMNIbus Product Champions Group
January 2004	Garry Lewis Gerry Van De Zanden Yves de Cloedt Ralph Riedel Tim Greenwood
March 2004	Beta sites: Windward Synergon
October 2004	Britta Binning – Omnibus development Stephen Cook – Omnibus development Heidi Modica - Support

Micromuse Inc. disclaimer of warranty and statement of limited liability

Micromuse Inc. does not warrant that the functions contained in the software will meet your requirements, or that the operation of the software will be uninterrupted or error-free. Any liability of Micromuse Inc. with respect to the software or the performance thereof under any warranty, negligence, or strict liability will be limited exclusively to product replacement or to a refund of the license fee. Micromuse Inc. shall not be liable for any indirect, consequential or incidental damages arising out of the use or the ability to use this product.
Micromuse Inc. specifically disclaims any express or implied warranty of fitness for high-risk activities. Micromuse Inc. wishes to make clear that its products are not certified for fault tolerance, and are not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring failsafe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapon systems ("High Risk Activities") in which the failure of products could lead directly to death, personal injury, or severe physical or environmental damage.
Micromuse Inc. makes no express or implied warranty with respect to the Program or goods or services to be supplied by Micromuse Inc., including without limitation any implied warranty of merchantability or fitness for a particular purpose. Micromuse Inc. does not warrant that the program will be error-free, or that any defects that may exist in the program will be corrected. Customer acknowledges that Micromuse Inc. has made no representations regarding warranty or performance or capability.

TABLE OF CONTENTS

1 Introduction

1.1 Overview

The purpose of this workshop is to provide a practical introduction to the new and modified functionality introduced by Netcool/OMNIbus v7 for engineers and administrators who have current practical knowledge of Netcool/OMNIbus 3.x systems. In particular it assumes good understanding of the ObjectServer structure of SQL, automations, tools and security as administered both via the command line, and interactively via nco_config.

This version of the document contains the original version of the v7 workshop extended to include the modifications introduced in version 7.0.1. The primary purpose of v7.0.1 is to implement version 12.5.1 of the Sybase OpenClient/OpenServer middleware and a series of minor fixes and enhancements identified during the initial rollout of v7. Existing v7 users will be familiar with the bulk of this document which includes change bars to identify additions and changes from version 1.0.

Netcool/OMNIbus v7 provides a new implementation of the in-memory database server. The ObjectServer supports persistence of data using disk-based checkpoints and logs. The v3.x concept of Logical Storage is not available in a v7 ObjectServer.

The v7 ObjectServer provides a richer implementation of a Structured Query Language (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, including tables, views, and restriction filters
- 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 ObjectServer objects

The v7 ObjectServer provides procedural language commands that give you programming constructs for defining actions that will take place when specified events occur and conditions that you define are met. Procedures are used within triggers to form automations replacing the trigger/action pairs of the V3.x ObjectServer.

1.2 Document Structure

The document is necessarily long to include a description of the new functionality of V7. However, as this functionality to an extent provides improved methods of managing familiar v3.x functions, the number of lab exercises is relatively small.

The document is designed to give a practical introduction to the key new and enhanced features of Netcool/OMNIbus v7. It does not provide full reference information for the ObjectServer. This may be found in the Installation and Administration documentation.

It may be helpful to read the document and then return to the lab exercises. Section 1.3 contains an index to help in location the Lab components.

The command line examples in this workshop are worked for UNIX systems. A few Windows commands are also highlighted. The new administration client is platform independent providing similar functionality on both UNIX and Windows platforms.

1.3 Index of Lab Exercises

2 Installation and Migration

2.1 Licensing

Like its predecessor (V3.6) Netcool/OMNIbus v7 requires FLEX licensing. Users migrating from earlier versions that have yet to migrate to the FLEX licensing system should refer to the appropriate installation guide for details.
Netcool/Omnibus V3.6 licenses are valid for the corresponding V7 components. Additional license codes will be required for the new components. The configuration components require the new version of the licensing server due for general release with v7. Other licenses are valid on both the current and new license servers.
Any additional licenses required should be requested from the Key issuance group via the Licensing form on the support web site.

2.2 Installation

The Netcool/OMNIbus 7 UNIX installation is similar to that of Netcool/OMNIbus 3.6 except for the creation of the ObjectServer. A new utility $OMNIHOME/bin/nco_dbinit provides this functionality. This utility makes use of default SQL initialization files from the $OMNIHOME.etc directory.

- application.sql – comprises the definitions of the alerts and custom databases as well as master table views included supporting v3.x backward compatibility. Only the alerts and custom databases should be considered for modification for local requirement. Note that the v3 compatible table views are used by the system when connecting to v3 components. Administrators of v7 should work with the underlying tables, not the v3 style views.
- automations.sql – provides the default automations and tools for the v7 system. This table may be modified for local requirements, but new users will find it simpler to install the defaults, and make modifications via the nco_config administration client.
- security.sql – provides the security structure for the v7 ObjectServer.
- system.sql – provides the new system table structure.

Note that these files are used only by nco_dbinit when first creating the ObjectServer database. Once an ObjectServer is created then database and table modifications are made only through the Administrator client or by the command line as described in the later sections of this document. Online changes to the database structure are not written out in text format. The structure of a modified ObjectServer can be exported by use of the nco_confpack utility first released with OMNIbus 3.6.

The security and system SQL files should not be modified except by the most experienced users.

The default structure provided emulates the v3.x security structure for backward compatibility. Local tuning is best carried out via the administration GUI once the database is created.

Additional tables desktop.sql and desktopserver.sql provide further configuration required to configure a desktopobjectserver when using the –desktopserver command option of nco_dbinit.

The v7 ObjectServer properties differ in some cases from v3.x. Also, some v3.x properties are not required by v7. For example, control of the creation of Connection Watch messages is now managed within the automations system so the MonitorConections property is obsolete.

Most ObjectServer properties can be set firstly in the properties file, and subsequently modified in real time. An important exception to this is the MEMSTORE property that sets the default soft and hard limits for memory usage within the ObjectServer. These values are defined in the system.sql file as

```
create memstore table_store persistent hard limit 500M soft limit 450M;
```

In the initial v7 version these values were both set to 100M. This was found to be too low for some installations with high event volumes. Users of v7 should consider increasing the old default limits depending on their needs. The memory footprint of the ObjectServer will give an approximate guide to the appropriate choice.

A warning alert is generated when the ObjectServer reaches its soft limit. At this point no further writes are possible to the ObjectServer. The soft limit may be modified by use of the alter system command as a temporary respite.

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

An ObjectServer will shutdown if its size hits the hard limit. In current versions of the ObjectServer modification to the hard limit requires creation of a new database with a modified system.sql file, and migration of the data files using nco_confpack or gateway transfer.

Calculation of memstore size should be based on (average event size + average journal size per event + average detail size per event) * maximum number of events held in the ObjectServer. An allowance for growth should also be incorporated.

2.3 Upgrade and Migration

Netcool/OMNIbus 7 and 7.0.1 will install as:

- An upgrade to a v3.x installation with assisted database migration
- A full install

V7.0.1 will also install as an upgrade to a v7 installation. No database migration is required in this case.

A new separate probe distribution was provided with OMNIbus v7. The same version of the probe installation may be used for both v7 and v7.0.1 installations. Probes that have been made available since the preparation of the probe distribution may be obtained from the Micromuse Support download page.

The Netcool/OMNIbus 7.0.1 installation upgrade option applied to v3.x copies key configuration files from the 3.5 or 3.6 installations.

The v3 ObjectServer is not fully migrated automatically. A script is provided that will analyze .dat files from a 3.x ObjectServer to produce data and .sql files in the V7 format. A report is produced to a log file highlighting any issues with the conversion that may require manual intervention. The script is executed against available ObjectServers during the upgrade and may also be run stand-alone to migrate ObjectServers from other systems. The resulting files are then manually input to the utility nco_dbinit to create the new ObjectServer(s). An outline of the upgrade process is included in Appendix A.

Lab: 1) Installation

Obtain the Netcool/OMNIbus 7.0.1 Software for UNIX from the Micromuse Support Site or appropriate ftp site. You will need both the core Netcool/OMNIbus and new separate Probe distribution. The same probe distribution is used for both v7 and v7.0.1 installs.

The full OMNIbus distribution includes a copy of the new license server. If you download an operating system specific copy then you will also need to acquire a copy of the new version of common licensing.

Unset OMNIHOME (if already set) and install by running the OINSTALL script as usual. Select all components.

If you have not already done so, then install the new license server to be able to license the new components. Copy your new licenses to the licensing etc directory.

Ensure Yours truly, set OMNIHOME to the new value selected during the install.

Install the Probes from the new distribution with the PINSTALL script.

Run the command $OMNIHOME/bin/nco_dbinit –server NCOMS (or your choice of server name). This will create an ObjectServer with default security and automations.

Run nco_xigen to create the interfaces file, and test run $OMNIHOME/bin/nco_objserv

Make sure that you have a user belonging to the ncoadmin group.

Configure process control to only run the ObjectServer. Start PA.

Configure and run a probe to feed events to your ObjectServer.

N.B: if the user does not set OMNIHOME, the default install directory is /opt/netcool/omnibus to better organize Netcool/OMNIbus along with other Netcool components (like Webtop /opt/netcool/webtop and Impact /opt/netcool/impact).

/opt/netcool is therefore the default value of the NCHOME environment variable denoting the Netcool suite home.

3 V7 ObjectServer

3.1 Overview

Netcool/OMNIbus v7 provides an entirely new memory resident database with enriched SQL functionality. The ObjectServer database uses logical, also known as fuzzy, check-pointing for auto-recovery. The initial database is created by a utility using a reference set of default SQL definitions. Once instantiated, the ObjectServer is managed on-line with new configuration utilities, backup automations, and enhanced configuration export/import tool.

3.2 ObjectServer Maintenance

Administration of the ObjectServer at the SQL level is supported from within the new tool nco_config described in Section 3.9.9.8 that supercedes nco_admin, and from the command line utilities nco_sql (unix) and isql.bat (Windows) described in Section 3.2.1.

Each option allows the authorized user to administer a running ObjectServer.

This section contains some Lab exercises using the command line interface to introduce the key concepts, but primarily describes the functionality that will be used interactively in nco_config.

Note. Once an ObjectServer has been taken into use and modified, there is no equivalent restore or rebuild option via .dat and SQL files using the now obsolete nco_migrate utility. Users are advised therefore to take a backup of their ObjectServers at regular intervals and especially so before making any structural modifications to the database. The online backup command is described in section 3.6.1. Configurable automations are provided making use of the backup command to facilitate the creation of regular backup files of the entire ObjectServer that may be restored in the event of catastrophic failure. An enhanced configuration replicator utility provides the means to export and import full or selected ObjectServer definitions easing the task of configuring and maintaining multiple ObjectServer instances.

3.2.1 Command Line Utility

The v3.x command line utilities have been fully upgraded to support the enhanced SQL of the v7 ObjectServer that is described in the following sections.

The utilities are launched as before using the commands:

> UNIX $OMNIHOME/bin/nco_sql -server *`servername`* -user *`username`*
> Windows %OMNIHOME%\bin\isql -s *`servername`* -u *`username`*

In addition to supporting the full ObjectServer SQL, these utilities also provide a number of system options:

- To cancel a command, enter `reset` at the beginning of a new line or Control + C anywhere on a line. Any commands that have not been executed are discarded.
- To run an operating system command, enter `!!` followed by the command (for example, `!!ls`) at the beginning of a new line.
- To run the default editor in `nco_sql`, enter `vi` at the beginning of a new line.
- To read in a file, enter `:r` *`filename`* at the beginning of a new line. Do not include the `go` command in the file. Instead, enter the `go` command at the beginning of a new line.

The `nco_sql` utility is able to accept text files containing SQL commands redirected from the command line. The text file must contain only SQL commands and be terminated with the `go` keyword. For example, to execute the SQL commands in a text file named `myfile.txt` from a UNIX command line, enter the following command:
nco_sql -server OS1 -username *`username`* -password *`password`* < infile.txt
You can also direct the output to a file, for example:
nco_sql -server OS1 -username *`username`* -password *`password`* < infile.txt > output.txt

3.3 ObjectServer Structure

3.3.1 Databases

When you initialize an ObjectServer using the default configuration supplied with nco_dbinit, the following databases are created:

- **alerts**, that contains alert status information, forwarded to the ObjectServer from probes and gateways.
- **security**, that contains information about the security system, including users, roles, groups, and permissions.
- **catalog**, that contains metadata about ObjectServer objects. The ObjectServer maintains this database; you can view, but not modify, the data in it.
- **service**, that is primarily intended to support Netcool/ISMs.
- **custom**, that can be used for tables added by users.
- **transfer,** used by the system during gateway synchronization
- **master and tools**, that are used by the system for compatibility with prior releases of Netcool/OMNIbus. Tables and view in the `master` database also support the Desktop ObjectServer architecture. Administrators should understand and work with the underlying v7 tables not the v3 style views.

The names used with SQL table and column definitions must be unique within the ObjectServer and comply with the naming conventions avoiding the use of Reserved Words.

3.3.2 Tables

New System tables maintained by the ObjectServer have been introduced. These tables contain metadata about ObjectServer objects. System tables are identified by the `catalog` and `security` databases. For example, the `catalog.columns` table contains metadata about all the columns of all the tables in the ObjectServer.

You can view information in the system tables using the `SELECT` and `DESCRIBE` commands, but you cannot add, modify, or delete system tables or their contents using ObjectServer SQL.

Within the data tables a number of table and column sizing limitations have been extended in the new ObjectServer. The maximum number of columns in a table is now 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 K.

The range of data types has been modified and extended. A full list is contained in Section 3.3.4.

3.3.3 Views

Table views are now supported to create a virtual table from selected rows and columns of a real table. The user can also create virtual columns within views, composed using expressions on columns in the underlying table.

Views in v7 are primarily implemented to provide table structures supporting backward compatibility with v3.x. Views are not implemented in the event list views of the Desktop clients.

The ObjectServer does not support “join”.

3.3.4 Column Data Types and Properties

The data types supported by the ObjectServer are listed in the following table.

Note the following changes:

- Several 32 and 64 bit integer types have replaced “int”.
- “timestamp”, “ltime”, “optime” and “opcount” are redundant. All time and count values are now under user control via triggers and associated automations described in later Sections.
- The maximum length of “char” and “varchar” columns is increased to 8192 bytes.
- The combined length of the columns in a row may not exceed 64k.

SQL Type	Description	Default Value
`INTEGER`	32 bit signed integer	0
`INCR`	32 bit unsigned auto-incrementing integer. Applies to table columns only, and can only be updated by the system.	Increments from 1
`UNSIGNED`	32 bit unsigned integer	0
`INTEGER64`	64 bit signed integer	0
`UNSIGNED64`	64 bit unsigned integer	0
`BOOLEAN`	TRUE or FALSE	FALSE
`REAL`	64 bit signed floating point number	0.0
`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
CHAR(`integer`)	Fixed size character string, *integer* 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.	‘’
VARCHAR(*`integer`*)	Variable size character string, up to *integer* 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.	‘’

The properties that may be applied to modify the behaviour of columns within the ObjectServer are shown in the following table.

Column Property	Description
`PRIMARY KEY`	The column is created as a primary key. The primary key column or columns uniquely identify each row.
`NODEFAULT`	The required value of this column must be specified in the initial `INSERT` command.
`NOMODIFY`	The value of this column cannot be changed after the initial `INSERT` command.
`HIDDEN`	The column name of a hidden row 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.

Note: The “UPDATEONDEDUPLICATION” property is no longer required. Deduplication updates are controlled via the trigger automations described later in the workshop.

3.4 Storage Structures and Data Definition Language SQL Commands

In addition to the CREATE DATABASE and DESCRIBE commands provided in v3.6, Netcool/OMNIbus 7 introduces a number of new Data Definition commands. These commands may be issued against a running ObjectServer.

3.4.1 Database SQL

3.4.1.1 Creating a Database

```
CREATE DATABASE database_name;
```

A database is always persistent.

3.4.1.2 Dropping a Database

```
DROP DATABASE database_name;
```

You cannot drop a database if it contains any objects. You cannot drop a system database

3.4.2 Table SQL

3.4.2.1 Creating a Table

```
CREATE TABLE [database_name.]table_name PERSISTENT | VIRTUAL
(column_def,...) [, PRIMARY KEY(column_name,...) ];
```

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

The syntax for the *column_def* column definition in the `CREATE TABLE` command is:

```
column_name data_type [ PRIMARY KEY | NODEFAULT | NOMODIFY | HIDDEN ]
```

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

Example

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

3.4.2.2 Altering a Table

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

You cannot alter system tables.

To add or drop columns from an existing table use the ADD COLUMN and DROP COLUMN settings, respectively. The syntax for the column_def column definition is described in “Creating a Table” above

You cannot add primary keys to an existing table. You cannot drop a column if one or more of the following is true:

- The column is system-initialized (for example, RowSerial)
- The column is a primary key

When dropping a column, it will be the users’ responsibility to ensure that any references to that column are removed from all affected ObjectServers or external objects. If a column is dropped from a table that has views, triggers, procedures, or restriction filters that depend on it, then the dependent objects will be recompiled. Objects that reference the dropped column will be deleted. All affected objects will be logged.

To alter the NOMODIFY, HIDDEN, and NODEFAULT attributes of an existing column set the appropriate attribute to TRUE or FALSE using the ALTER COLUMN setting. A primary key column must have a default value and cannot be hidden. You can change more than one setting in a single ALTER TABLE command.

Example

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

3.4.2.3 Describing a Table

```
DESCRIBE [database_name.]object_name;
```

The output includes the column name, the internal representation of the data type, the length of the column, and whether the column is part of a primary key (1 if TRUE, 0 if FALSE).
Hidden columns are not displayed.

Lab: 2) Describe tables

Run nco_sql
describe catalog.tables;
go
(This command will list the field definitions of the catalog table. Using the main field names from the describe output, you can then list the databases and tables in the catalog:)
Select DatabaseName, TableName from catalog.tables
Order by DatabaseName asc;
Go
(This outputs the table and database names of all tables and views in the ObjectServer)

Lab: 3) Add table

Some Netcool components require the addition of rows and tables to the ObjectServer. SLAM for example requires a services table. In 3.x, you would add the table into NCOMS.sql and run migrate.

Here is the 3.x table definition:

```
create table service_deps
(
    Service       varchar(127),   -- ServiceType + ServiceName
    EventKey      varchar(127),   -- ServerName + ServerSerial
    KeyField      varchar(255),   -- Service + EventKey
    primary key   (KeyField),
    permanent
);
```

In V7, the table can be added via nco_sql or nco_config whilst the ObjectServer is running.

Add this table to your ObjectServer using the Create and Alter Table commands.
Run describe on the catalog, and test that your table has been correctly defined.

3.4.2.4 Dropping a Table

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

You cannot drop a table if it is referenced by other objects, such as triggers, or if it contains any data. You cannot drop system tables.

Lab: 4) Drop table

Drop the table you created in the previous lab. Use describe to check that the table has been removed from the catalog.

3.4.3 View SQL

3.4.3.1 Creating a View

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

The SELECT_cmd is any SELECT command (including aggregate SELECT commands) as described in the Netcool/OMNIbus SQL Reference Guide 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 sub-query containing a WHERE clause in an aggregate SELECT statement.
- You cannot specify virtual columns in an aggregate SELECT statement.
- If you create an aggregate view, you cannot perform an aggregate SELECT on it.
- If you create an aggregate view, you cannot perform an INSERT, UPDATE, or DELETE on it.

If you think that a view already exists with the same name as the one you want to create, or if you want to replace an existing view, use the optional OR REPLACE keywords. An existing view will be replaced by the one you are creating. If the view does not already exist, a new one is created.

The view name must be unique within the database and comply with the 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 a system table.

The storage type is either TRANSIENT or PERSISTENT, depending on 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.
Example

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

Lab: 5) Create view

*Create two views comprising based on the example above. Make one a transient view, the other permanent. Use describe and select * to check the results.*

3.4.3.2 Dropping a View

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

If you do not specify a database name, the view is dropped from the alerts database. A view that is referenced by other objects cannot be dropped.

3.4.4 Restriction Filters

3.4.4.1 Creating Restriction Filters

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

A restriction filter provides a way to restrict the rows that are displayed when a user views a table. Restriction Filters may be applied to a user, or a group. Group filters are applied to all members of the Group.

If you want to replace an existing filter, use the optional OR REPLACE keywords. If the filter does not already exist a new one is created. If you are replacing an existing filter only the condition can be changed. The restriction filter name must be unique and comply with the naming conventions

The condition is an expression or expressions that returns a subset of rows of the table.

A filter is always persistent.

Example

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

The filter controls the data that can be displayed and modified as the restriction filter is automatically applied in the DML commands SELECT, INSERT, UPDATE, DELETE issued by the User or Group members.

3.4.4.2 Dropping a Restriction Filter

```
DROP RESTRICTION FILTER filter_name;
```

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

3.4.5 ObjectServer Files

3.4.5.1 Creating Files

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

An ObjectServer file provides a way to log or report information about ObjectServer events.

The “WRITE INTO” DML command is provided for files that may be accessed as required from within Actions and Procedures. This command is described in Section 3.5.2.

If you think that a file already exists with the same name as the one you want to create, or if you want to replace an existing file, use the optional OR REPLACE keywords. If the file does not already exist a new one is created. If the file already exists it is replaced by the one you are creating.

The file name must be unique and comply with the naming conventions.

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

Note: A number, starting with 1 and incremented depending on the number of files in the file set, is always appended to the specified file name (or file extension if there is one). You can optionally set MAXFILES to specify the number of files in the file set. The default is 1.
If you set MAXFILES to a value greater than 1, when the first file exceeds the maximum size, a new file is created. When that file reaches the maximum size, another new file is created and the process is repeated until the maximum number of files in the set is reached. Then the oldest file is deleted and the process repeats.
You can optionally set MAXSIZE to specify the maximum file size. After a record is written to the file that meets or exceeds that size, a new file is created. The default setting is 0. If set to 0, there is no maximum file size, and therefore the file set always consists of one file. The minimum file size is 1064 Bytes. If the ObjectServer is restarted, new data is appended to the existing file.

The following sequence of files are created and used:

- When the command is executed, the ObjectServer creates an empty file named logfile1 in the specified directory.
- The ObjectServer writes data to logfile1 until it reaches the maximum file size.
- The ObjectServer renames logfile1 to logfile2. It then creates a new logfile1 and writes to it until it reaches the maximum size.
- The ObjectServer renames logfile2 to logfile3 and renames logfile1 to logfile2. It then creates a new logfile1 and writes to it until it reaches the maximum size.
- 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 reaches the maximum size. This sequence is repeated until the file is altered or dropped.

Lab: 6) Create ObjectServer file

create file <your file name> '/log/<your log file>'
maxfiles 3
maxsize 20 Kbytes;

Check that an empty log file has been created. It will be used in a later lab.

3.4.5.2 Altering a File

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

Example

```
alter file log truncate;
```

3.4.5.3 Dropping a File

```
DROP FILE file_name;
```

Dropping a file deletes the ObjectServer file; it does not delete any of the physical files created in the file system.
You cannot drop a file if it is being used, for example, in a trigger.

Example

```
drop file log;
```

3.5 Data Manipulation Language SQL Commands

3.5.1 INSERT, UPDATE, DELETE and SELECT

The basic DML commands UPDATE, DELETE and SELECT now support Views, but are otherwise unchanged. The INSERT command is unchanged and does not support Views.

Note: You cannot assign values to system-maintained columns such as RowSerial and Serial.

You cannot update system-maintained columns such as Serial, or columns where the NOMODIFY attribute is set to TRUE.

There are some restrictions on updates/deletes/selects on views.

If you include virtual columns, you can not update them.
e.g. create view alerts.v1 as select Identifier, Severity + 1 as fred, Node from alerts.status;
update alerts.v1 set fred = 2; // illegal
update alerts.v1 set Node = 'node'; // no error

If you create an aggregate view, you can not perform an aggregate select on it
e.g. create view alerts.m1 as select count(*) as fred from alerts.status;
select count(*) from alerts.m1; // illegal
select fred from alerts.m1; // no error

If you create an aggregate view, you can not perform an update or delete
e.g. create view alerts.m1 as select count(*) as fred from alerts.status;
update alerts.m1 set fred = 3; // illegal
delete from alerts.m1; // illegal

3.5.2 Add to Log File: WRITE INTO

A new WRITE INTO command is introduced to add log messages to an ObjectServer file that has been created with the CREATE FILE command (Section 3.4.5).

Syntax

```
WRITE INTO file_name [ VALUES ] (expression, ...);
```

A carriage return follows each message.
Example

```
WRITE INTO file1 VALUES ('User', %user.user_name, 'connected at',
getdate );
```

This command adds a message to the physical file associated with the ObjectServer file file1 each time a user connects to a database. The %user.user_name user variable used in this example is only available in procedures and triggers.

3.5.3 Operators

The ObjectServer SQL supports an extended range of options to modify the effect of DML statements. The full set of Operator, Function, Expression and Condition statements is listed in the official documentation set.

The following Comparison operators have been added in v7:

Comparison Operator	Description	Example
%= %!= %<>	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';

Comparison Operator	Description	Example
%< %> %<= %>=	Compares the lexicographic relationship between two strings, ignoring case. This comparison determines whether strings come before (%<) or after (%>) other strings alphabetically. You can also find strings which are less than or equal to (%<=) or greater than or equal to (%>=) other strings. For example, aaa comes before AAB because alphabetically aaa is less than (comes before) AAB when the case is ignored.	SELECT * FROM london.status WHERE site_code %< 'UK3';

The syntax of a list comparison expression has been extended:

```
expression comparison_operator { ANY | ALL } ( expression,... )
or
expression [ NOT ] IN ( expression,... )
```

If you use the `ANY` keyword, the list comparison condition evaluates to `TRUE` if the comparison of the left hand expression to the right hand expressions returns `TRUE` for any of the values. If you use the `ALL` keyword, the list comparison condition evaluates to `TRUE` if the comparison of the left hand expression to the right hand expressions returns `TRUE` for all of the values. An `IN` comparison returns the same results as the `=ANY` comparison. A `NOT IN` comparison returns the same results as the `<>ANY` comparison.

The `ANY` and `ALL` operators are not supported in subqueries.
You can use the logical operators NOT AND OR and XOR on boolean values to form expressions that resolve to `TRUE` or `FALSE`.

If an expression contains multiple operators, the ObjectServer uses operator precedence to determine the order in which to evaluate the expression.

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

3.5.4 Functions

The following table lists the extended range of functions supported by Netcool/OMNIbus7.

array_len(*array*)	Returns the number of elements in an array. This function can only be used in procedures or triggers.	If the array myarray has ten elements, array_len(myarray) returns 10.
Ceil(*real*)	Takes a real argument and returns the smallest integral value not less than the argument.	select ceil(myreal) from mytab;

dayasnum(*time*)	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. Sunday is 0, Monday is 1, and so on.	select dayasnum(date_col) from mytab;
dayname(*time*)	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. The output is mon, tue, and so on.	select dayname(date_col) from mytab;
dayofmonth(*time*)	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.	Select dayofmonth(date_col) from mytab;
dayofweek(*time*)	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. Unlike dayasnum, Sunday is 1, Monday is 2, and so on.	select dayofweek(date_col) from mytab;
getdate()	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 alerts.status table that are more than ten minutes old: select Summary, Severity from alerts.status where LastOccurrence < getdate() - 600;
Getenv(*string*)	Returns the value of the specified environment variable.	getenv('OMNIHOME') returns a directory name, for example, /opt/Netcool/omnibus.
hourofday(*time*)	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.	select hourofday(date_col) from mytab;
is_env_set(*string*)	Returns TRUE if the specified environment variable is set; FALSE otherwise.	is_env_set('OMNIHOME') returns TRUE when the OMNIHOME environment variable is set.
Log_2(*real*)	Takes a positive real argument and returns logarithm to base 2.	select log_2(my_real) from mytab;
lower(*string*)	Converts a character string argument into lower case characters.	lower('LIMA') returns lima
minuteofhour(*time*)	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.	Select minuteofhour(date_col) from mytab;
mod(*int1*,*int2*)	Returns the integer remainder of *int1* divided by *int2*.	mod(12,5) returns 2

monthasnum(*time*)	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. January is 0, February is 1, and so on.	Select monthasnum(date_col) from mytab;
monthname(*time*)	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. The output is jan, feb, and so on.	select monthname(date_col) from mytab;
monthofyear(*time*)	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. Unlike monthasnum, January is 1, February is 2, and so on.	Select monthofyear(date_col) from mytab;
Power(*real1*, *real2*)	Takes two real arguments and returns raises *real1* raised to the power of *real2*.	select power(myreal1, myreal2) from mytab;
secondofminute(*time*)	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.	Select secondofminute(date_col) from mytab;
Substr(*string, x, y*)	where string is the input string and x and y are integers. Returns the y-character substring of string that begins at position x. Indexing is consistent with SQL arrays.	
to_char(*argument* [, *conversion_ specification*])	Converts the argument to a string. The argument can be of any data type except a character string. If the argument is a time type, you can specify a second argument consisting of a conversion specification to format the output. This format is determined by the POSIX strptime function.	to_char(73) returns '73'
to_int(*argument*)	Converts the argument to an integer. The argument can be of any data type except integer. If the argument is a string, but it does not contain numeric characters, the function returns 0.	to_int('73') returns 73
to_int32(*argument*)	Converts the argument to a 32-bit integer. The argument can be of any data type except a 32-bit integer. If the argument is a string, but it does not contain numeric characters, the function returns 0.	to_int32('73') returns 73 to_int32('UK') returns 0

to_real(*argument*)	Converts the argument to a 64 bit real number. The argument can be of any data type except real. If the argument is a string, but it does not contain numeric characters, the function returns 0.	to_real('7.3') returns 7.3
to_time(*argument*) to_date(*argument*)	Converts the argument to a time type. The argument can be of any data type except a time type.	
to_unsigned(*argument*)	Converts the argument to a 64 bit unsigned integer. The argument can be of any data type except a 64 bit unsigned integer. If the argument is a string, but it does not contain numeric characters, the function returns 0.	to_unsigned('73') returns 73
upper(*string*)	Converts a character string argument into upper case characters.	upper('Vancouver') returns VANCOUVER
year(*time*)	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.	select year(date_col) from mytab;

3.6 System and Session SQL Commands

The ObjectServer now includes a system command, ALTER SYSTEM, that enables you to change the default and current settings of a running ObjectServer. The various options of this command replace a number of single commands, for example:

```
DUMP is replaced by ALTER SYSTEM BACKUP
SHUTDOWN by   ALTER SYSTEM SHUTDOWN
PASSWORD by   ALTER USER xxxx SET PASSWORD xxxxxxxxx
```

3.6.1 Managing the ObjectServer: ALTER SYSTEM

Use the ALTER SYSTEM command to change the settings for ObjectServer properties, drop user connections, or to shut down or back up the ObjectServer.

Syntax

```
ALTER SYSTEM
{
SHUTDOWN |
SET 'property_name' = value |
DROP CONNECTION connection_id [, ... ] |
BACKUP 'directory_name'
}
;
```

You can set ObjectServer properties with the ALTER SYSTEM SET command. 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.

3.6.1.1 ObjectServer Shutdown

You can stop the ObjectServer with the ALTER SYSTEM SHUTDOWN command. After processing the SHUTDOWN command, nco_sql will allow you to exit cleanly.

Lab: 7) Shutdown the ObjectServer

Run nco_sql,
Enter the commands
alter system shutdown;
go
The ObjectServer will shutdown cleanly, and return to the SQL command prompt. Enter exit to exit nco_sql and return to the unix command line.
Restart the ObjectServer ready for later labs.
Check that the persistent view that you defined earlier remains in the ObjectServer. The transient view should have been deleted.

3.6.1.2 ObjectServer Backup

You can back up the ObjectServer with the ALTER SYSTEM BACKUP command by specifying an existing directory name in quotes.

Note: The directory cannot be the one in which ObjectServer data files are stored, which is set to $OMNIHOME/db/server_name by default. The backup generates copies of the ObjectServer .tab files in the specified directory.

To recover the ObjectServer to the point in time at which the BACKUP command was issued, move the copies of the ObjectServer .tab files into the ObjectServer data file directory. The backup files can only be used on a machine with the same architecture as the machine on which they were created.

Note. Once an ObjectServer has been taken into use and modified, there is no equivalent restore or rebuild option via .dat and SQL files with nco_migrate. Users are strongly advised therefore to take a backup of their ObjectServers at regular intervals and especially so before making any structural modifications to the database.

3.6.1.3 Drop User connections

You can drop user connections via the ALTER SYSTEM DROP CONNECTION command. First locate the connection identifiers for the connections by select from the catalog.connections table. Specify the connections that you wish to drop in a comma separated list within the DROP command.

Lab: 8) Drop User Connections

*Open an event list or other desktop client. In nco_sql, select * from catalog.connections to identify the client. Drop that connection using the ALTER command described above. You should quickly see that your client has lost its connection to the ObjectServer.*

3.6.2 Changing the Default Database: SET | USE DATABASE

The USE option has been added to the DATABASE command.

The SET DATABASE and USE DATABASE commands perform the same function. After you set the current database with the SET DATABASE or USE DATABASE command in an nco_sql session, you can specify an object name without preceding it with the database name. The current database setting lasts for the length of the session in which it is set.

Syntax

```
{ SET | USE } DATABASE database_name;
```

You cannot use this command in triggers or procedures.

3.6.3 Verifying SQL Syntax: CHECK STATEMENT

The CHECK STATEMENT command parses and checks the syntax of the SQL commands entered between quotes and returns either a success message or a description of any errors.
Syntax

```
CHECK STATEMENT 'command; command; ...';
```

Note: Because CHECK STATEMENT does not execute the SQL commands, runtime errors are not detected. Additionally, some spurious errors may be displayed if there is series of commands that relies on the preceding commands being executed.

3.7 Security and Associated SQL Commands

3.7.1 Controlling Netcool/OMNIbus Component Security

Netcool/OMNIbus 7 incorporates an extended security model.

Each Netcool/OMNIbus component and object has different actions associated with it. These actions can be allowed and disallowed by granting or revoking permissions associated with each component or object. The available "permissions" are defined within the ObjectServer on creation.

3.7.2 Administering Users, Groups, and Roles

Permissions control access to objects and data in the ObjectServer.
Roles are defined as sets of permissions.
Roles are the assigned at the Group level. Users that are members of a group inherit the roles defined for that group.

- A Permission may be assigned to one or more Roles.
- A Role may be assigned to one or more Groups.
- A User may be assigned to one or more Groups.

The concept of User Types (Super-User, Administrator, and Normal) is retained for backward compatibility with Version 3.x. In v7 the User Type is only relevant in relation to the Alert Security Model and is set automatically by the system depending on the Groups to which the user is assigned. A User assigned to the System group will be defined as type Super-User; whereas a User assigned to the Administrator Group will be given a type of Administrator unless they are also a member of the System Group.

The default configuration creates a set of Groups Roles and Permissions that aim to reflect the permissions previously hard linked to the User Types. The v7 system can be modified to create new group and role structures to suite individual requirements. Changes of permissions will modify the user's ability to use or modify components of the system, but will not have any effect on the application of the Alert Security Model which remains linked to User Type.

The Permissions structure is highly granular to support future development.

3.7.3 Users

3.7.3.1 CREATE USER

Use the `CREATE USER` command to add a user to the ObjectServer.
Syntax

```
CREATE USER 'user_name'
[ ID identifier ]
FULL NAME 'full_user_name'
[ PASSWORD 'password' [ ENCRYPTED] ]
[ PAM { TRUE | FALSE } ]
```

The user_name is a text string containing the name of the user being added.

Note: User, group, and role names are case-sensitive, and must be specified in quotes.
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.
The full_user_name is a text string containing a descriptive name for the user. You can specify the user password using the PASSWORD keyword. The default is an empty string. If you add the keyword ENCRYPTED, the password is assumed to be encrypted.

If you are using Pluggable Authentication Modules (PAM) to authenticate the user, set PAM to TRUE. You must also set the Sec.UsePam property to TRUE for the ObjectServer.

Example

```
create user 'joe' id 1 full name 'Joseph R. User';
```

The user types NORMAL, ADMINISTRATOR, or SUPERUSER are retained for backward compatibility with the desktop Alert Security Model. Within v7 UserType is a hidden field that will be set according to the users Group membership. When a user is first created the user type is set to NORMAL. If a User is added to or later removed from either the ADMIN or SYSTEM groups, then the hidden User Type will be modified appropriately.

For example, if a user is added to the ADMIN group, their UserType will be set to Administrator. A User added to the SYSTEM group will become UserType SuperUser. If the User is removed from either of these groups, their UserType will be set down to a level appropriate to their remaining group membership.

3.7.3.2 ALTER USER

Use the ALTER USER command to change user settings, such as the password, for the specified user in the security repository.
Syntax

```
ALTER USER 'user_name'
SET PASSWORD 'password'
       [ AUTHORIZE PASSWORD 'old_password' ]
       [ ENCRYPTED ]
[ SET FULL NAME 'full_user_name' ]
[ SET ENABLED { TRUE | FALSE } ]
[ SET PAM { TRUE | FALSE } ]
[ ASSIGN [ RESTRICTION ] FILTER restriction_filter_name ]
[ REMOVE [ RESTRICTION ] FILTER restriction_filter_name ] ;
```

The user_name is a text string containing the name of the user being altered.
Use the PASSWORD setting to change the password for the specified user. When changing a password stored on an external system and accessed using PAM, you must also specify the old password following the AUTHORIZE PASSWORD keywords.

Use the ENABLED setting to activate or deactivate the specified user.

Use the PAM setting to enable or disable the use of PAM to authenticate the specified user.

Use the ASSIGN or REMOVE RESTRICTION FILTER settings to assign or remove the restriction filters that apply to the user.

Tip: Only one restriction filter per table can be applied to a user.
You can change more than one setting in a single ALTER USER command.

Example

```
alter user 'joe' set password 'topsecret';
```

3.7.3.3 DROP USER

Use the DROP USER command to drop the specified user from the security repository.
Syntax

```
DROP USER 'user_name';
```

The user_name is a text string containing the name of the user being dropped.
Example

```
drop user 'joe';
```

3.7.4 Groups

3.7.4.1 CREATE GROUP Command

Use the CREATE GROUP command to create a group. You can then add one or more users to the group.

Syntax

```
CREATE GROUP 'group_name'
[ID identifier]
[ COMMENT 'comment_string' ]
[ MEMBERS 'member_name', ... ] ;
```

The group_name is a text string containing the name of the group being created.
Note: User, group, and role names are case-sensitive, and must be specified in quotes. The identifier is an integer value that uniquely identifies the group. If you do not specify an identifier, one is automatically assigned. The Public group has the identifier 0, and all users are part of this group. The System group has the identifier 1.
Use the optional COMMENT setting to add a description of the group you are creating.
When you create a group, you can specify one or more group members following the MEMBERS keyword.

Example

```
create group 'MyAdmin' id 3 COMMENT 'AutoAdmin group' members 'joe',
'bob';
```

3.7.4.2 ALTER GROUP

Use the ALTER GROUP command to change user settings, such as the included users, for the specified group.
Syntax

```
ALTER GROUP 'group_name'
[ SET COMMENT 'comment_string' ]
[ ASSIGN [ RESTRICTION ] FILTER 'restriction_filter_name' ]
[ REMOVE [ RESTRICTION ] FILTER 'restriction_filter_name' ]
[ ASSIGN MEMBERS 'user_name', ... ]
[ REMOVE MEMBERS 'user_name', ... ];
```

The group_name is a text string containing the name of the group being altered.
Use the ASSIGN or REMOVE RESTRICTION FILTER setting to assign or remove restriction filters which apply to the group. Only one restriction filter per table can be applied to a group.
Use the COMMENT setting to modify the description of the group.
Use the ASSIGN or REMOVE MEMBERS setting to assign or remove members of the group.
Example

```
alter group 'admingroup' assign members 'root';
```

3.7.4.3 DROP GROUP

Use the DROP GROUP command to drop the specified group from the security repository.
Syntax

```
DROP GROUP 'group_name';
```

The group_name is a text string containing the name of the user being dropped.

Example

```
drop group 'LondonAdmin';
```

3.7.5 Roles

3.7.5.1 CREATE ROLE

Use the CREATE ROLE command to create a role, which will be a collection of permissions. You can assign a role to any number of Groups to 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.
Syntax

```
CREATE ROLE 'role_name'
[ ID identifier ]
[ COMMENT 'comment_string' ];
```

The role_name is a text string containing the name of the role being added.
Note: User, group, and role names are case-sensitive, and must be specified in quotes.
The identifier is an integer value that uniquely identifies the role. 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. If you do not specify an identifier, one is automatically assigned.
Use the optional COMMENT setting to add a description of the role you are creating.
Example

```
create role 'superadmin' id 500
comment 'only users with root access should be assigned this role';
```

3.7.5.2 ALTER ROLE

Use the ALTER ROLE command to change the comment for an existing role.

Syntax

```
ALTER ROLE 'role_name' SET COMMENT 'comment_string' ;
```

The role_name is a text string containing the name of the role being altered.
Use the COMMENT setting to modify the description of the role.
Example

```
alter role 'admin' set comment 'enhanced description of role';
```

3.7.5.3 DROP ROLE

Use the DROP ROLE command to drop an existing role.
Syntax

```
DROP ROLE 'role_name';
```

The role_name is a text string containing the name of the role being dropped.
Example

```
drop role 'admin';
```

3.7.5.4 GRANT ROLE

Use the GRANT ROLE command to grant a role to one or more groups. Role grants take effect immediately.

Syntax

```
GRANT ROLE 'role_name',...
TO { GROUP 'group_name',...};
```

The role_name is a text string containing the name of a role being granted to one or more groups.

Example

```
grant role 'administrator' to group 'admin';
```

3.7.5.5 REVOKE ROLE

Use the REVOKE ROLE command to revoke one or more roles from one or more Groups.

Syntax

```
REVOKE ROLE 'role_name',...
FROM { Group 'role_name',... };
```

The role_name is a text string containing the name of a role being revoked from one or more groups.

Example

```
revoke role 'administrator' from group 'admin';
```

3.7.6 Granting and Revoking System and Object Permissions

This section describes how to grant and revoke system and object permissions. System permissions control which commands can be executed in the ObjectServer. Object permissions control access to individual objects, such as tables.

The default structure can be found in $OMNIHOME/install/dbcore/security.sql which shows the default grant statements supplied with the ObjectServer.

3.7.6.1 GRANT Command

Use the GRANT command to grant system and object permissions to roles.

Syntax for Granting System Permissions

```
GRANT system_permission,...
TO { ROLE 'role_name' ,... }
[ WITH GRANT OPTION ];
```

A system_permission is any of the following:

```
ISQL |
ISQL WRITE |
CREATE MEMSTORE |
CREATE DATABASE |
CREATE FILE |
CREATE RESTRICTION FILTER |
CREATE SQL PROCEDURE |
CREATE EXTERNAL PROCEDURE |
CREATE SIGNAL |
CREATE TRIGGER GROUP |
ALTER SYSTEM SHUTDOWN |
ALTER SYSTEM BACKUP |
ALTER SYSTEM SET PROPERTY |
ALTER SYSTEM DROP CONNECTION
CREATE USER |
CREATE GROUP |
CREATE ROLE |
ALTER USER |
ALTER GROUP |
ALTER ROLE |
DROP USER |
DROP GROUP |
DROP ROLE |
GRANT ROLE |
REVOKE ROLE
```

Example

```
grant create table to role 'administrator';
```

Syntax for Granting Object Permissions

```
GRANT object_permission,... ON permission_object object_name
TO { ROLE 'role_name',... }
[ WITH GRANT OPTION ];
```

A permission_object is any of the following:

```
DATABASE |
MEMSTORE |
TABLE |
VIEW |
RESTRICTION FILTER |
FILE |
TRIGGER |
TRIGGER GROUP |
SQL PROCEDURE |
EXTERNAL PROCEDURE |
SIGNAL
```

Each of these objects has permissions associated with it. The owner of each object automatically has grant and revoke permissions associated with that object, and can grant and revoke those permissions (individually) to other users and roles. The creator of an object owns the object.

The WITH GRANT OPTION option enables the group members assigned the Role to grant the permission to other roles.

The following table shows the mapping between objects and the permissions the owner has and is able to grant to others.

Objects	Permissions
Database	DROP CREATE TABLE CREATE VIEW
Memstore	DROP ALTER
Table	DROP ALTER SELECT INSERT UPDATE DELETE

View	DROP ALTER SELECT UPDATE DELETE
Trigger group	DROP ALTER CREATE TRIGGER
Trigger	DROP ALTER
File	DROP ALTER WRITE
SQL procedure External procedure	DROP ALTER (permission require for CREATE or REPLACE) EXECUTE
Signal	DROP ALTER (permission require for CREATE or REPLACE) RAISE
Restriction Filter	DROP ALTER (permission require for CREATE or REPLACE)

Example

```
grant drop on database testdb to role 'dba_admin';
```

3.7.6.2 REVOKE Command

Use the `REVOKE` command to revoke system and object permissions from users and roles.

Syntax for Revoking System Permissions

```
REVOKE system_permission,...
FROM { ROLE 'role_name',... };
```

Example

```
revoke create table from role 'db_admin';
```

Syntax for Revoking Object Permissions

```
REVOKE object_permission,...
ON permission_object object_name
FROM { ROLE 'role_name',... };
```

Example

```
revoke drop on database testdb from role 'db_admin';
```

3.7.6.3 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.

Parent Object	Child Objects
System	DATABASE TRIGGER GROUP FILE SQL PROCEDURE EXTERNAL PROCEDURE SIGNAL RESTRICTION FILTER
Database	TABLE VIEW
Trigger group	TRIGGER

3.8 Procedures

A procedure is an executable SQL object that can be called to perform common operations. Once you create a procedure in the ObjectServer, you can execute it from nco_sql or within a trigger. The types of procedures are:

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

3.8.1 Creating SQL Procedures

An SQL procedure is a set of parameterized SQL commands, or code fragments, with programming language constructs that enable you to perform complex tasks on database objects. You can create a procedure containing a logical set of commands, such as a set of queries, updates, or inserts, that make up a task.
Procedures expand SQL syntax so you can:

- pass parameters into and out of a procedure
- create local variables and assign values to them
- perform condition testing
- perform scanning operations over tables and views

Use the CREATE PROCEDURE command to create SQL procedures. This command defines the structure and operation of the procedure, including the types of parameter passed into and out of the procedure, and the local variables, condition testing, row operations, and assignments performed in the procedure.

Syntax:

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
([ procedure_parameter,... ])
[ DECLARE variable_declaration;...[;] ]
BEGIN
procedure_body_statement;...[;]
END
```

If you think that a procedure already exists with the same name as the one you want to create, or if you want to replace an existing procedure, use the optional OR REPLACE keywords. If the procedure already exists it is replaced by the one you are creating. If the procedure does not already exist a new one is created.

The procedure_name must be unique within the ObjectServer and comply with the naming conventions. Once created, a procedure can be explicitly executed by a user, application, or trigger using the EXECUTE PROCEDURE command described in Section 3.8.3.2.

3.8.1.1 Elements of an SQL Procedure

This section provides an overview of the structure of an SQL procedure. The detailed syntax of each component is described in the following sections.
SQL procedures have the following major components:

- parameters
- local variable declarations
- procedure body

Parameters are values passed into or out of a procedure. You declare the parameters of the procedure when you create the procedure and specify what values are passed as parameters when you execute the procedure. The name of the variable that contains a parameter is called a formal parameter, while the value of the parameter when the procedure is executed is called an actual parameter.
The values you pass to the procedure must be of the same data type as in the parameter declaration.
Example
In the procedure declaration:

```
CREATE PROCEDURE calculate_average_severity
( IN current_severity ARRAY OF INTEGER )
```

The formal parameter is the array variable current_severity. When you execute the procedure, you pass an actual parameter. For example, in the procedure call:

```
EXECUTE PROCEDURE calculate_average_severity(3,4,5);
```

The actual parameter is the array of integers 3,4,5, which is assigned to the formal parameter current_severity.

You can also create local variables for use within the procedure to hold and change temporary values in the body of the procedure. Local variables and values are always discarded when the procedure exits. For example, you can create an integer counter as a local variable.

Note: Because both parameters and local variables contain data that can change, both parameters and local variables are referred to as variables within procedures.
The body of a procedure contains a set of statements that test conditions and manipulate data in the database. The body of a procedure is enclosed within the keywords BEGIN and END.

3.8.1.2 SQL Procedure Parameters

Use the procedure_parameters that make up the parameter declaration to specify the parameters that can be passed into or out of a procedure.

Syntax

```
[ IN | OUT | IN OUT ] parameter_name parameter_type
```

Each procedure parameter has a mode, which can be IN, OUT, or IN OUT.

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. This is the default if you do not specify the parameter mode.

An OUT parameter is a write-only variable. You can use an OUT parameter to assign a value to the parameter, but you cannot read from it within the body of the procedure. Therefore, this type of parameter cannot be used in an expression.

An IN OUT parameter is a read and write variable, with none of the constraints of an IN or OUT parameter.

The parameter_name must be unique within the procedure and comply with the naming conventions. The parameter_type defines the type of data the parameter can pass into or out of the procedure. A parameter can be one of the following types:

Syntax

```
parameter_type |
ARRAY OF parameter_type |
```

A parameter_type is any valid ObjectServer data type except VARCHAR or INCR.

Example

```
CREATE PROCEDURE add_or_concat
( IN counter INTEGER, IN one_char_string CHAR(1))
```

An ARRAY OF parameter_type is an array of any valid parameter type.

3.8.1.3 SQL Procedure Variable Declarations

In the optional DECLARE section of a procedure, you can define (declare) local variables for use within a procedure. A local variable is a placeholder for values used during the execution of the procedure.

Syntax

```
DECLARE base_variable_declaration;...
```

Local variable declarations within a procedure must be separated by semi-colons. Local variable names must be unique within the procedure and comply with the naming conventions. A base_variable_declaration creates a simple local variable or array variables.

Syntax

```
variable_name variable_type
```

```
variable_name variable_type [ ARRAY ] [ integer ]
```

Define the size of an array by specifying an integer value greater than 1 in square brackets.

A variable_type is any valid ObjectServer data type except VARCHAR or INCR.

Example

```
DECLARE SeverityTooHigh BOOLEAN;
DECLARE NodeNameArray integer [20]
```

3.8.1.4 SQL Procedure Body

The body of a procedure contains a set of SQL statements and programming constructs that manipulate data in the ObjectServer. The body of a procedure is enclosed within the keywords BEGIN and END. Each statement, except the last one, must be separated by a semi-colon. You can execute the following SQL commands in a procedure:

```
ALTER SYSTEM BACKUP
ALTER SYSTEM SET
ALTER SYSTEM DROP CONNECTION
ALTER TRIGGER
ALTER TRIGGER GROUP
ALTER USER
UPDATE
INSERT
DELETE
WRITE INTO
RAISE SIGNAL
{ EXECUTE | CALL } PROCEDURE
```

You can use the following additional programming constructs, described next, in the procedure body:

```
SET assignment statement
IF THEN ELSE statement
CASE WHEN statement
FOR EACH ROW loop
FOR loop
BREAK
CANCEL
```

The BREAK command exits from the current loop. Execution continues with the next statement in the procedure.

The CANCEL command stops the execution of a procedure.

Warning: Do not use the CANCEL command when using a desktop ObjectServer in DualWrite mode.

3.8.1.5 Set Statement

Use a SET assignment statement to write the value of an expression to a variable or parameter.
Syntax

```
SET { parameter_name | variable_name } = expression
```

Note: The value returned by the expression must be of a type compatible with the variable into which you write the value.
You can assign a value to a column within a row, to a variable, or to an array variable. If you assign a value to a column in a row using the FOR EACH ROW loop, the variable must refer to a row parameter name.
Example
If you have a row variable named alert_row, which contains a column named Severity, you can assign a new severity value to the row with the statement:

```
SET alert_row.Severity = alert_row.Severity + 1;
```

3.8.1.6 IF THEN ELSE Statement

The IF THEN ELSE statement performs one or more actions based on the specified conditions.
Syntax

```
IF condition THEN action_command_list
```

```
[ ELSEIF condition THEN action_command_list ]
...
[ ELSE action_command_list ]
END IF;
```

If the first condition is met (evaluates to TRUE), the commands following the THEN keyword are executed in sequence until an ELSEIF, ELSE, or END IF keyword is reached. If the first condition is not met and there is an ELSEIF statement for which the condition is met, the commands following that ELSEIF statement are executed until the next keyword is reached. If an ELSE statement exists, and no previous conditions have been met, the statements following the ELSE are executed until the END IF is reached.

3.8.1.7 CASE WHEN Statement

The CASE WHEN statement performs one or more actions based on a condition. If the condition is not met, you can optionally perform a different action.

Syntax

```
CASE
WHEN condition THEN action_command_list
...
[ ELSE action_command_list ]
END CASE;
```

If the first condition is met (evaluates to TRUE), the statements following the THEN keyword are executed in sequence until a WHEN, ELSE, or END CASE is reached. Otherwise, if there is any WHEN statement for which the condition is met, the statements following the THEN keyword are executed until a WHEN, ELSE, or END CASE is reached. If no previous condition is met and there is an ELSE statement, the statements following the ELSE are executed until an END CASE is reached.

3.8.1.8 FOR EACH ROW Loop

The FOR EACH ROW loop performs actions on a set of rows that match a certain condition.

Syntax

```
FOR EACH ROW variable_name in database_name.table_name [ WHERE condition ]
BEGIN
procedure_body_statement_list;
END;
```

In this statement the variable name is declared implicitly as a row reference in a database table. 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 target table. 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 procedure body.

If you include a WHERE clause, only rows meeting the criteria specified in the condition are returned.

Example

To increase the severity of all alerts in the alerts.status table that have a severity of 3 to a severity of 4:

```
FOR EACH ROW alert_row in alerts.status WHERE alert_row.Severity=3
BEGIN
SET alert_row.Severity = 4;
END;
```

When this statement is executed, the ObjectServer reads each row of the alerts.status table and tests to see if the value of Severity is 3. For each row that matches this condition, the statements within the BEGIN and END are executed, until all the rows are processed.

Note: You cannot perform an insert DML command on the named table inside the body of a FOR EACH ROW loop. You may execute update and delete statements but must take care to avoid affecting rows matching the WHERE condition that have not yet been processed in the FOR EACH ROW loop.

The following generates an error because the ObjectServer is already scanning the alerts.status table for rows when the DELETE statement is executed on the same table:

```
FOR EACH ROW temporary_row IN alerts.status
BEGIN
DELETE FROM alerts.status WHERE temporary_row.Identifier=Identifier.
END;
```

The FOR EACH ROW loop may also be applied to an array created with the evaluate clause of a trigger

3.8.2 FOR Loop

The FOR loop performs actions a set number of times, based on a counter variable. :
Syntax

```
FOR counter = 1 to integer DO
BEGIN
procedure_body_statement_list;
END;
```

Example
This procedure, when passed an array of valid Identifier fields, updates the alerts.status table and sets the acknowledged flag to TRUE:

```
CREATE PROCEDURE ACKNOWLEDGE_TOOL( ids ARRAY OF CHAR(255) )
DECLARE
k INTEGER;
BEGIN
FOR k = 1 TO array_len( ids ) DO
BEGIN
UPDATE alerts.status VIA ( ids[k] ) SET Acknowledged = TRUE;
END;
END;
```

3.8.2.1 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.

The % symbol indicates that you are referencing an implicit variable. The user keyword references the current user. The available attributes are listed in the following table:

Variable	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, PROBE).
%user.description	STRING	A useful name supplied by some the applications, eg, the probes where it's "nco_p_...". The use of this new field will increase as new application versions are implemented.
%user.host_name	STRING	Name of the connected host.
%user.connection_id	UNSIGNED	Connection identifier.
%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.

3.8.3 External Procedures

3.8.3.1 Creating External Procedures

You can create external procedures to run an executable file on a local or remote system.
Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
( [ procedure_parameter,... ] )
EXECUTABLE executable_name
HOST host_name
USER user_name
GROUP group_name
[ ARGUMENTS expression,... ] [;]
```

If you think that a procedure already exists with the same name as the one you want to create, or if you want to replace an existing procedure, use the optional OR REPLACE keywords. If the procedure already exists it is replaced by the one you are creating. If the procedure does not already exist a new one is created.
The procedure_name must be unique within the ObjectServer and comply with the naming conventions.

Implicit user variables that may be referenced from within the procedure are listed in section 3.8.2.1 above.

Use the procedure_parameters that make up the parameter declaration to specify the parameters that can be passed into the external procedure.
Syntax

```
parameter_name parameter_type
```

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_name must be unique within the procedure and comply with the naming conventions.
The parameter_type defines the type of data the parameter can pass into the procedure. A parameter can be one of the following types:
Syntax

```
parameter_type |
ARRAY OF parameter_type |
ROW OF database_name.table_name
```

A parameter_type is any valid ObjectServer data type except VARCHAR or INCR.

The executable is the path to an executable on a local or remote file system.

The host is the host on which to execute the procedure executable.

The user is the effective user ID under which to execute the executable.

The group is the effective group ID under which to execute the executable.

Note: In order to execute an external procedure, you must have a process control agent daemon (nco_pad) running.

3.8.3.2 Executing a Procedure

Once you have created a procedure, you must execute it using the EXECUTE PROCEDURE command for the actions in the procedure to occur. You can do this using nco_sql or from within a trigger.
Syntax

```
{ EXECUTE | CALL } [ PROCEDURE ] procedure_name
[ ( expression,... ) | ( [ expression, expression,... ] ,... ) ];
```

Each of the expressions passed as actual parameters must resolve to an assignable value which matches the type of the parameter specified when the procedure was created. If you are passing an array parameter, the square brackets around the expression list, shown in bold type, are not optional.

When passing a string field that may contain spaces as a parameter then you must explicitly enclose the field in single quotes, for example:

```
execute test_proc( '\'' + row.Summary + '\'', '\'' + row.Node + '\'' );
```

3.8.4 Dropping a Procedure

If you no longer need a procedure, you can remove it using the DROP PROCEDURE command.

Syntax

```
DROP PROCEDURE procedure_name;
```

You cannot drop a procedure if it is referenced by other objects, such as triggers.

3.9 Automations

The automation sub-system has been significantly enhanced by the introduction of Procedural SQL within triggers.

Triggers allow the ObjectServer to automatically execute an action (fire) when an incident of interest occurs. When the ObjectServer detects an incident associated with a trigger, it executes the trigger action.

A database trigger is configured to implicitly fire (execute an action) when a triggering database incident occurs. For example, a default trigger is supplied to perform an action each time a reinsert occurs (an attempt is made to insert a row into a table, but a row with the same value for the primary key already exists) on the alerts.status table. This trigger replaces the previously built-in deduplication functionality allowing the user to modify the action as required.

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

A signal trigger is configured to fire when a system or user signal is raised. For example, you can send an email to an operator when the ObjectServer starts or stops because these are system signals. A user signal is one you define and then raise using the RAISE SIGNAL command.

Every trigger belongs to a trigger group so you can create a collection of related triggers. You can then enable or disable all triggers at an individual or group level. Disabling a group disables all triggers in that group regardless of the trigger setting. Enabling a group enables all triggers in the group that have not been disabled at the trigger level.

In the following sections, the syntactic elements common to all types of triggers are described. Then the elements unique to each type of trigger are described in more detail.

3.9.1 Trigger Groups

Every trigger belongs to a trigger group. This enables you to create a collection of related triggers that may then be activated | deactivated as a group. This functionality may be used to manage the activation of triggers after failover in a resilient pair of ObjectServers.

3.9.1.1 The CREATE TRIGGER GROUP Command

Use the CREATE TRIGGER GROUP command to create a new trigger group.

Syntax

```
CREATE TRIGGER GROUP trigger_group;
```

Example

```
create trigger group update_database_triggers;
```

3.9.1.2 The ALTER TRIGGER GROUP Command

Use the ALTER TRIGGER GROUP command to activate or deactivate an existing trigger group.

Syntax

```
ALTER TRIGGER GROUP trigger_group
SET ENABLED { TRUE | FALSE };
```

Example

```
alter trigger group update_database_triggers set enabled false;
```

3.9.1.3 Removing a Trigger Group: The DROP TRIGGER GROUP Command

Use the DROP TRIGGER GROUP command to drop an existing trigger group.
Syntax

```
DROP TRIGGER GROUP trigger_group;
```

You cannot drop a trigger group if it contains any triggers.

Example

```
drop trigger group update_database_triggers;
```

3.9.1.4 Default Trigger Groups

The default ObjectServer configuration includes a set of Trigger Groups. The names of the Trigger Groups can be listed with the command

select GroupName from catalog.trigger_groups;

The default groups are

- compatibility_triggers
  A group that has only one member trigger that provides functionality for backward compatibility with v3. This group need only be enabled if a Webtop server is connected to the ObjectServer and Group restriction filters have been enabled for the Webtop users.
- system_watch
  includes triggers that react to system events, e.g. server start-up and shutdown
- default_triggers
  includes the main triggers for system operations, e.g. deduplication, generic_clear.
- connection_watch
  includes the connection_watch triggers.
- security_watch
  includes triggers for managing user sessions including example triggers for disabling user accounts after successive password failures.
- audit_config
  includes triggers that fire when a system object is subject to an “alter” command
- profiler_triggers
  includes triggers that provide the ObjectServer client profiling capability.
- trigger_stat_reports
  includes triggers that provide the ObjectServer trigger profiling capability.
- stats_triggers
  includes triggers to provide the v3 style statistics gathering capability.
- automatic_backup_system
  includes triggers that provide server backup functionality.

3.9.2 Triggers

3.9.2.1 Syntax Elements Common to All Types of Triggers

Database, temporal, and signal triggers have many common syntax elements, described in this section.

Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string' ]
...trigger syntax depending on the type of trigger...
[ WHEN condition ]
[ DECLARE variable_declaration ]
```

```
trigger_action
```

The trigger_name must be unique within the ObjectServer and comply with the naming conventions.

The group_name can be any trigger group already created using the CREATE TRIGGER GROUP command.

If DEBUG is set to TRUE, debugging information is sent to the ObjectServer message log.

If ENABLED is set to TRUE, the trigger fires when the associated incident occurs. Otherwise, the trigger does not fire when the incident occurs.

A trigger's PRIORITY determines the order in which the ObjectServer fires triggers when more than one trigger is associated with the same incident. If more than one trigger of the same priority is attached to a single signal, the order in which the triggers fire is undetermined. 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.

The optional COMMENT attribute enables you to add a comment for the trigger.

The syntax that follows the comment and precedes the optional WHEN attribute and the action depends on the trigger type. Further explanation is contained in the later sections describing Database, Signal, and Temporal triggers.

The optional WHEN attribute allows you to test for a particular condition before the action is executed. If the condition is not met, the action is not executed.

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 from previous executions of the trigger.

The trigger_action in a trigger is a set of statements that are executed when the trigger is fired.

Syntax

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

3.9.2.2 Executing Commands in Trigger Actions

The trigger_statement_list contains a set of commands that manipulate data in the ObjectServer. You can execute the following SQL commands in a trigger:

```
ALTER SYSTEM BACKUP
ALTER SYSTEM DROP CONNECTION
ALTER SYSTEM SET
ALTER TRIGGER
ALTER TRIGGER GROUP
ALTER USER
UPDATE
INSERT
DELETE
WRITE INTO
RAISE SIGNAL
{ EXECUTE | CALL } PROCEDURE
```

You can use the following additional programming constructs in a trigger:

```
SET assignment statement
IF THEN ELSE statement
CASE WHEN statement
FOR EACH ROW loop
FOR loop
```

```
break
cancel
```

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.
Note: The value returned by the expression must be of a type compatible with the variable into which you write the value.

The IF THEN ELSE statement performs one or more actions based on the specified conditions.
Syntax

```
IF condition THEN action_command_list
[ ELSEIF condition THEN action_command_list ]
...
[ ELSE action_command_list ]
END IF;
```

If the first condition is met (evaluates to TRUE), the commands following the THEN keyword are executed in sequence until an ELSEIF, ELSE, or END IF is reached. If the first condition is not met and there is an ELSEIF statement for which the condition is met, the commands following that ELSEIF statement are executed until the next keyword is reached. If an ELSE statement exists, and no previous conditions have been met, the statements following the ELSE are executed until the END IF is reached.

The CASE WHEN statement performs one or more actions based on a condition. If the condition is not met, you can optionally perform a different action.
Syntax

```
CASE
WHEN condition THEN action_command_list
...
[ ELSE action_command_list ]
END CASE;
```

If the first condition is met (evaluates to TRUE), the statements following the THEN keyword are executed in sequence until a WHEN, ELSE, or END CASE is reached. Otherwise, if there is any WHEN statement for which the condition is met, the statements following the THEN keyword are executed until a WHEN, ELSE, or END CASE is reached. If no previous condition is met and there is an ELSE statement, the statements following the ELSE are executed until an END CASE is reached.
The FOR EACH ROW loop performs actions on a set of rows that match a certain condition.
Syntax

```
FOR EACH ROW variable_name in database_name.table_name
[ WHERE condition ]
BEGIN
action_command_list;
END;
```

In this statement the variable name is declared implicitly as a row reference. Therefore you do not need to declare the variable at the start of the procedure. This means that any changes made to the columns referenced by the variable directly affect the referenced rows in the ObjectServer. When the END is reached, the implicitly declared variable is discarded and cannot be used elsewhere in the procedure.
If you include a WHERE clause, only rows meeting the criteria specified in the condition are returned.
The FOR EACH ROW statement may also be applied to the rows of an array variable defined earlier in the procedure.

The FOR loop performs actions a set number of times, based on a counter variable.
Syntax

```
FOR counter = 1 to integer DO
BEGIN
action_command_list;
END;
```

A BREAK command exits from the current loop. Execution continues with the next statement in the procedure.

A CANCEL command stops the execution of a procedure.

In the following lab example, a system signal trigger logs the name of each user who connects to the ObjectServer to a file.

Lab: 9) Write log from trigger

Using nco_sql, create a new Trigger Group. Add the following trigger to the new group, that will write to the log file you created earlier each time a user logs on:

```
create trigger logconnections
group mygroup
priority 1
on signal connect
begin
write into myfile values ('User ', %signal.user_name, ' has logged on.');
end;
```

Log into an event list and check that an appropriate entry has been written to your file.

The following example shows the effect of the FOR EACH ROW loop. To increase the severity of all alerts in the `alerts.status` table that have a severity of 3 to a severity of 4:

```
FOR EACH ROW alert_row in alerts.status WHERE alert_row.Severity=3
BEGIN
       SET alert_row.Severity = 4;
END;
```

When this statement is executed, the ObjectServer reads each row of the `alerts.status` table and tests to see if the value of `Severity` is `3`. For each row that matches this condition, the statements within the `BEGIN` and `END` are executed, until all the rows are processed.

3.9.2.3 Using Variables in Triggers

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:
%trigger.attribute_name
The % symbol indicates that you are referencing an implicit variable. The trigger keyword references the current trigger. Table 36 lists the read-only attributes available in the WHEN and action sections of a trigger.
Implicit Trigger Variables

Trigger Attribute	Description
previous_condition	Value of condition on last execution.
previous_rowcount	Number of rows returned by the EVALUATE clause the last time the trigger was raised.
num_positive_rowcount	Number of consecutive fires with >0 matches in EVALUATE clause.
num_zero_rowcount	Number of consecutive fires with zero matches in EVALUATE clause.

Example
To reference the previous trigger rowcount, use the syntax:
%trigger.previous_rowcount
Note: In a database trigger, the only valid trigger variable is %trigger.
previous_condition. All other trigger variables are used in an EVALUATE clause, which is not supported for database triggers.

3.9.2.4 ALTER TRIGGER command

Use the ALTER TRIGGER command to change the settings of an existing trigger.
Syntax

```
ALTER TRIGGER trigger_name
SET PRIORITY integer
SET ENABLED { TRUE | FALSE }
SET GROUP trigger_group_name
```

```
SET DEBUG { TRUE | FALSE };
```

Use SET PRIORITY to change the priority of a trigger to a value between 1 and 20. The lower the number, the higher the priority. Use SET ENABLED to activate or deactivate a trigger. If a trigger is ENABLED, it fires when the associated signal is raised. If a trigger is not ENABLED, it does not fire when the associated signal is raised.

Use SET GROUP to change the trigger group of the trigger to trigger_group_name.

Use SET DEBUG to turn debugging on or off for the trigger. If the DEBUG attribute is set to TRUE, debugging information is sent to the ObjectServer message log.
You can change more than one setting in a single ALTER TRIGGER command.

3.9.2.5 DROP TRIGGER command

Use the DROP TRIGGER command to drop an existing trigger.
Syntax

```
DROP TRIGGER trigger_name;
```

3.9.3 Database Triggers

You can create database triggers that fire when a modification or attempted modification to an ObjectServer table occurs (or when a modification or attempted modification to a view affects a base table).
You can create a trigger to fire if one 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 primary key already exists. You can use a reinsert to deduplicate rows in the ObjectServer.

Note: Deduplication is now controlled as a database trigger. You can create your own deduplication trigger to cause a different action to occur. An example simple deduplication trigger is shown in Section 3.9.9.1.

Note that a database trigger fires every time that matching condition occurs on the specified table. These triggers ideally should be used to make decisions and modify data based on the content of the affected row and/or the new data that caused the trigger to fire. Operations requiring table scans should be managed via temporal triggers managing a batch of data.

3.9.3.1 Creating Database Triggers

The syntax of the create database trigger command is:

Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string' ]
( BEFORE | AFTER )
( INSERT | UPDATE | DELETE | REINSERT )
ON database_name.table_name
FOR EACH ( ROW | STATEMENT )
[ WHEN condition ]
[ DECLARE variable_declaration ]
BEGIN
trigger_statement_list
END;
```

The BEFORE or AFTER timing attribute specifies whether the trigger is executed before or after the database modification that caused the trigger to fire occurs. For example, you can create a

BEFORE trigger that evaluates the user's name before a row in the alerts.status is deleted. In the trigger, you can detect whether the user is allowed to delete from the alerts.status table, and if not, prevent the database modification from taking place. With an AFTER trigger, the database modification always takes place.

The database_name.table_name is the name of the database and table affected by the trigger action.

The level at which a database trigger fires is one of the following:

- FOR EACH ROW [WHEN condition] (known as a row-level trigger)
- FOR EACH STATEMENT [WHEN condition] (known as a statement-level trigger).

Row-level triggers fire once for each row returned as a result of the database modification.

Statement-level triggers fire once for each database modification.

Note: Only row-level triggers can be defined to fire on insert and reinsert events.

For example, a database signal is raised as a result of the following SQL statement:

```
DELETE FROM alerts.status WHERE Severity = 5;
```

When this statement is executed, the ObjectServer deletes all the rows in the alerts.status table with a severity of 5.

Assume a trigger is set to fire as a result of this database trigger, and there are 20 rows in the table with this severity. If the level attribute is set to FOR EACH ROW, the trigger is raised 20 times, once for each row deleted. If the level is set to FOR EACH STATEMENT, the trigger is raised once.

Note: BEFORE statement-level triggers always fire before BEFORE row-level triggers, and AFTER statement-level triggers always fire after AFTER row-level triggers, regardless of trigger priority.

If you include a WHEN clause, the trigger action is executed only if the condition is TRUE. The WHEN condition is written in the format of a WHERE clause that is applied to the row that caused the trigger to fire. Note that the word "where" must not be included.

```
Severity = 5;          // valid WHEN clause for an alerts.status trigger
WHERE Severity = 5     // INVALID
```

3.9.3.2 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 an event is raised; the NEW variable refers to a column affected by the event, after the event 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 event raised. For example, if the ObjectServer deletes a row, there is no NEW row to read or modify. The next table shows when the NEW and OLD variables are available depending on the database operation (event) that is raised.

Availability of Special Row Variables

Operation	TimingMode	Is the NEW Variable Available?	Is the NEW Variable Modifiable?	Is the OLD Variable Available?	Is the OLD Variable Modifiable?
INSERT	BEFORE	Y	Y	N	N
INSERT	AFTER	Y	N	N	N
UPDATE	BEFORE	Y	Y	Y	N
UPDATE	AFTER	Y	N	N	N
DELETE	BEFORE	N	N	Y	N
DELETE	AFTER	N	N	Y	N
REINSERT	BEFORE	Y	N	Y	Y
REINSERT	AFTER	Y	N	N	N

The following database trigger uses the NEW variable to update the StateChange column when a row in the alerts.status table is modified to timestamp the change.

Example

```
create trigger SetStateChange
group default_triggers
priority 1
before update on alerts.status
for each row
begin
set new.StateChange = getdate();
end;
```

3.9.4 Temporal Triggers

Temporal triggers fire at a specified frequency with optional starting and ending times.

The syntax of the create temporal trigger command is:

Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string'
EVERY integer { HOURS | MINUTES | SECONDS }
[ EVALUATE SELECT_cmd BIND AS variable_name ]
[ WHEN condition ]
[ DECLARE variable_declaration ]
begin
trigger_action;
end;
```

Within a temporal trigger, you must specify how often the trigger will fire in seconds (the default unit of time), minutes, or hours.

The optional EVALUATE attribute enables you to build a temporary result set from a single SELECT statement to be processed in the trigger_action. The SELECT statement cannot contain an ORDER BY clause.

The EVALUATE clause allows you to construct similar behaviour to a v3 trigger and action pair. EVALUATE and BIND allows you to construct a static array of results in the same way as the v3 trigger select statement. The array may then be processed in the trigger_action section. In v7 it is more usual to operate on a dynamic result set using sql, or FOR EACH ROW statements directly against the required table in the trigger_action. Using the dynamic method allows the v7 trigger to work against multiple tables within the same trigger.

Unlike v3, the nature of v7 triggers allows actions to be taking when a select returns zero rows. This effect can be obtained, for example, by retrieving metric counts that are tested in subsequent logic, or by setting a variable to TRUE when a FOR EACH ROW statement returns a positive result, and then basing subsequent actions on the variable not being TRUE.

The following temporal trigger deletes all clear rows (Severity = 0) from the alerts.status table that have not been modified within 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;
```

3.9.5 Signals and Signal Triggers

Signal triggers fire when a predefined system signal is raised or when a user signal is raised using the RAISE SIGNAL command.

A signal is an occurrence in the ObjectServer that can be detected and acted upon. Signals comprise part of the automation subsystem and can have triggers attached to them, so that the ObjectServer can automatically respond when a signal is raised.

3.9.5.1 System Signals

System signals are raised spontaneously by the ObjectServer when it detects changes to the system. You do not have to do anything to create or configure them. You can attach triggers to them to create automatic responses to incidents in the ObjectServer.
Examples of system signals include:

- system startup
- system shutdown
- client connect
- client disconnect
- backup success or failure
- connection failure

When a system signal is raised, attributes that identify the cause of the signal are attached to the signal. These attributes are passed as implicit variables into the associated signal trigger, and can be accessed within triggers.

System signals are predefined. When a system signals is raised, attributes that identify the cause of the signals are set. These attributes can be accessed from the associated system signal triggers. The following table describes some system signals that can be raised by the ObjectServer and the available attributes.

Signal	Attributes	Data Type	Description
Startup	server	string	Indicates the name of the ObjectServer that started.
	node	string	Indicates the machine on which the ObjectServer started.
	at	UTC	Indicates the time at which the ObjectServer started.
Shutdown	server	string	Indicates the name of the ObjectServer that shut down.
	node	string	Indicates the machine on which the ObjectServer shut down.
	at	UTC	Indicates the time at which the ObjectServer shut down.
Connect	process	string	Indicates the type of client process connected to the ObjectServer.

Signal	Attributes	Data Type	Description
	description	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.
	username	string	Indicates the name of the user connected to the ObjectServer.
	node	string	Indicates the name of the client machine connected to the ObjectServer.
	at	UTC	Indicates the time at which the client connected.
Disconnect	process	string	Indicates the type of process that disconnected from the ObjectServer.
	description	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.
	username	string	Indicates the name of the user that disconnected from the ObjectServer.
	node	string	Indicates the name of the client machine that disconnected from the ObjectServer.
	at	UTC	Indicates the time at which the client disconnected.
backup_failed	error	string	Indicates a reason that the backup attempt failed.
	at	UTC	Indicates the time at which the backup attempt occurred.
	path_prefix	string	Indicates the directory to which the backup attempted to write.
	elapsed_time	real	Indicates the amount of time the backup was running before it failed.
	node	string	Indicates the name of the machine from which the backup was run.
backup_succeeded	at	UTC	Indicates the time at which the backup occurred.
	path_prefix	string	Indicates the directory to which the backup was written.
	elapsed_time	real	Indicates the amount of time the backup took to complete.
	node	string	Indicates the name of the machine from which the backup was run.
license_lost	server	string	Indicates the name of the ObjectServer on which a license was lost.
	node	string	Indicates the machine on which a license was lost.
	at	UTC	Indicates the time at which a license was lost.
login_failed	process	string	Indicates the name of the process that could not connect because the login was denied.
	username	string	Indicates the name of the user that failed to connect because login was denied.
	node	string	Indicates the name of the client machine that could not connect because the login was denied.
	at	UTC	Indicates the time at which the client failed to connect because the login was denied.
security_timeout	process	string	Indicates the name of the process that failed to connect because login credentials could not be validated.
	username	string	Indicates the name of the user that failed to connect because login credentials could not be validated.

Signal	Attributes	Data Type	Description
	node	string	Indicates the name of the client machine that failed to connect because login credentials could not be validated.
	at	UTC	Indicates the time at which the client failed to connect because login credentials could not be validated.
create_object	objecttype	string	Indicates the object type, which is one of the following: CREATE DATABASE CREATE TABLE CREATE TRIGGER GROUP CREATE TRIGGER CREATE PROCEDURE CREATE RESTRICTION FILTER CREATE USER SIGNAL CREATE FILE CREATE USER CREATE GROUP CREATE ROLE
	parentname	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.
	name	string	Indicates the name of the object. For example, the value for the alerts.status table is status.
	username	string	Indicates the name of the user that executed the command.
	server	string	Indicates the name of the ObjectServer to which the object was added.
	node	string	Indicates the machine from which the request was made.
	at	UTC	Indicates the time at which the object was added.
alter_object	objecttype	string	Indicates the object type, which is one of the following:
			ALTER TABLE
			ALTER TRIGGER GROUP
			ALTER TRIGGER
			ALTER FILE
			ALTER USER
			ALTER PROCEDURE
			ALTER USER SIGNAL
			ALTER RESTRICTION FILTER
			ALTER GROUP
			ALTER ROLE
	Parentname	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.
	name	string	Indicates the name of the object. For example, the value for the alerts.status table is status.
	username	string	Indicates the name of the user that executed the command.
	server	string	Indicates the name of the ObjectServer in which the object was altered.
	node	string	Indicates the machine from which the request was made.
	at	UTC	Indicates the time at which the object was altered.

Signal	Attributes	Data Type	Description
drop_object	objecttype	string	Indicates the object type, which is one of the following:
			DROP TABLE
			DROP DATABASE
			DROP TRIGGER GROUP
			DROP TRIGGER
			DROP PROCEDURE
			DROP RESTRICTION FILTER
			DROP USER SIGNAL
			DROP FILE
			DROP USER
			DROP GROUP
			DROP ROLE
	Parentname	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.
	name	string	Indicates the name of the object. For example, the value for the alerts.status table is status.
	username	string	Indicates the name of the user that executed the command.
	server	string	Indicates the name of the ObjectServer in which the object was altered.
	node	string	Indicates the machine from which the request was made.
	at	UTC	Indicates the time at which the object was altered.

3.9.6 User Signals

User signals are explicitly created, raised, and dropped. Use the CREATE SIGNAL command to create a user signal. When you create a signal, you define a list of data typed attributes, as follows:

Syntax

```
CREATE [ OR REPLACE ] SIGNAL signal_name
[ (signal_attribute_name data type,...) ]
[ COMMENT 'comment_string' ]
```

The signal name must be unique within the ObjectServer. You cannot create a user 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.

3.9.6.1 Raising a User Signal

To raise a user signal, use the RAISE SIGNAL command, as follows:

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.

3.9.6.2 Dropping a User Signal

To drop a user signal, use the DROP SIGNAL command, as follows:

Syntax

```
DROP SIGNAL signal_name;
```

You cannot drop a signal if a trigger references it.

3.9.6.3 Create Signal Trigger Syntax Definition

The syntax of the create signal trigger command is:

Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
GROUP group_name
[ DEBUG { TRUE | FALSE } ]
[ ENABLED { TRUE | FALSE } ]
PRIORITY integer
[ COMMENT 'comment_string' ]
ON SIGNAL { system_signal_name | user_signal_name }
[ EVALUATE SELECT_cmd BIND AS variable_name ]
[ WHEN condition ]
[ DECLARE variable_declaration ]
trigger_action;
```

The ON SIGNAL attribute can be the name of a system or user signal that fires the trigger.

The optional EVALUATE attribute enables you to build a temporary result set from a single SELECT statement to be processed in the trigger_action. The SELECT statement cannot contain an ORDER BY clause.

When a system or user signal is raised, attributes that identify the cause of the signal are attached to the signal. These attributes are passed as implicit variables into the associated signal trigger, and are described next.

3.9.6.4 Signal Variables

You can refer to user and system signal variables using the %signal notation in the action section of a signal trigger, as follows:

```
%signal.attribute_name
```

The % symbol indicates that you are referencing an implicit variable. The signal keyword references the signal currently passed to the trigger. The attribute_name is the name of an attribute declared when the signal was created.

For example, to reference the time at which a connection occurred within a system signal trigger, use the syntax:

```
%signal.at
```

Tip: You can query the catalog.primitive_signal_parameters table to view all system signals using the following SQL command. For example:

```
SELECT * FROM catalog.primitive_signal_parameters ORDER BY SignalName,
OrdinalPosition;
```

3.9.7 Using Signals and Triggers in Automations

Signals and triggers are linked in Automations to defined actions to be taken when specific incidents occur

For example, a signal is created called illegal_delete with two character string parameters, user_name and row_summary, by use of 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) or (hourofday() < 9) ) then
raise signal illegal_delete %user.user_name, old.Summary;
cancel;
end if;
end;
```

The following user signal trigger, triggered by the preceding database trigger, executes an external procedure to send mail notification of the attempted delete. [Note that the execute command must be entered on a single line]

```
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;
```

This combination of signals and triggers cancels the delete outside normal hours after raising a signal that in turn emails a warning that the delete attempt has been made.

3.9.8 Controlling Automation Processing Sequence

In v3 the processing sequence of automations that fire concurrently was managed by the lexical sequence of the automation name. In v7, automations may be assigned Priority values. Automations firing concurrently will be executed in Priority order. The firing sequence of automations of the same priority is indeterminate.

The flexibility of the v7 automations sub-system provides several methods to achieve better management of the automation flow.

The capabilities of the procedural sql allows conditional statements to be constructed within a single trigger. For example, in v3.6 the improved GenericClear required four consecutive automations to complete its operation. The same functionality in v7 is achieved within a single trigger removing the need for sequence. (See section 3.9.9.7)

Common functionality may be constructed within stored procedures that may then be executed from several triggers.

Where multiple triggers are required to fire in sequence, then the dependant triggers may be created as Signal triggers using user defined signals. The appropriate signal is then raised within the parent triggers as required.

3.9.9 Default Automations

This section contains examples of some commonly performed automations. To create these and other standard automations, either choose the load automations option (selected by default) during ObjectServer installation or run the `automation.sql` script after installation.

3.9.9.1 Simple Reinsert Deduplication Trigger

The v7 ObjectServer uses triggers to manage deduplication. Deduplication was not configurable in 3.x. It is now configurable in v7.

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.

The following SQL fragment taken from the default deduplication trigger illustrates how an event's "old" values are updated by the incoming "new" values from the probe. This replicates the behavior of the 3.x deduplication functionality.

```
-- increment Tally
set old.Tally = old.Tally + 1;
-- Update Last Occurrence Time with the value from the probe
set old.LastOccurrence = new.LastOccurrence;
-- update system timestamps
set old.StateChange = getdate();
set old.InternalLast = getdate();
-- update Summary with the value from the probe
set old.Summary = new.Summary;
-- update Alert Key
set old.AlertKey = new.AlertKey;
```

The second fragment indicates how conditional logic may be added to modify the behavior of the deduplication. In this example, the old severity is only modified if it is clear and the incoming severity indicates a recurrence of the problem.

```
if (( old.Severity = 0) and (new.Severity > 0))
then
       set old.Severity = new.Severity;
end if;
```

Further modifications may be made to tailor the deduplication processing to meet precise individual requirements.

3.9.9.2 Details Table Deduplication Trigger

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;
```

3.9.9.3 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 Serial not in (select Serial from alerts.status);
end;
```

3.9.9.4 Set a StateChange Column in alerts.status

When a row in the alerts.status table is modified, this database trigger updates the StateChange column to timestamp the change.

```
create or replace trigger state_change
group default_triggers
priority 1
comment 'State change processing for ALERTS.STATUS'
before update on alerts.status
for each row
begin
set new.StateChange = getdate();
end;
```

3.9.9.5 Delete Clears

This temporal trigger deletes all clear rows (Severity = 0) from the alerts.status table that have not been modified within the last two minutes.

```
create or replace trigger DeleteClears
group default_triggers
priority 1
comment 'Delete cleared events over 2 minutes old every 60 seconds'
every 60 seconds
begin
delete from alerts.status where
Severity = 0 and StateChange < (getdate() - 120);
end;
```

3.9.9.6 Email on Critical Alerts

This temporal trigger sends email, by calling an external procedure, if any critical alerts are not acknowledged within 30 minutes.

```
create or replace trigger MailonCritical
group default_triggers
priority 1
comment 'Send email about critical alerts that are
unacknowledged after 30 minutes'
every 10 seconds
evaluate
select Node, Severity, Summary, Identifier from alerts.status where
Severity = 5 and
Grade < 2 and
Acknowledged = 0 and
LastOccurrence <= (getdate() -(60*30))
bind as criticals
begin
for each row critical in criticals
begin
   execute send_email( critical.Node, critical.Severity,
       'Netcool Email', 'root@localhost', critical.Summary,
       'localhost');
   update alerts.status via critical.Identifier set Grade=2;
end;
end;
```

The sendemail external procedure is declared as follows and calls the nco_mail utility. Note the concatenation of the escaped single quotes around the string parameters to allow for the passing of fields with imbedded spaces:

```
create or replace procedure sendemail
(in node character(1), in severity integer, in subject character(1),
in email character(1), in summary character(1), in hostname
character(1))
executable '/opt/netcool/utils/nco_mail' host hostname user 0 group 0
arguments '\''+node+'\'', severity, '\''+subject+'\'', '\''+email+'\'',
'\''+summary+'\'';
```

3.9.9.7 Generic Clear (Up/Down Correlation)

Netcool/OMNIbus v7 provides an updated version of the GenericClear automation. The user is recommended to use this version in place of the original GenericClear, and of that defined in later versions of Netcool/OMNIBus 3.x.

The enhanced SQL in v7 allows the multiple dependent automations that were required in v3.6 to be combined into a single one in v7 for ease of maintenance. Additional string concatenation functionality delivers further optimization of the SQL reducing the number of “where” clause evaluations needed.

This temporal trigger clears (sets Severity to 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).

Like its predecessor the GenericClear requires a workspace that is defined as a virtual table within the ObjectServer. This table is created as part of the default ObjectServer structure with the following columns:

```
Identifier   varchar(255) primary key,
LastOccurrence      date,
AlertKey     varchar(255),
AlertGroup   varchar(64),
Node         varchar(64),
Manager            varchar(64),
Resolved     boolean
```

The single trigger has several components that achieve the functionality that required four successive automations in earlier versions of the database. Note particularly the use of the new string concatenation functionality highlighted in bold:

```
create trigger GenericClear
group my_triggers
priority 1
every 5 seconds
begin

-- Populate a table with Type 1 events corresponding to any uncleared
Type 2 events
for each row problem in alerts.status where
problem.Type = 1 and problem.Severity > 0 and(problem.Node +
problem.AlertKey + problem.AlertGroup + problem.Manager) in( select Node
+ AlertKey + AlertGroup + Manager from alerts.status where Severity > 0
and Type = 2 )
begin
insert into alerts.problem_events values ( problem.Identifier,
problem.LastOccurrence, problem.AlertKey, problem.AlertGroup,
problem.Node, problem.Manager, false );
end;
-- For each resolution event, mark the corresponding problem_events
entry
-- as resolved and clear the resolution
for each row resolution in alerts.status where resolution.Type = 2 and
resolution.Severity > 0
begin
set resolution.Severity = 0;
update alerts.problem_events set Resolved = true where LastOccurrence <
resolution.LastOccurrence and Manager = resolution.Manager and Node =
resolution.Node and AlertKey = resolution.AlertKey and AlertGroup =
resolution.AlertGroup ;
end;


-- Clear the resolved events
for each row problem in alerts.problem_events where problem.Resolved =
true
begin
update alerts.status via problem.Identifier set Severity = 0;
end;
-- Remove all entries from the problems table
delete from alerts.problem_events;
end;
go
```

3.9.9.8 Problem/Resolution Correlation by Deduplication

Some users prefer to use deduplication for problem resolution rather than the Generic Clear. One consequence of this in v3.x is to artificially inflate the value of Tally. The Tally is incremented for problem and resolution events at the point of deduplication. This is unacceptable to many users. The Netcool/OMNIbus v7 deduplication trigger described above may be enhanced to remove these side effects.

The following conditional SQL fragment illustrates how this is achieved with the lines in bold indicating the minimum change required, and is extended to capture and calculate “Resolution Time” “Resolution Tally” and “Time To Resolve” optional fields that could be added to the alerts.status table to contain details of the resolution events.

```
if( (new.Type = 2) and (old.Type = 1) )
```

```
then
        -- The new event is a Resolution matching a Problem
        -- event in the ObjectServer
        set old.ResolutionTime = getdate();
        set old.ResolutionTally = old.ResolutionTally + 1;
        set old.Summary = "Cleared: " + old.Summary;
        set old.Severity = 0;
        set old.TimeToResolve = getdate() - old.FirstOccurrence;
else
        -- Not a matched pair, deduplicate as normal
        set old.Tally = old.Tally + 1;
        ……
        ……
        ……
end if;
```

This technique may be used for any problem/resolution event pairing that has a one-to-one relationship. The performance effect is that the Clear occurs instantly avoiding the need for the SQL searches and updates of the Generic Clear. The batch processing of Generic Clear is then only required to manage resolution and problem events that have a one-to-many relationship.

The Resolution Time in this example could only be calculated in 3.x after the deduplication had taken place. This would be achieved by using a temporal automation to search the database at regular intervals for recently cleared events, then to performing the calculation and update. The new trigger capability removes the need for a full database scan.

Note. Implementation of this functionality requires modification to probe rules files to ensure that the Identifier field is set equal for matching Problem (Type 1) and Resolution (Type 2) events. You may first make the modification to the automation, then modify rules files over time. Field trials have identified significant reductions in processing time required for this solution compared to making use of the GenericClear to handle all Resolution.

Note also that use of this technique does not rely on comparison of LastOccurrence times to match the Problem/Resolution pair. It is simply the sequence of arrival at the ObjectServer that determines the match. This allows management of sub-second alarms provided that the network element or manager delivers the Problem event before the corresponding clear.

4 Netcool Administrator for Omnibus

4.1 Introduction

Netcool/OMNIbus v7 introduces the first phase of Netcool Administrator for OMNIbus providing a new ObjectServer configuration tool. The nco_config tool is java based providing a standard look and feel across disparate platforms. The tool supports the maintenance of the revised structures within the ObjectServer similar to the functionality provided nco_admin, and adds interactive SQL functionality. With nco_config you can:

- Administer and configure an ObjectServer
- Configure security, including roles, group, and users
- Update properties and rules files for Netcool/OMNIbus components

Run $OMNIHOME/bin/nco_config. Select your server name and login as you would for a 3.x system.

Once authenticated Permissions are determined by the User's Group settings in the ObjectServer. For example, a user may only have permission to create and edit tools. If the user uses the nco_config to connect to the ObjectServer and then attempts to do anything other than create and edit tools, permission will be denied.

To add new and configure existing Netcool/OMNIbus components, click the button for the desired component type on the Netcool/Administrator Console button bar.

4.2 Configuring ObjectServer Components

4.2.1 Overview

You can use nco_config to configure the following ObjectServer components:

- Databases
- ObjectServer Properties
- Conversions
- Classes
- Event list alert severity colors
- Prompts
- Column visuals
- Tools
- Procedures
- User signals
- Triggers
- Trigger groups
- Netcool/OMNIbus event list menus
- ObjectServer files
- Restriction filters

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

Lab: 10) ObjectServer configuration

In the nco_config button bar, each group on the left-hand panel represents multiple, related ObjectServer components.
Click the group name to display the components in that group.
Look around the various components to see the structure of the menus and associated toolbars.
Keep the configuration window open.

Some ObjectServer configuration windows enable you to enter SQL commands when creating or editing components (for example, triggers and restriction filters). These windows contain helper buttons that you can use to simplify the creation of SQL commands.

> *Click on the SQL helper buttons in one of the windows. You will see that they are similar to the old nco_config structures.*

A set of variables are available to localize the commands. These are:

- **%display**—The current display running the application.
- **%password**—The password of the user running the application.
- **%server**—The name of the ObjectServer to which the tool is currently connected.
- **%uid**—The user identifier of the user running the application.
- **%username**—The user name of the user running the application.

Note: Once you have opened an ObjectServer for administration the options available are the same set as you would see if launching the new nco_config from the command line as described in Section 4.1.

4.2.2 Configuring Databases

You can use nco_config to manage and create the ObjectServer databases and tables. You can create and drop databases. You can create, drop, and alter database tables.

Note: You are not permitted to make changes to system databases.

To configure databases, on the ObjectServer Configuration window, click the **System** button; then, click the **Databases** button. The Databases, Tables and Columns window contains a list of the databases in the selected ObjectServer.

Lab: 11) Recreate Tables

In the following paragraphs you will study the interactive options that support database maintenance. Use these options to recreate the database and table that you created in Lab: 3.

4.2.2.1 Creating Databases

To create an ObjectServer database:

- Click the **Create Database** button. The New Database window is displayed.
- In the **Name** field, enter the database name.
- Click **OK**. The database is added the ObjectServer and is displayed in the Databases, Tables and Columns window.
- You can now add tables and table columns to the database.

4.2.2.2 Dropping a Database

To drop a database:

- Select the database to drop.
- Click the **Drop Database** button. The database is removed from the ObjectServer.

4.2.2.3 Creating Database Tables

To create a database table:

Select the database in which you are creating the table.

- Click the **Create Table** button. The New Table Details window is displayed.
- In the **Name** field, enter the table name.
- Select the required table **Type**.

Use the column buttons to create, edit, and drop columns in this table. Use the arrow buttons to change the order of the selected column in the table.

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

4.2.2.4 Dropping a Database Table

To drop a database table:

- Select the database containing the table to drop.
- Select the table to drop.
- Click the **Drop Table** button. The database table is removed from the database.

4.2.2.5 Creating and Editing Table Columns

To create or edit a table column:

- Select the table in which you are creating or editing the column.
- Click the Column Definitions tab.
- If you are editing a column, select the column to edit.
- Click the **Add Column** or **Edit Column** button. The Edit Column Details window is displayed.
- Complete or edit the following window items:
- Column Name. If you are editing the column, you cannot change the column name.
- Data Type

4.2.2.6 Dropping Table Columns

To drop a table column:

- On the Databases, Tables and Columns window, click the **Column Definitions** tab.
- Select the column to drop.
- Click the **Drop Column** button. The column is removed from the table.

4.2.3 Viewing and Changing ObjectServer Properties

You can view and change ObjectServer properties while the ObjectServer is running.
You cannot add ObjectServer properties; you can only edit existing ones.

To view ObjectServer properties, on the ObjectServer Configuration window click the **System** button; then, click the **Properties** button.
The Properties window contains a list of the current properties and settings for the selected ObjectServer.

Note: White rows contain view-only properties.

Lab: 12) Change ObjectServer properties

*On the ObjectServer Configuration window, click the **System** button.*
*Click the **Properties** button to display the Properties window*
*In the **Value** column, select the property value to edit.*
Change the debug level and profiling options
Tail the ObjectServer log to note the effect

4.2.4 Maintaining Visual Options

Visual Options includes the maintenance of Conversions, Classes, Column Visuals and Colours.

To maintain these options on the ObjectServer Configuration window, click the **Visual** button, then select the required option from the menu.

Lab: 13) Configure Visuals

Modify a set of Visual options including colours. Resynch your event list and test the effect of your changes.

4.2.4.1 Creating and Editing Conversions

To configure conversions, then, click the **Conversions** button. The Conversions window contains a list of the existing conversions for each column.
Click the plus symbol (+) next to a column to see existing conversions.

Conversions may be added, created and deleted using the standard windows menus and options.

4.2.4.2 Creating and Editing Classes

To configure classes, click the **Classes** button. The Classes window contains a list of the classes set up for the selected ObjectServer.

Classes may be added, created and deleted using the standard windows menus and options.

4.2.4.3 Creating and Editing Column Visuals

To configure column visuals, on the ObjectServer Configuration window, click the **Visual** button; then, click the **Column Visuals** button. The Column Visual Details window contains a list of the column visuals set up for the selected ObjectServer.

Column Visuals may be added, created and deleted using the standard windows menus and options.

4.2.4.4 Configuring Event List Alert Severity Colors

You can view, create, and modify the alert severity colors used in Netcool/OMNIbus event lists.

To configure alert severity colors, on the ObjectServer Configuration window click the **Visual** button; then, click the **Colors** button. The Color Details window contains a list of the colors set up for the selected ObjectServer.

Click the **Add** button to add a new severity color, or select the color to edit and click the **Edit** button. The Colors Detail window is displayed.

Complete or edit the Colors Detail window items:

- Severity: If you are creating a new color, enter the alert severity value for the color to create or edit.
- Conversion: Displays the conversion for this alert severity. For example, the default conversion for a severity of **4** is **major**. You cannot use the Colors Detail window to edit conversions.
- Color selection button Click the color selection button to select the severity color for both acknowledged and unacknowledged alerts. You can choose the color using its swatch, HSB, and RGB values.

4.2.5 Configuring Event List Menus

You can use the ObjectServer Configuration window to customize Netcool/OMNIbus desktop menus. You can:

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

To configure event list menus, on the ObjectServer Configuration window, click the **Menus** button; then, click the **Menus** button in the drop-down list. The Menus window contains a list of the menus set up for the selected ObjectServer.

Click the plus symbol (+) next to a menu to see existing menu items and tools. The order in which the menu items appear on the Menu Details window is the order in which they appear in Netcool/OMNIbus.

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

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

- Select the menu to which you are adding the menu item.
- Click the **Add** button. The New Menu Item window is displayed.
- Click the **Type** drop-down arrow and select one of the following:
  - Tool
  - Separator
  - Sub-menu
- If you are adding a tool, click the **Tool** drop-down arrow to select a tool. You can also click the **New** button to create a new tool, or click the **Edit** button to edit an existing tool.
- If you are adding a tool or a sub-menu, in the **Title** field, enter the name as it will appear in the menu.
- Click **OK**. The new menu item appears on the Menus window.

The menu item will appear in Netcool/OMNIbus the next time the Netcool/OMNIbus desktop is started.

4.2.5.2 Renaming Menu Items

You can rename tools and sub-menus on Netcool/OMNIbus event list menus:

- Select the menu item to rename.
- Click the **Edit** button. The Edit Menu Item window is displayed.
- In the **Title** field, enter the name as it will appear in the menu.
- Click **OK**. The menu item appears on the Menus window.

The menu item will appear in Netcool/OMNIbus the next time the Netcool/OMNIbus desktop is started.

4.2.5.3 Changing the Order of Menu Items

To change the order of Netcool/OMNIbus desktop menu items:

- Select the menu item to reorder.
- Use the buttons to adjust the position of the menu item.

The menu item will be displayed in the selected position the next time the Netcool/OMNIbus desktop is started.

4.2.5.4 Removing a Menu Item

To remove a menu item from the Netcool/OMNIbus desktop:

- Select the menu item to remove.
- Click the **Delete** button.

The menu item will be removed from Netcool/OMNIbus the next time the Netcool/OMNIbus desktop is started.

4.2.5.5 Testing Menus

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

- Select the required menu item and Click the **Test Menu** button. The menus and menu items appear as they will in the Netcool/OMNIbus desktop.

4.2.6 Configuring Tools

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

To configure tools, on the ObjectServer Configuration window, click the **Menus** button; then, click the **Tools** button. The Tool Details window contains a list of the tools set up for the selected ObjectServer.

4.2.6.1 Creating and Editing a Tool

To create or edit a tool:

- Click the **Add** button, or select the tool to edit and click the **Edit** button. The New Tool or Edit Tool window is displayed.
- In the **Name** field, enter a unique name for this tool.
- Select the **Enabled** check box to enable operators to use this tool.
- Complete or edit the remaining window tabs. These are described in the following sections.
- Click **OK**. The tool is saved and displayed in the Tool Details window.

4.2.6.2 Deleting a Tool

To delete a tool:

- Select the tool to delete, Click the **Delete** button. The tool is deleted.

4.2.7 Configuring Prompts

To configure prompts, on the ObjectServer Configuration window , click the **Menus** button; then, click the **Prompts** button. The Prompts Details window contains a list of the prompts set up for the selected ObjectServer.

4.2.7.1 Creating and Editing a Prompt

To create or edit a prompt:

- Click the **Add** button, or select the prompt to edit and click the **Edit** button. The New Prompt or Edit Prompt window is displayed.
- Complete the following fields:
    - Enter a unique name for the prompt.
    - Enter the prompt text to appear when the tool requests information from the user.
    - Select the prompt Type.
        - String
        - Integer
        - Float
        - Time
        - Fixed Choice
        - Lookup
        - Password
        - Dynamic Choice
    - The remaining fields depend on the selected prompt type.
    - The String prompt creates a prompt window that accepts one or more characters.
    - The Integer prompt creates a prompt window that accepts an integer value..
    - The Float prompt creates a prompt window that accepts a floating point number, which can contain a decimal point.
    - The Time prompt creates a prompt window that accepts a time. The default display is the current time.
    - A single Fixed Choice prompt in a tool creates a popup menu. The menu is populated by the values that you enter into the **Options** list.
      To create a new option, click the **New** button. You can also edit and delete the existing options.
    - A single Lookup prompt in a tool creates a popup menu. The menu is populated by the values in a file.
      To complete the prompt details do one of the following:
        - In the File field, enter the path and name of the file.
        - Click the **Browse** button to open a standard file selection window. Select the file.

- The Password prompt creates a prompt window that accepts one or more characters. The password characters appear as asterisks as you type.
- A single dynamic choice prompt in a tool creates a popup menu. The menu is populated by the results of a database query.
  - Select a database from the drop-down list.
  - Select a table in the selected database from the drop-down list.
  - Show defines the table field used to populate the prompt menu. Select a field name from the drop-down list.

- Click **OK**. The prompt is saved and displayed in the Prompt Details window.

4.2.7.2 Deleting a Prompt

To delete a prompt:

- Select the prompt to delete and click the **Delete** button. The prompt is deleted.

4.2.8 Roles, Groups and Users

The structure of Netcool/OMNIbus security is described in Section 3.7. The following paragraphs describe the Administrator functions that support the maintenance of the entities controlling access and permissions within the ObjectServer.

To access the Security related functions, in the Netcool/Administrator button bar, click the **Users** button.

4.2.8.1 Roles

4.2.8.1.1 Creating and Editing Roles

To create or edit a role:

- Click the **Roles** button. The Role Details window is displayed.
- Click the **Add** button, or select the role to edit and click the **Edit** button. The Edit Role Details window is displayed.
- In the **Role Name** field, enter the name for this role. If you are editing the role, you cannot change the role name. Role names are restricted to 64 characters and can include spaces.
- Complete or edit the **Identity** and **Permissions** tabs, as described below.
- Click **OK** to save the role information.

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

To assign permissions to a role:

- Click the **Permissions** tab.
- Click the Add button. The Select a new object to grant permissions to window is displayed.
- Select the **Object Type** for which you want to grant permissions.
- Click **OK**. The Role Details window is displayed with the selected permissions.
- Repeat steps these steps to add additional permissions.
- Click **OK** to save the permissions for this role.

4.2.8.1.2 Deleting Roles

To delete a role:

- Click the **Roles** button. The Role Details window is displayed.
- Select the role to delete and click the **Delete** button. The role is deleted.

4.2.8.2 Configuring Groups

To maintain Groups in the Netcool/Administrator button bar, click the **Users** button.

4.2.8.2.1 Creating and Editing Groups

To create or edit a group:

- Click the Groups button. The Group Details window is displayed.
- Click the Add button, or select the group to edit and click the Edit button. The Edit Group Details window is displayed.
- Complete or edit the following window items:
    - **Group Name:** Group names are restricted to 64 characters and can include spaces. If you are editing this group, you cannot change the group name.
    - **Group ID:** Select a unique ID for this group. If you are editing this group, you cannot change the group ID.
    - **Description**: A text description for this group.
    - **Roles:** Select the group and click the **Edit** button. The Edit Group Details window is displayed. The **Src List** column lists the available roles to which you can assign this group. The **Dst List** column lists the roles to which this group is currently assigned. Use the arrow buttons to assign roles to the group.
- Click **OK**. The group information is saved.

4.2.8.2.2 Deleting Groups

To delete a group, click the **Groups** button. The Group Details window is displayed. Select the group to delete. Click the **Delete** button. The group is deleted.

4.2.8.3 Configuring Users

You can create and modify Netcool/OMNIbus users. You can also assign users to groups for organizational purposes. In the Netcool/Administrator button bar, click the **Users** button. The User Details window is displayed.

4.2.8.3.1 Creating and Editing Users

To create or edit a user:

- Click the Add button, or select the user to edit and click the Edit button. The Edit User Details window is displayed.
- Complete or edit the following window items:
    - **User Name:** User names are restricted to 64 characters and can include spaces. If you are editing this user, you cannot change the user name.
    - **ID**: Select a unique ID for this user. This should be set to match the UNIX UID where possible. If you are editing this user, you cannot change the user ID.
    - **Full Name:** The user's full name.
    - **Create Conversion**: Indicates whether to create a conversion for this user. A conversion enables the user's name to appear in Netcool/OMNIbus event lists instead of the user ID.
    - **Enabled**: Indicates whether this user is enabled.
    - **Use PAM**: Indicates whether Pluggable Authentication Modules (PAM) are used as the authentication mode to verify user credentials.
    - **Password/Verify**: Enter the password in the text box.
    - **Restriction Filter**: You can optionally select a restriction filter to apply to this user
    - Click **OK**. The user information is saved.

4.2.8.3.2 Adding Users to Groups

To add a user to a group or multiple groups:

- Select the user and click the Edit button. The Edit User Details window is displayed.
- Click the Groups tab. The Unassigned column lists the available groups to which you can add this user. The Assigned Groups lists the groups to which this user is currently assigned. Use the arrow keys to assign or remove the User from selected Groups.
- Click **OK**. The user group information is saved.

4.2.8.3.3 Deleting Users

To delete a user select the required user. Click the **Delete** button. The user is deleted.

4.2.9 Triggers and Groups

The Structure of Triggers and groups is defined in Section 3.9.

4.2.9.1 Trigger Groups

4.2.9.1.1 Configuring Trigger Groups

Trigger Groups are maintained on the ObjectServer Configuration window by clicking the **Automation,** then the **Trigger Groups** buttons. The Trigger Group Details window is displayed.

Trigger Groups are modified via the **Add, Edit**, and **Delete** buttons. The Group name must be unique, and the Enabled option ticked to activate the Group.

Lab: 14) Create Trigger

In the next sections create a set of signals and triggers based on the example in Section 3.9.7. If you do not have email setup on your system, write an error message to your log file. Modify the time band to test the trigger relative to the time setting on your system.

4.2.9.2 Triggers

Triggers are maintained on the ObjectServer Configuration window by clicking the **Automation,** then the **Trigger** buttons. The Trigger Details window is displayed.

- To create or edit a trigger click the toolbar button for the trigger type to create, or select the trigger to edit and click the Edit button. If you are creating a new trigger, in the Name field, enter a unique trigger name. If you are editing a trigger, you cannot edit this field.
- Select the trigger group to which this trigger belongs.
- Complete or edit the window for the selected trigger type. Instructions for creating each trigger type are included in the following sections.
- When you have completed the window, click **OK**. The trigger is saved and displayed in the Triggers Detail window.

4.2.9.2.1 Creating and Editing Database Triggers

The Database Trigger window is comprised of the **Settings** and **Definition** tabs.

- On: Select the ObjectServer database and table on which this trigger will fire.
- Run: Indicate whether the trigger action is executed:
  - *Before* the database modification that caused the trigger to fire occurs (**Pre Database Action**)
  - *After* the database modification that caused the trigger to fire occurs (**Post Database Action**)
  - Also, select the database modification:
    - Insert
    - Reinsert
    - Update
    - Delete
  - Priority Determines the order in which the ObjectServer fires triggers when more than one trigger is attached to an event. Use the slider to select 1-20, with 1 being the highest priority.
- **Apply to:**
  - Select **Row** to set the trigger to execute its action once for every alert that matches the trigger conditions. This mode allows for a many-to-many mapping between particular alerts and actions taken.
  - Select **Statement** to set the trigger to execute its actions once regardless of the number of matched rows in the temporary table. This allows only a one-to-one mapping between the overall state and the action taken.
  - **State:** Select **Enabled** to enable (activate) this trigger. A disabled trigger will not fire when the associated database modification occurs.

- Select **Debug** to send debugging information to the ObjectServer message log.

4.2.9.2.2 Creating and Editing Signal Triggers

Signal triggers fire when a system or user signal is raised. *System* signals are raised spontaneously by the ObjectServer when it detects changes to the system; you do not have to do anything to create or configure them. *User* signals are explicitly created, raised, and dropped.

The Signal Trigger window is comprised of the **Settings** and **Definition** tabs.
Set the following details for the trigger

- Signal: Select the type of signal that will cause this trigger to fire.
- Priority: Determines the order in which the ObjectServer fires triggers when more than one trigger is attached to an event.
- State: Select Enabled to enable (activate) this trigger.
- Select Debug to send debugging information to the ObjectServer message log.
- When: Optionally allows you to test for a particular condition after the trigger has fired but before the action is executed. If the condition is not met, the action is not executed.
- Evaluate: You can optionally build a temporary result set from a single SELECT statement to be processed during the trigger action (defined on the Actions tab). Use the format:
- EVALUATE <SELECT_cmd> BIND AS <variable_name>
- Bind As <array name>
- Actions: Enter the trigger action, which is a set of SQL statements that are executed when the trigger is fired. It has the following syntax:

```
DECLARE [ <variable_declaration> ]
BEGIN
<trigger_statement_list>
END;
```

  You can optionally declare any variables used within the trigger. Trigger variables are static; they maintain their value from previous executions of the trigger. Use the `%TRIGGER` notation to specify trigger variables: %TRIGGER.<attribute_name>

4.2.9.2.3 Creating and Editing Temporal Triggers

Temporal triggers fires repeatedly based on a specified frequency. The Temporal Trigger window is comprised of the **Settings** and **Definition** tabs.

- **Declarations**: tab You can optionally declare any variables used within the trigger. Trigger variables are static; they maintain their value from previous executions of the trigger. Use the `%TRIGGER` notation to specify trigger variables: %TRIGGER.<attribute_name>
- **Comment** tab An optional text comment for this trigger.
- **Every**: Select how frequent the trigger will fire. Enter the numeric value and select **HOURS**, **MINUTES**, or SECONDS.
- **Priority: Determines the order in which the ObjectServer fires triggers when more than one trigger is attached to an event. Use the slider to select 1-20, with 1 being the highest priority.**
- **State: Select Enabled to enable (activate) this trigger.**
- **Select Debug to send debugging information to the ObjectServer message log.**
- **When: Optionally allows you to test for a particular condition after the trigger has fired but before the action is executed. If the condition is not met, the action is not executed.**
- **Evaluate**: You can optionally build a temporary result set from a single `SELECT` statement to be processed during the trigger action (defined on the **Actions** tab). Use the format:

```
EVALUATE <SELECT_cmd> BIND AS <variable_name>
Bind As <array name>
```

- **Actions**: Enter the trigger action, which is a set of SQL statements that are executed when the trigger is fired. It has the following syntax:

```
DECLARE [ <variable_declaration> ]
BEGIN
<trigger_statement_list>
```

```
END;
```

4.2.9.3 Deleting a Trigger

To delete a trigger, Select the trigger to delete. Click the **Delete** button. The trigger is deleted.

4.2.10 Configuring User Signals

A signal is an occurrence within the ObjectServer that can be detected and acted upon.
To configure user signals, on the ObjectServer Configuration window, click the **Automation** button; then click the **User Defined Signals** button. The User Signals window contains a list of the user signals set up for the selected ObjectServer.

4.2.10.1 Creating and Editing User Signals

To create or edit user signals:

- **Click the Add button, or select the user signal to edit and click the Edit button. The Edit User Signal Details window is displayed.**
- **Complete** the following window items:
  - **Signal Name**: Enter a unique name for this signal.
  - For each required parameter:
    - Name Enter the parameter name.
    - Data Type: Select from the valid data types for a user signal:
    - Length For Char data types only, enter the parameter length.
    - New: button After completing the **Name**, **Data Type**, and **Length** fields, click this button to add the parameter to the parameter list.
- **Parameter list:** Displays the parameters that comprise this user signal. The order in which the parameters appear must match the order that they appear in the trigger `RAISE SIGNAL` command.
  - **Up**/**Down** buttons: Use to change the parameter order.
  - **Delete:** Deletes the selected parameter.
- **Comment**: Enter a text comment for this parameters.
- Click **OK**. The user signal is saved and displayed in the User Signals window.

4.2.10.2 Deleting a User Signal

To delete a user signal, select the user signal to delete and click the **Delete** button. The user signal is deleted.

4.2.11 Configuring Procedures

There are two types of procedures:

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

The structures that make up a procedure are defined in section 3.8.

To configure procedures, on the ObjectServer Configuration window, click the Automation button; then, click the **Procedures** button. The Procedure Details window contains a list of the procedures set up for the selected ObjectServer.

- **Click either the Add SQL Procedure or Add External Procedure toolbar button, or select the procedure to edit and click the Edit button.**
- **Complete the window for the selected procedure type. Instructions for creating each procedure type are included in the following sections.**
- **click OK**. The procedure is saved and displayed in the Procedure Details window.

4.2.11.1 Creating and Editing SQL Procedures

SQL procedures have the following major components:

- **Procedure: Name Enter a unique name for the procedure.**
- **Parameters List: Enter the parameters to pass into and out of the procedure. Each procedure parameter has a mode, which can be IN, OUT, or IN OUT.**

- **Declarations: You can optionally declare local variables for use within a procedure.**
- **Actions:** Enter the SQL commands for this procedure.

4.2.11.2 Creating and Editing External Procedures

You can create external procedures to run an executable file on a local or remote system. The required parameters are:

- **Procedure Name: Enter a unique name for the procedure.**
- **Executable: Enter the full path for the command to run.**
- **Host: Enter the host on which to execute the procedure**
- **User: Enter the user ID under which to run the executable.**
- **Group: Enter the user role under which to run the executable.**

4.2.11.3 Deleting Procedures

To delete a procedure: Select the procedure to delete. Click the **Delete** button. The procedure is deleted.

4.2.12 Configuring ObjectServer Files

ObjectServer files are user-defined storage objects for log or report data described in detail in section 3.4.5.

To configure ObjectServer files, on the ObjectServer Configuration window click the **System** button; then, click the **Log Files** button. The Log Files window contains a list of the ObjectServer files set up for the selected ObjectServer.

4.2.12.1 Creating and Editing ObjectServer Files

To create or edit ObjectServer files:

- **Click the Add button, or select the ObjectServer file to edit and click the Edit button. The Edit File Details window is displayed.**
- **Complete** the following window items:
  - **File Name:** Enter a unique name for this ObjectServer file.
  - **File Path:** Enter the complete path to the directory in which to write the ObjectServer files.
  - **Maximum Files:** Select the maximum number of ObjectServer files to create.
  - **Maximum Size:** Select the maximum ObjectServer file size. Additionally, select one of the following from the drop-down list:
    - BYTE
    - KBTE (kilobytes)
    - MBYTE (megabytes)
    - GBYTE (gigabytes)
  - **Enabled**: Select this check box to enable the creation of this ObjectServer file set. If you do not select this check box, the ObjectServer will not create this ObjectServer file set.
- Click **OK**. The ObjectServer file is saved and displayed in the Log Files window.

4.2.13 Truncating ObjectServer Files

Truncating an ObjectServer file clears any information that has been written to the physical file, but does not delete the file. In situations where there is more than one physical file in a set, only the ObjectServer file that is currently being written to on the file system is truncated.
To truncate an ObjectServer file:

- Select the ObjectServer file to truncate and click the Edit button. The Edit File window is displayed.
- Click the **Truncate File** button. The contents of the ObjectServer file are deleted.

4.2.13.1 Deleting ObjectServer Files

To delete an ObjectServer file: select the ObjectServer file to delete. Click the **Delete** button. The ObjectServer file is deleted. The ObjectServer will no longer write information to this file.

4.2.14 Configuring Restriction Filters

Restriction filters are described in section 3.4.4.

To configure restriction filters, on the ObjectServer Configuration window , click the **Users** button; then, click the **Restriction Filters** button. The Restriction Filter Details window contains a list of the restriction filters set up for the selected ObjectServer.

4.2.14.1 Creating and Editing Restriction Filters

To create or edit restriction filters:

- Click the Add button, or select the restriction filter to edit and click the Edit button. The Edit Restriction Filter Details window is displayed.
- Complete the following window items:
  - **Filter Name:** Enter a unique name for this restriction filter.
  - **Database:** Select the ObjectServer database for which you are creating the restriction filter.
  - **Table**: Select the ObjectServer database table for which you are creating the restriction filter.
  - **Text area:** Enter the SQL text for the restriction filter. For example: WHERE GID=10 AND Tally > 100 AND Severity >=4
- Click **OK**. The restriction filter is saved and displayed in the Restriction Filter Details window.

4.2.14.2 Deleting Restriction Filters

To delete a restriction filter: Select the restriction filter to delete. Click the **Delete** button. The restriction filter is deleted.

4.2.15 Accessing ObjectServers Using the SQL Interactive Interface

This section describes how to use the SQL interactive interface to connect to ObjectServers and use SQL commands to interact with and control ObjectServers. The SQL interactive interface allows you to perform tasks such as creating a new database table or stopping the ObjectServer.

To open and use the SQL interface:

- In the Netcool/Administrator button bar, click the ObjectServers button. The ObjectServer Report window is displayed.
- Right-click the ObjectServer with which you want to interact.
- From the pop-up menu, select Show. The ObjectServer Configuration window is displayed.
- Click the System button.
- Select the SQL button. The SQL window is displayed.
- To issue a command, type the command in the text field at the top of the window and click the **Go** button. You can use a semicolon to separate multiple commands. You can also use the SQL helper buttons to facilitate the creation of SQL commands.

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

Lab: 15) Interactive SQL

Use the SQL window to check the status of the tables and other objects created earlier in the lab.

5 Other Key Functionality

5.1 Authentication via ObjectServer and LDAP

The ObjectServer is now PAM-enabled, adding support for LDAP and central ObjectServer authentication. Existing UNIX and NIS authentication continues to be supported via appropriate PAM modules.

5.1.1 ObjectServer Pluggable Authentication Module (PAM)

The ObjectServer Pluggable Authentication Module (PAM) is a dynamic library that can be loaded by the Operating Systems PAM system to allow a system to be authenticated against an ObjectServer. The ObjectServer itself is a PAM aware client application which can use these system PAM services. This feature enables an ObjectServer to authenticate its locally connected clients with a remote ObjectServer that is acting as a centralised authentication server.

The ObjectServer PAM module provides authentication and password management capabilities. This allows users to update their password in the central ObjectServer via the PAM system. Deploying the ObjectServer PAM module is simple, but once deployed allow users to be authenticated in a central location and update their password in this central location without using gateways.

As the ObjectServer is now a PAM aware client application and you do not wish to use the ObjectServer PAM module, you can configure you PAM system to use either UNIX, NIS or LDAP modules as the central authentication systems for users connecting to an ObjectServer.

5.1.1.1 PAM module installation

To install the ObjectServer PAM module on the local machine, run the installation script shown below and follow on-screen prompts.

```
${OMNIHOME}/install/nco_install_ospam
```

Do not forget to enable PAM support in the ObjectServer by adding or setting the ObjectServer property, '`Sec.UsePam`' to 'True'.

The ObjectServer PAM module is configured in two locations as detailed below.

1. The global ObjectServer PAM module configuration file, pam_omnibus_os.conf, is read when the module is invoked, from the ${OMNIHOME}/etc/ directory, if it exists. Here you can store global settings for all invocations of the ObjectServer PAM module via various PAM client applications. This file is a standard properties file. The current properties that can be defined in this file are defined below.

    - Server: The name of the ObjectServer that will be used to authenticate users. The default server name is NCOMS.

    - Debug: The module uses the syslog() call to log debugging information to the system log files. Default setting is False.

    - LogToStderr: Send all messages to the stderr stream of the process that is running the authentication module, if one exists, instead of to syslog. Default setting is False.

2. The module can also be configured via arguments to the module that are defined in the system PAM configuration file pam.conf. The list of arguments that can be use are defined below.

The following options may be passed to the ObjectServer PAM module.

- Debug The module uses the syslog() call to log debugging information to the system log files.
- no_warn Instruct the module to not give warning messages to the application.

- use_first_pass The module should not prompt the user for a password, it should obtain the previously entered password and use that one. This module is used when it is not the first module in the stack. Option is intended for auth and password modules only.

- try_first_pass This option is similar to 'use_first_pass', but is only intended for auth modules.

- server=XXXXX Specifies the name of the ObjectServer that will be used to authenticate users. Setting the server name as an argument to module overrides any server setting in the global configuration file. XXXXX is the name of the server.

- log_to_stderr Send all messages to the stderr stream of the process that is running the authentication module, if one exists, instead of to syslog. This is useful in debugging problems without having to examine the system logs.

The module provides the following services:-

Management Group	Functions
PAM_SM_AUTH (auth)	pam_sm_authenticate
PAM_SM_PASSWORD (password)	pam_sm_chauthtok
PAM_SM_SESSION (session)	pam_sm_open_session, pam_sm_close_session

5.2 Probes

The v7 probes can now forward events to any non-system table in an ObjectServer. The tables may be in the same or different ObjectServers. Each event may only be sent to one destination.

The new syntax for sending alerts to different tables and ObjectServers provides three new rules file functions

registertarget: defines a target name for a specific ObjectServer, backup, and table name combination

setdefaultarget: sets the named target as default until instructed otherwise

settarget: sets a target for the current event

The format of the registertarget function is:

```
target = registertarget ( <string servername>, <string
backupservername>,
<string alertstable> [,string detailstable] )
```

e.g.

```
London = registertarget( "NCOMS", "", "alerts.london" )
Sydney = registertarget( "SYD_P", "SYD_B", "alerts.sydney" )
```

Calling setdefaulttarget(London) will set the default destination for events to table alerts.london on server NCOMS.

Calling settarget(Sydney) will set the target for the current event to the Primary server SYD_P, backup SYD_B, and table alerts.sydney

Any given server may not appear in more than one server/backup pairing. I.e. if B is the backup to A, it cannot be the backup to any other server, it cannot be the primary to any other server.

The functions setobjectserver & setdefaultobjectserver will provide the same behaviour as in previous versions for backward compatibility. The target tables will be set to the values of the properties OplStatusTableName (default “alerts.status”) and OplDetailsTableName (default “alerts.details”). However the two styles are mutually exclusive in the rules for a single instance of an executing probe.

5.3 Gateways

5.3.1 New ObjectServer Gateways

Unlike the ObjectServer Gateway for Netcool/OMNIbus 3.x, the ObjectServer Gateway for Netcool/OMNIbus version 7 is not a writer within the gateway server binary (`nco_gate`). Instead, there are two binaries: one for unidirectional gateways (`nco_g_objserv_uni`) and one for bidirectional gateways (`nco_g_objserv_bi`). Both binaries use the Netcool Gateways Toolkit (NGTK) library, which provides the basic framework for the gateway process and are configured independently of each other. The NGTK library is installed as a part of Netcool/OMNIbus version 7.

Separate licenses are required for the unidirectional and bidirectional ObjectServer gateways. You also require a resynchronizing license if you want to use the resynchronization feature of the gateway.

A separate ObjectServer gateway guide is available as part of the documentation set that should be used for detailed descriptions of the administration of the gateway. This workshop simply describes the key components that need to be configured to use the new gateways.

Note that additional functionality has been added to the new gateways in the 7.0.1 release to include missing options that were provided in v3. These options are “ORDER BY” and “UPDATE TO INSERT CHECK”. Consult the gateway guide for the revised format of the replicate command

5.3.1.1 Configuration

The ObjectServer Gateway uses a centralized property management library; this separates properties from data processing configuration. Both the unidirectional and bidirectional gateways use the following set of configuration files:

- **Properties file**: define the gateway’s operational environment, such as, connection details and the location of the other configuration files.
- **Map definition file**: The gateway can replicate specific system tables and any user table in the ObjectServer. To do this, the gateway maps data to the appropriate fields in the ObjectServer using a map definition file. This contains mappings that define how the gateways map this data.
- **Startup command file**: The startup command file contains a set of commands that the gateway executes each time it starts. These commands allow the gateway to transfer any subsidiary table data to a set of target tables.
- **Table replication file**: The gateway can replicate the data in specific system tables and any user table between ObjectServers in a backup pair. Details of the tables to be replicated are stored in the Table Replication Definition file

In v7 there is a difference in transfer between system tables and other "user" tables. Many of the system tables now have relationships that have to be preserved and so cannot be transferred without supporting code underlying each case. Refer to the default configuration files contained in the directory $OMNIHOME/gates/objserv_uni and $OMNIHOME/gates/objserv_bi for the structure of the various components.

Note that Automations cannot be replicated via IDUC gateway operations. The confpack utility may be used to migrate automations between ObjectServers.

5.3.1.2 Failover

Failover functionality comes into operation when a gateway connects to a resilient pair of ObjectServers. For example, when connecting a Reader to a resilient Master ObjectServer pair to forward events to a Display ObjectServer. When the gateway loses its connection to the primary ObjectServer of the pair; Failover enables the gateway to connect to a backup ObjectServer. Failback functionality also enables the gateway to reconnect to the primary ObjectServer when it becomes active again.

To set up failback, set the ObjectServer's `BackupObjectServer` property for the backup ObjectServer to `TRUE`.

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

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

5.3.1.3 Resynchronization

The gateway supports two resynchronization modes:

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

To specify that the gateway uses resynchronization, set the `Gate.Resync.Enable` property to `TRUE`. To specify which type of resynchronization the gateway uses, set the `Gate.Resync.Type` property. When two ObjectServers are linked by a bidirectional gateway, it regards one as the master and the other as the servant. By default, the gateway treats the ObjectServer that has been running the longest as the master.

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

Lab: 16) ObjectServer Gateways

Examine the configuration files, props, map, tablerep.def and startup.cmd for the uni, and then bi directional gateways. Setup a bidirectional gateway to create a pair of resilient ObjectServers. Create a DualServerDesktop and configure a uni-directional gateway to feed events from the Master pair.

Lab: 17) Multiple alert tables

Configure your second ObjectServer, with a second table for receiving alerts. One way to achieve this is to cut the definition of alerts.status into a text file. Change the table name, and read the file into nco_sql.
Configure your simnet probe to forward selected events (perhaps all link up/down events) into the new table using the new "target" parameters.
Configure a backup ObjectServer, and include the new file in a bi-directional gateway.

5.4 Desktops

5.4.1 Tools

No changes have been made directly to the Tools sub-system. However, the introduction of procedural SQL enhances the power of desktop tools. This results in simplified maintenance and provides significant reductions in the number of messages between the desktop and the ObjectServer. Network traffic is reduced and the processing load in the ObjectServer is also reduced.

For example, the “Acknowledge” tool in v3.x requires two SQL statements in the tool.

```
update alerts.status set Acknowledged = 1 where Serial in (
$selected_rows.Serial );
insert into alerts.journal values (.... );
```

Each time the tool is used, this generates a message for the update statement, and an insert message for every row selected. Each of these messages is constructed in the Desktop, then sent to the ObjectServer which parses and executes each statement.

In Netcool/OMNIbus v7 a stored procedure can be created to build the required SQL in the ObjectServer:

```
create procedure AckProc( in serial_array array of integer )
begin
	for each row myrow in alerts.status where myrow.Serial in
serial_array
	begin
		set myrow.Acknowledged = 1;
		insert into alerts.journal values ( ... );
	end;
end;
```

The desktop tool is then redefined as:

```
execute AckProc( [ $selected_rows.Serial ] );
```

The tool only sends one message to the ObjectServer regardless of the number of rows selected.

The task of parsing, type checking, and validating SQL procedures is performed when they are created. This is not required at execution time leading to further performance enhancements.

Lab: 18) Tool Modification

Create the acknowledge procedure described in this section. Add a username parameter to include in your journal text. Create a new tool for your desktop to invoke the procedure passing the username and array of selected serial numbers. Test that your procedure correctly updates the selected alerts.

5.4.2 Use of Top in the Event List

The Event List now supports the use of the Top command to further restrict the number of events that match the selected Filter that will be displayed in the view.

A user with access to the View builder can enable the use of Top for the view, and specify parameters controlling the number events that may be displayed in the filter.

A parameter “Set from Event List” may also be set if the value of Top is to be modifiable from within the view.

Using these parameters an Administrator can restrict the use of the Top functionality for those operators with Read Only access to the view configuration.

In the Restrict Rows area, select the **Restrict rows [1-100]** check box. The **Minimum**, **Maximum**, and **Default** number entry fields are enabled. Enter the required values to control the range of numbers that can be applied to Top within this view. The restriction range default 1-100 is set by the ObjectServer property DTMaxTopRows.

Select the **Set from Event List** if you want event list view users to be able to choose the number of events to display in their event list view.

If you select **Set from Event List**, the number of events that users can choose to view using the event list controls is constrained by the **Minimum** and **Maximum** values specified in the View Builder. If you do not select **Set from Event List**, the number of events displayed is the **Default** value specified.

Note: If you select the **Restrict rows [1-100]** check box, and save the configuration as an .elv file, the .elv file is not compatible with Netcool/OMNIbus version 3.6 and earlier.

At the top of the Event List view there is now a **Top [*Min-Max*]** text box. To change the number of rows displayed, enter the required number in the box. On first opening the window the view will display the Default number of events.

If the text box label reads **Top [FIXED]**, you are not permitted to change the number of alerts displayed. The default value will always be displayed.

If the text box label reads **Top [OFF]**, then the Top facility is not enabled for this view. All events matching the filter will be displayed.

Lab: 19) Top in Event List

Explore the Top feature using the Min, Max and default values lower than the number of events in your view. Test in the view the effect of selecting the various parameters.

5.4.3 Load Balanced Mode

In a configuration where there are a group of desktop ObjectServers, it is likely that the number of event list users logged into each desktop ObjectServer will not be even. In extreme cases, all could be logged into one desktop ObjectServer, leaving the remainder idle. Load balanced mode automatically distributes event list user logins among a specified group of desktop ObjectServers according to a weighting specified by the Netcool administrator. This process is transparent to the event list user.

5.4.4 Configuring Load Balanced Mode

Load balanced mode is configured using the master.servergroups table. This can be done in the master ObjectServer and copied to all other ObjectServers, including desktop ObjectServers. Alternatively, the table data can be entered into each ObjectServer using the nco_sql tool.
The following table describes the format of the master.servergroups table:

Field	Data type	Description
ServerName	varchar(11)	The name of the desktop ObjectServer.
GroupID	int	Specifies the group to which each desktop ObjectServer belongs. Event list user logins are only distributed among desktop ObjectServers having the same GroupID.
Weight	int	Specifies 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.

Example Weighting
A system is set up with four desktop ObjectServers, DISP_A, DISP_B, DISP_C, and DISP_D.

DISP_A can support 1/6 of the connections.
DISP_B can support 1/3 of the connections.
DISP_C can support 1/2 of the connections.
DISP_D is not available for load-balanced connections.

DISP_A, supporting the least number of connections is given the weight 1. DISP_B, supporting twice the number of connections as DISP_A is given the weight 2. DISP_C, supporting three times the number of connections as DISP_A is given the weight 3. DISP_D is not accepting load balanced connections so is given a weight of 0.
All of the ObjectServers are given the same GroupID so that connections can be redirected between them.
The master.servergroups table contains the following data:

ServerName	GroupID	Weight
DISP_A	1	1
DISP_B	1	2
DISP_C	1	3
DISP_D	1	0

Example Load balanced groups
The system described above is now extended to cater for two additional desktop ObjectServers, DISP_E and DISP_F. These desktop ObjectServers can support the same number of connections between themselves, but do not share load-balanced connections with the existing ObjectServers. DISP_E and DISP_F are assigned a GroupID of 2 and both have a weighting of 1.

The master.servergroups table now contains the following data:

ServerName	GroupID	Weight
DISP_A	1	1
DISP_B	1	2
DISP_C	1	3
DISP_D	1	0
DISP_E	2	1
DISP_F	2	1

5.5 Restriction Filters for Non-Desktop Users

Restriction filters are now applied to SQL statements for all users connected to the ObjectServer except for “standard” users, (primarily gateways in this context).

This feature is probably most applicable to Impact where the user connecting can be modified.

Lab: 20) Non-desktop restriction filters

If you have the opportunity to test this functionality create a user and optionally group for your application with a restriction filter. Test that the filter is applied to the relevant tables.

5.6 Profiling and Monitoring

Netcool/OMNIbus v7 provides significantly enhanced profiling data that may be used both manually and in Triggers to monitor the ObjectServer performance and identify areas for action.

Detailed examination of this topic is beyond the scope of this introductory workshop. However useful examples are available to assist users in developing their knowledge with practical information.

Key profiling information is stored in the tables catalog.profiles for all connected clients, and catalog.trigger_stats for active Triggers. The SQL usage data in these tables provides valuable loading data. A detailed practical example of using this data may be found in the granularity_check automations provided in the Contributory directory described in section 5.7.

Connection data enables more detailed monitoring of Operator activity. Example automations making use of that data to monitor password failures are included in the default ObjectServer in Trigger Group Security_Watch. These automations may be viewed and adapted in the Automations component of nco_config.

5.7 Configuration Replicator

Configuration Replication with the nco_confpack utility provides the means to export full or partial ObjectServer configuration data to a package file for import to other ObjectServers

The utility provides a reliable means of replicating complete ObjectServer configurations across multiple sites, and subsequently rolling out additional partial configurations from development to live ObjectServers.

The Replicator may also be used to provide a backup of your ObjectServer configuration by copying the package file created by a full export to backup media.

5.8 Contributory Directory

A contributory direct has been added to the CD and will be available for download. This directory provided example automations and scripts developed

The contributory directory contains the following tar files:

failover_autos.tar
: a revised script for controlling automations in failover ObjectServers

granularity_check.tar
: a set of automations that make use of the improved profiling within v7 to adjust the granularity period of the ObjectServer according to configurable load parameters.

new_connection_watch.tar
: automations that make use of new information fields within connectionwatch messages that may be used in place of the simple default automations.

objectserver_heartbeat.tar
: automations that create and monitor ObjectServer heartbeat events

Silent_Manager.tar
: Example automations demonstrating how v7 can monitor event activity identifying devices that have been silent for a period of time.

nco_av_convert.Beta-1.6.solaris.tar
nco_av_convert.Beta-1.6.linux.tar
: Utilities to assist in the migration of Desktop configurations to Webtop.

ncw_export.1.2-Beta12.solaris.tar
ncw_export.1.2-Beta12.linux.tar
: Utilities to assist in the migration of ObjectiveView maps to Webtop format.

Each tar file contains a configuration package file and/or an SQL file, and a readme. The readme explains what the package is for and how to install it.

Note: The failover_autos.tar package contains a shell script and is not currently provided with a Windows version.

Untar the tar files to a temporary directory using the command: tar xvf <filename>. On Windows, you can use a file compression/extraction utility such as WinZip.

The files provided assist configuration and customization of your Netcool/OMNIbus and Netcool/Webtop installation. It is not expected that you deploy all of the files listed, or even any; they are provided to represent typical scenarios and workflow actions. However, they should be examined and further refined since the requirements for which they are intended cannot be sufficiently generic for all deployments.

If you use any of any of the files provided, it is recommended that you do so initially in a test environment. Only when you are satisfied that they are suitable should they be deployed in a live system, then only by a qualified Netcool/OMNIbus administrator.

5.9 Gateway Deduplication

V3 ObjectServers provide variable behaviour on deduplication via the gateway by setting the GWDeduplication property. This property exists within v7, but has no hard-coded effect within the ObjectServer.

This functionality can be enabled by use of a trigger that may reference the GWDeduplication property, or may be structured to behave differently depending on the specific gateway connection.

An example trigger to achieve the gateway deduplication functionality based on the value set in the GWDeduplication property is shown below. If this trigger or similar functionality is enabled, the default State_Change automation may be disabled.

```
create or replace trigger gateway_update
 group default_triggers
 priority 1
 before update on alerts.status
 for each row
 declare
   gw_dedup char( 255 );
   time_now utc;

 begin
   set gw_dedup = get_prop_value( 'GWDeduplication' );
   set time_now = getdate();

   -- If this is not a gateway treat as a normal update function
   if( %user.is_gateway = false ) then
     set new.StateChange = time_now;
     set new.InternalLast = time_now;
   else
     case
       -- Do not increment Tally
       when( gw_dedup = '0' )
       then
         set new.Tally = old.Tally;
         set new.StateChange = time_now;
         set new.InternalLast = time_now;

       -- Just replace the 'old' row with any supplied 'new' row values
       when( gw_dedup = '1' )
       then
         set new.StateChange = time_now;
         set new.InternalLast = time_now;

       -- Drop the update
       when( gw_dedup = '2' )
       then
         cancel;

       -- Increment Tally
       when( gw_dedup = '3' )
       then
         set new.Tally = old.Tally + 1;
         set new.StateChange = time_now;
         set new.InternalLast = time_now;
```

```
        -- Any other value is taken to be a drop
        else
          cancel;
      end case;
    end if;
end;
```

Appendix A. Upgrade Notes

A.1 Upgrade and Migration

These notes provide an overview of the upgrade and migration process. See the installation guide for full details. The upgrade and migration script may not work fully with versions prior to v3.5. In particular there are significant additions to the ObjectServer tables in both v3.5 and v3.6 providing valuable additional functionality that would be lost if migrating directly from a 3.4 ObjectServer. Customers are encouraged to upgrade to at least 3.5 before migrating to v7.

V7 will install as an upgrade to v3.x including an assisted migration of the ObjectServer databases, or as a full new install.

V7.0.1 will install as an upgrade to v3.x or v7, or as a full new install. When upgrading a v7 installation no database migration is necessary.

To upgrade from a previous release you must cleanly shut down the Netcool/OMNIbus processes running in the installation that you wish to upgrade. If running in Region Storage mode, ensure that you create current .dat files for each ObjectServer that you wish to migrate. The upgrade script will process each ObjectServer that has an entry in the interfaces.<arch> file.

The following example is of a v7 installation. V7.0.1 has a similar format.

Note. When upgrading an HP installation, ensure that you are not installing from the current OMNIHOME directory. Attempting an upgrade from this directory will overwrite your existing files.

You must run the OINSTALL script not the script upgrade.sh. Select the upgrade option by replying y to the initial question :

```
Netcool/OMNIbus 7.0 Installation
For Solaris.
Micromuse Inc.

Do you wish to perform an Upgrade, (rather than a full install)?
(y/n)?.
```

Select the components that you wish to install, and insure that the installation directory points to the correct directory for your existing installation.

```
Netcool/OMNIbus 7.0 Installation
For Solaris.
Micromuse Inc.

   1.  Desktop             SELECTED
   2.  Gateways            SELECTED
   3.  Process Control     SELECTED
   4.  Server       SELECTED
   5.  Confpack            SELECTED
   6.  Administration UI   SELECTED
   7.  Common Files        SELECTED

1-7. Toggle Component
S. Select All Components
U. Unselect All
I. Install Selected Components
C. Change Install Directory: /opt/netcool/omni36
H. Help
Q. Quit the Install
Option:
```

When you select option I, your entire existing directory structure is copied to the directory `$OMNIHOME.old` and the upgrade proceeds.

```
Netcool/OMNIbus 7.0 Installation
For Solaris.

Micromuse Inc.

Installing for Solaris
Info: Moving /opt/netcool/omni36 to /opt/netcool/omni36.old
Installing Common Components
Installing Patch "omnibus-7.0-solaris2-common-upgrade-7.0_1" ...
Installing Patch "omnibus-7.0-solaris2-common-libOpl-8_0" ...
Installing Process Control
Installing ObjectServer
Installing Desktop
Installing Gateways
Installing Confpack
Installing Administration User Interface
Installation complete
Installation log in /opt/netcool/omni36/NetcoolOMNIbusInstall.log
Upgrading 3.6 -> 7.0
Warning: This upgrade only copies standard configuration Files e.g.
nco_pa.conf, *_GATE.conf. Any other non-standard files will need to be
copied manually.
Upgrading Common Components
Info: To ensure new utils provided in 7.0 are not overwritten by
previous versions a backup of the 7.0 utilities has been created in
/opt/netcool/omni36/install/utils7.0

Upgrading Desktop
Upgrading Probes
Info: The old distribution's props and rules files will be saved until
probes are installed into /opt/netcool/omni36/probes/solaris2.
```

Note the message indicating that the probe configurations will be saved during this installation. The install does not halt at this message, but proceeds to the initial migrate phase for the ObjectServers.

```
Upgrading ObjectServer NCO36
Warning: Using the same name for the new v7 ObjectServer.
         If Netcool/Reporter is being used, this could lead to possible
         duplicate records, because of the Serial numbers.
Warning: Different widths for column AlertKey in alerts.status: 255 and
64
```

{various warning messages may appear here after each ObjectServer that are repeated in the log file for reference in later stages described in section A.2}

```
Upgrading Process Control
Upgrading TSMs
Info: The old distribution's props and rules files will be saved until
probes are installed

Netcool/OMNIbus 7.0 Installation
For Solaris.

Micromuse Inc.

Netcool/OMNIbus has been installed in /opt/netcool/omni36.

You will need access to one or more license servers to run
Netcool/OMNIbus.
To ease administration we recommend that you only install license
servers
where necessary.
You can install a license server on this machine by running
./license_install.
Do you wish to install a license server immediately?
(y/n)?
```

You may install the new license server now, or use an existing instance if available. If replacing an older version of Flex, first take a backup of your existing license files (.lic) as the new version will simply install in place of the old, deleting the existing license files. The following example assumes a new install over an old version.

```
#####################################
##  Netcool/Licensing installation  ##
```

```
#####################################


Checking platform for correct operating system version...

Please enter the directory for the installation.
[/opt/netcool/omni36/../common/license]

The installation directory already exists. If you continue please make
sure you
have a backup of any license files in
/opt/netcool/omni36/../common/license/etc/*.lic
as they will be removed. Do you wish to continue
and overwrite the installed contents of this directory (y/n)? [n]
y
Removing the contents of /opt/netcool/omni36/../common/license ...
... Done removing the contents of /opt/netcool/omni36/../common/license

Installing base files ...
... Done installing base files.

This script copies a startup script into the /etc/init.d directory to
enable you to automatically start Netcool/License processes when the
system boots and stop them on system shutdown

It does this by:
        Copying /opt/netcool/
omni36/../common/license/platform/solaris2/etc/nclicense to
/etc/init.d/nclicense
        Linking /etc/rc0.d/K65ncl to /etc/init.d/nclicense
        Linking /etc/rc1.d/K65ncl to /etc/init.d/nclicense
        Linking /etc/rc2.d/S81ncl to /etc/init.d/nclicense

*************************************
Do you wish to install /etc/init.d/nclicense now (y/n)? [y]
y
The file /etc/init.d/nclicense already exists, do you want to overwrite
it (y/n)? [y]
y
NOTE:
        You may wish to check or further edit the startup script
        in /etc/init.d/ before installing the startup symlinks in the
platform
        specific directory.
Do you wish to install the symlinks now (y/n)? [y]

The system has been modified
nco_hostcode v1.04 (15-OCT-2002 11:20:01)
---OUTPUT-BEGIN---
Version: v1.04
Time: 1083166562
cksum: 0017e690
[hex character strings]
---OUTPUT-END---

USAGE:

1. Request your license using this code at
   http://support.micromuse.com/helpdesk/licenses/
2. Define the environment variable
NCLICENSE=/opt/netcool//common/license.
3. Place the directory /opt/netcool/omni36/../common/license/bin in your
executable PATH.
4. Install the licenses you receive from Micromuse into
   /opt/netcool/omni36/../common/license/etc
   in a file with the extension .lic
5. Start the license server by running
   /opt/netcool/omni36/../common/license/bin/nc_start_license
Successfully installed in /opt/netcool/omni36/../common/license
Installation log in /opt/netcool/
omni36/../common/license/log/nc_license_install.log
```

You may now copy back your 3.6 flex licenses as these are valid for the new license server. You will need to request new licenses for the ObjectServer gateways, nco_config and nco_confpack components. These may be added into your licensing etc directory as a separate

.lic file as Flex will read all valid .lic files from this directory on startup or after a nc_read_license command is issued to a running flex server.

After the new installation is created, the following files will have been copied from the `$OMNIHOME.old` directory to the new version 7 installation:

- Communications file ($OMNIHOME/etc/omni.dat)
- Database files ($OMNIHOME/db/*.*, $OMNIHOME/etc/*.props, and OMNIHOME/etc/*.sql)
- License files ($NCLICENSE/etc/*.lic)
- Configuration files (*.conf)

The upgrade only copies configuration files that use default names, for example, `$OMNIHOME/etc/nco_pa.conf` and `$OMNIHOME/*/*GATE.conf`. Any other configuration files must be copied manually. Probe properties and rules files are copied from the `$OMNIHOME.old` directory to `$OMNIHOME/probes/`*`arch`*. The Netcool/OMNIbus version 3.6 default probe rules and properties files are in `$OMNIHOME/probes/default`.
TSM properties and rules files are copied from the `$OMNIHOME.old` directory to `$OMNIHOME/tsm/`*`arch`*. The Netcool/OMNIbus version 3.6 default TSM rules and properties files are in `$OMNIHOME/tsm/default`.
Monitor properties and rules files are copied from the $OMNIHOME.old directory to $OMNIHOME/monitors/arch. The Netcool/OMNIbus version 7 default monitor rules and properties files are in $OMNIHOME/monitors/default. Profiles, data logs, and demos are also copied.
Desktop default.elc files are copied from the $OMNIHOME.old directory to $OMNIHOME/desktop. The Netcool/OMNIbus version 7 default files are copied to default.elc.orig.
Utilities are copied from the $OMNIHOME.old directory to $OMNIHOME/utils. To ensure version 7 utilities are not overwritten by previous versions, a backup of the version 7 utilities is created in $OMNIHOME/install/utils7. Any old scripts in $OMNIHOME/scripts overwrite the new scripts.
The nco_os_migrate script is run automatically against each ObjectServer in the db directory. This will produce intermediate SQL files representing your v3.x ObjectServer migrated to v7 formats. Diagnostic messages will be produced indicating any points that may need investigation. Creation of your new database using these files as input is described in the section A.2.

Probe migration is not entirely completed by the Netcool/OMNIbus upgrade option. Once the Netcool/OMNIbus upgrade is completed, run the probe install script PINSTALL as described for a new installation. Provided that OMNIHOME is set to the directory where the Netcool/OMNIbus upgrade was installed the script will detect the presence of the partial probe configuration upgrade and complete the required processing including installation of the v7 probe versions.

```
Netcool/OMNIbus 7.0 Probe Installation
For Solaris.

Micromuse Inc.

Installing for Solaris
Installing Probes
Installation complete
Installation log in /opt/netcool/omni36/NetcoolOMNIbusProbeInstall.log
Taking copy of original probe configuration
Taking copy of original tsm configuration
Restoring probe configuration in upgrade
Restoring tsm configuration in upgrade
Updating file ownership
Updating directory permissions
```

The installation phase of the upgrade to your system is now complete. Completing the database migration is described in section A.2.

Note: The Windows upgrade will require more care and intervention because only one version of Netcool/**OMNIbus can be resident on a single Windows platform. Refer to the Installation guide for more details.**

A.2 Completing the Database migration

The nco_os_migrate script produced intermediate SQL files representing your v3.x ObjectServer migrated to v7 formats. Diagnostic messages will be found in the install log indicating any points that may need investigation. For example upgrading the NCO36 ObjectServer:

```
Upgrading ObjectServer NCO36
Warning: Different widths for column AlertKey in alerts.status: 255 and
64
```

A set of files are created in the v7 directory $OMNIHOME/etc for each migrated database; for example:

```
NCO36.props
NCO36_alertsdata.sql
NCO36_application.sql
NCO36_automation.sql
NCO36_desktop.sql
NCO36_nco_dbinit.props
NCO36_security.sql
NCO36_system.sql
```

To deal with the warning shown above, you would edit NCO36_application.sql and set the size of AlertKey to 255 if you decided to increase your AlertKey size from the old 64 bytes to the new default.

Notes:

The conversion of v3.6 triggers that use a select * from alerts.status can result in an excessive evaluate clause in the migrated sql. This will only occur when alerts.status has many additional fields or fields with very long datanames. In v7.0.1 the allowable length has been increased from 1024 to 2048 bytes reducing the likelihood of this problem occurring. An error message will be produced during migration identifying clauses with excessive length.

You may manage this situation either by modifying the 3.x trigger statement prior to migration to select only the fields that are required for the related action, or by manually editing the sql file produced by the migration script to remove unnecessary fields from the evaluate clause.

V3 actions which are not linked to a trigger will not be migrated.

UserID values are restricted to a maximum value of 65534. Values greater than this will need to be managed manually in the resulting sql.

The file automations.sql will include the new v7 automations, and the migrated versions of the v3 automations from the ObjectServer with the names prefixed v3_ to avoid naming conflict. V7 versions of existing automations are disabled by default. The v3 automations will run with the expected results, but will not be optimized to take advantage of the new v7 functionality. It is recommended that you run initially with the v3 automations and reference the improved v7 profiling data to target for tuning those automations that are seen to create the heaviest load on the ObjectServer.

Then with the following command, you would create your migrated database in v7 format:

```
../bin/nco_dbinit -propsfile /opt/netcool/omnibus/etc/NCO36_nco_dbinit.props
```

Repeat the migration for each set of database files produced by the upgrade script.

You may also run nco_os_migrate from the command line to produce similar migration files for any other V3.5/V3.6 ObjectServer databases that you wish to migrate.

A.3 Post-Installation Tasks

When the installation is complete, you must:

- Set the required environment variables.
- Configure licensing if this is you first use of the FLEX licensing system.
- Configure the server communications

When configuring the nco startup scripts add the licensing environment variable NETCOOL_LICENSE_HOST to the script to ensure that this is correctly set on system boot.

Note. After installation or upgrade, audit functions are disabled by default regardless of the audit settings of the v3 ObjectServers.