IBM® Tivoli® Netcool/OMNIbus Gateway for JDBC
7.0

Reference Guide
August 21, 2020

SC22-5408-11

Notice

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 59.

Edition notice

This edition (SC22-5408-11) applies to version 7.0 of the IBM Tivoli Netcool/OMNIbus Gateway for JDBC and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC22-5408-10.

Contents

About this guide

The following sections contain important information about using this guide.

Document Control Page

Use this information to track changes between versions of this guide.

The IBM Tivoli Netcool/OMNIbus Gateway for JDBC documentation is provided in softcopy format only. To obtain the most recent version, visit the IBM® Tivoli® Netcool® Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/common/kc_welcome-444.html?lang=en

Table 1. Document modification history

Document version	Publication date	Comments
SC22-5408-00	December 02, 2011	First IBM publication.
SC22-5408-01	December 16, 2011	Requirements information updated in "Summary" on page 1. Script library information updated in Obtaining the gateway.
SC22-5408-02	March 02, 2012	Requirements information updated in "Summary" on page 1. List of supported databases updated in "Supported databases" on page 3. Information about obtaining the gateway updated in Obtaining the gateway. Database configuration information updated in "Configuring the database connection" on page 13. Sybase naming scheme issue described in "Known issues" on page 56.
SC22-5408-03	July 06, 2012	Package version updated in "Summary" on page 1. **`Gate.Jdbc.Url`** property descriptions for use with DB2 LUW and DB2 z/OS updated in "Configuring the database connection" on page 13. "Properties and command line options" on page 28 updated. ObjectServer table replication information updated in "Table replication definition file" on page 47. "AfterIDUC and Filter functions" on page 49 added. "Custom table labelling" on page 56added.

Table 1. Document modification history (continued)		
Document version	**Publication date**	**Comments**
SC22-5408-04	September 30, 2012	The descriptions for the following properties have been updated in the "Properties and command line options" on page 28: • **ConfigCryptoAlg** • **ConfigKeyFile** • **Gate.RdrWtr.Password** • **Gate.Jdbc.Password** "FIPS mode and encryption" on page 52 updated.
SC22-5408-05	November 30, 2012	Guide updated for Netcool/OMNIbus V7.4 release.
SC22-5408-06	March 14, 2013	Information regarding migrating from a persistent state gateway added to "Migrating from an existing gateway" on page 12. Support was added for IBM Netezza data warehouse. Information about how to use the internal JDBC driver for testing the database connection added to "Configuring the database connection" on page 13. The descriptions for the following properties have been updated in the "Properties and command line options" on page 28: • **Gate.Jdbc.SuppressDeletes** • **Gate.Jdbc.TestCount** Instructions to use the **TRANSFER** command to transfer data from one table to another added to "Startup command file" on page 46. Instructions to use the **Gate.ResynchTables** property in the Gateway NGtKTool kit to transfer data from dynamic tables added to "Table replication definition file" on page 47. Core dump issue described in "Known issues" on page 56.
SC22-5408-07	September 30, 2013	"Supported databases" on page 3 updated. A description for the following property has been added to the "Properties and command line options" on page 28: • **Gate.ResyncTables** Instructions to use the **TRANSFER** command to transfer data from one table to another improved in the "Startup command file" on page 46. Instructions for using the **Gate.ResynchTables** property improved in "Table replication definition file" on page 47.

Table 1. Document modification history (continued)

Document version	Publication date	Comments
SC22-5408-08	November 8, 2013	The `gateway-nco-g-jdbc-reporting-scripts` information when running the gateway in reporting mode has been updated in "Audit mode and reporting mode" on page 5 "Configuring SSL connections" on page 25
SC22-5408-09	March 12, 2015	Package version and Requirements updated in "Summary" on page 1. Removed the Obtaining the gateway topic. Obtaining the gateway is described in the installation related topics. See "Installing the gateway" on page 9. 64 bit support added.
SC22-5408-10	July 28, 2016	"Frequently asked questions" on page 56 added.

Table 1. Document modification history (continued)

Document version	Publication date	Comments
SC22-5408-11	August 21, 2020	Updated for version 7 of the gateway. "Summary" on page 1 updated. Description for the generic Netcool/OMNIbus property **Props.LiveUpdate** added to "Common Tivoli Netcool/OMNIbus properties" on page 28. **Props.LiveUpdate** is available in the Netcool/OMNIbus patch that fixes APAR IJ21246. Descriptions for the following new properties added to "Properties and command line options" on page 28: • **Gate.Jdbc.DriverPasswordPropertyName** • **Gate.Jdbc.DriverUserPropertyName** • **Gate.Jdbc.InitializationTimeout** • **Gate.Jdbc.InitializeAllSessions** • **Gate.Jdbc.JavaSystemPropsFile** • **Gate.Jdbc.JdbcPropsFile** • **Gate.Jdbc.PreconnectionWait** • **Gate.Jdbc.SqlTimeout** The following database integration sections added: • "Integrating with an Oracle database" on page 16 • "Integrating with a Microsoft SQL Server database" on page 21 • "Integrating with a DB2 database" on page 23 **Fixes:**Version 7 of the JDBC Gateway includes fixes for the following APARs: • IV98887: Limit result sets when validating fields to one row, so we can efficiently handle large target tables without invoking full table scans. IV79462: Add comment to reporting JDBC map relating to STATECHANGE column. It is only defined in the DB2 schema, and needs commenting out for other databases with the default schemas. The gateway wass enhanced for **RFE 137263: Netcool OMNIbus JDBC Gateway support for Kerberos Authentication to Oracle**. The scope and system configurations of the gateway's Kerberos testing: The following setup was used in the verification: Linux 7, Kerberos V5, Oracle 19c, IBM DB2 10.5. The KDC server and the target database were residing in same machine. Due to the setup complication encountered in the respective databases, combined security of Kerberos and SSL is not verified. The following gateway dependency bundles also updated:: • `gateway-libngjava-9` • `gateway-libngtktk-8` The paths specific to IBM Java, Oracle Java, and OpenJDK

Conventions used in this guide

All gateway guides use standard conventions for operating system-dependent environment variables and directory paths.

Operating system-dependent variables and paths

All gateway guides use standard conventions for specifying environment variables and describing directory paths, depending on what operating systems the gateway is supported on.

For gateways supported on UNIX and Linux operating systems, gateway guides use the standard UNIX conventions such as **$***variable* for environment variables and forward slashes (**/**) in directory paths. For example:

```
$OMNIHOME/gates
```

For gateways supported only on Windows operating systems, gateway guides use the standard Windows conventions such as **%***variable***%** for environment variables and backward slashes (****) in directory paths. For example:

```
%OMNIHOME%\gates
```

For gateways supported on UNIX, Linux, and Windows operating systems, gateway guides use the standard UNIX conventions for specifying environment variables and describing directory paths. When using the Windows command line with these gateways, replace the UNIX conventions used in the guide with Windows conventions. If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Note : The names of environment variables are not always the same in Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX and Linux environments.

Operating system-specific directory names

Where Tivoli Netcool/OMNIbus files are identified as located within an *arch* directory under NCHOME or OMNIHOME, *arch* is a variable that represents your operating system directory. For example:

```
$OMNIHOME/platform/arch
```

The following table lists the directory names used for each operating system.

Note : This gateway may not support all of the operating systems specified in the table.

Table 2. Directory names for the arch variable

Operating system	**Directory name represented by *arch***
AIX® systems	`aix5`
Red Hat Linux® and SUSE systems	`linux2x86`
Linux for System z®	`linux2s390`
Solaris systems	`solaris2`
Windows systems	`win32`

OMNIHOME location

Gateways and older versions of Tivoli Netcool/OMNIbus use the OMNIHOME environment variable in many configuration files. Set the value of OMNIHOME as follows:

- On UNIX and Linux, set $OMNIHOME to $NCHOME/omnibus.
- On Windows, set %OMNIHOME% to %NCHOME%\omnibus.

Chapter 1. Gateway for JDBC

The IBM Tivoli Netcool/OMNIbus Gateway for JDBC uses the standard Java™ Database Connectivity (JDBC) API to exchange alerts between Netcool/OMNIbus ObjectServers and external databases. It communicates with the supported databases using Java Type 4 JDBC drivers supplied by the database vendors.

The Gateway for JDBC can be used as a replacement for the Tivoli Netcool/OMNIbus Gateway for ODBC and the Tivoli Netcool/OMNIbus Gateway for Oracle.

The Gateway for JDBC has the following features:

- Reporting and audit modes of operation.
- Store and forward (SAF) capability using persistent event cache and data forwarding.
- Resynchronization on startup.
- Scalability provided by multiple database connections.
- Arbitrary table transfers.
- Migration from existing target databases using bidirectional resynchronization.

This guide contains the following sections:

Summary

Each gateway works in a different way to provide an interface to the ObjectServer. This summary describes the basic characteristics of the gateway.

The following table provides a summary of the gateway.

Table 3. Summary	
Gateway target	DB2® LUW, DB2 z/OS®, Informix®, Microsoft SQL Server, MySQL, Oracle, Sybase See "Supported databases" on page 3 for details of supported versions.
Gateway executable file name	`nco_g_jdbc`
Gateway installation package	`omnibus-`*`arch`*`-gateway-nco-g-jdbc-`*`version`*
Package version	7.0
Gateway supported on	For details of supported operating systems, see the following Release Notice on the IBM Software Support website: https://www-304.ibm.com/support/docview.wss?uid=swg21623766

Configuration Files	`$OMNIHOME/gates/jdbc/G_JDBC.props`
	`$OMNIHOME/gates/jdbc/audit.G_JDBC.props`
	`$OMNIHOME/gates/jdbc/audit.jdbc.map`
	`$OMNIHOME/gates/jdbc/jdbc.map`
	`$OMNIHOME/gates/jdbc/jdbc.rdrwtr.tblrep.def`
	`$OMNIHOME/gates/jdbc/jdbc.startup.cmd`
	`$OMNIHOME/gates/jdbc/reporting.G_JDBC.props`
	`$OMNIHOME/gates/jdbc/reporting.jdbc.map`

Requirements	A currently supported version of IBM Tivoli Netcool/OMNIbus . Audit mode script library (required to run the gateway in audit mode): `gateway-nco-g-jdbc-audit-scripts-1_0` Reporting mode script library (required to run the gateway in reporting mode): `gateway-nco-g-jdbc-reporting-scripts-1_0` Gateway Java Support Package: `gateway-libngjava-7` (or later) Gateway NGtkTK Support Package: `gateway-libngtktk-6` (or later) The Gateway for JDBC requires third party drivers to support connecting with the target database. For more information see "Supported databases" on page 3.
Multicultural support	Not available
IP environment	IPv4 and IPv6 **Note :** IPv6 support for the database connection depends on the JDBC driver. Consult your JDBC driver documentation for IPv6 support information.

Table 3. Summary (continued)	
Federal Information Processing Standards (FIPS)	IBM Tivoli Netcool/OMNIbus V7.3.0, 7.3.1 and 7.4.0 use the FIPS 140-2 approved cryptographic provider: IBM Crypto for C (ICC) certificate 384 for cryptography. This certificate is listed on the NIST web site at http://csrc.nist.gov/cryptval/140-1/1401val2004.htm. For details about configuring Netcool/OMNIbus for FIPS 140-2 mode, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

Supported databases

The Gateway for JDBC is supported for use with the databases listed here.

The gateway is supported for use with the databases and JDBC drivers listed in the following table.

Note : The Gateway for JDBC will support a database once the vendor supplies a suitable JDBC driver that works with the gateway. Drivers not listed in this table are not supported by IBM Software Support, but if a third party JDBC driver does not function correctly due to a fault with the Gateway for JDBC, then IBM Software Support will provide support.

Table 4. Supported databases and JDBC drivers	
Database	**JDBC Driver**
DB2 LUW	DB2 Type 4 Universal Driver (`com.ibm.db2.jcc.DB2Driver`)
DB2 z/OS	DB2 Type 4 Universal Driver (`com.ibm.db2.jcc.DB2Driver`)
Informix	IBM Informix JDBC Driver (`com.informix.jdbc.IfxDriver`)
Microsoft SQL Server	Microsoft JDBC Driver for SQL Server (`com.microsoft.sqlserver.jdbc.SQLServerDriver`) **Note :** On OMNIbus installations which are JRE 1.7 based the Microsoft JDBC Driver 4.2 up till 6.4 can be used.. For OMNIbus installations using JRE 1.8, Microsoft JDBC Driver 7.0 and above may be used providing that the driver used is the JRE 1.8 version. For more details on the Microsoft JDBC Driver, please refer to the Microsoft JDBC Driver for SQL Server website. Ensure that multiple versions of the `.jar` files are not installed in the `$CLASSPATH` or in the `$OMNIHOME/gates/java` location simultaneously.
MySQL	MySQL Connector/J (`com.mysql.jdbc.Driver`)
IBM Netezza data warehouse	Netezza JDBC Driver (`org.netezza.Driver`)
Oracle	Oracle JDBC Thin Driver (`oracle.jdbc.driver.OracleDriver`) **Note :** When the database connection is enabled with Kerberos authentication, the gateway must be run using Oracle Java. See: "Deploying the gateway with non-IBM Java" on page 46.

Table 4. Supported databases and JDBC drivers (continued)	
Database	**JDBC Driver**
Sybase	Sybase jConnect for JDBC (`com.sybase.jdbc4.jdbc.SybDriver`)

Overview of the gateway

The Gateway for JDBC uses the JDBC Manager built in to the IBM JVM (supplied with Tivoli Netcool/OMNIbus) to exchange alerts between Tivoli Netcool/OMNIbus ObjectServers and external databases. It communicates with the supported databases using Java Type 4 JDBC drivers supplied by the database vendors.

Gateway operation

In the default unidirectional resynchronization mode, the gateway processes Insert, Delete, Update, Control (IDUC) cycles as follows:

1. The reader-writer component receives data from the ObjectServer.
2. The reader-writer component formats the data into a map, keyed using field names, and passes the mapped data to the gateway manager component.
3. The manager component uses the type of the source table (status, journal, or details) and the lifecycle state of the alert (the alert exists or not in the gateway cache) to form the SQL statement required to send the alert update to the target database.
4. The manager component writes the SQL statement to the current batch.

   If there is no existing batch, the manager component creates one.
5. The manager component processes all the data in the current IDUC cycle, saves the batch to disk, and queues the batch into the gateway cache.

   The cache is saved as part of this process.
6. The consumer component reads the batch from the queue, reading it from the disk if necessary.
7. The consumer component processes all the SQL statements contained in the batch.
8. The processor component executes portions of the batch atomically.

Persistence and reliability

The gateway manages two pieces of persistent data: the cache of known alerts and outstanding batches, and the batches themselves. Batches are only processed after the batch and the cache have been successfully saved to disk. If the gateway shuts down unexpectedly, it should be possible to recover the committed state without duplicating or losing data.

Note : Successful recovery also depends on the proper operation of the underlying JVM and the host operating system.

The batching mechanism effectively operates as a store and forward (SAF) mechanism. Fatal errors, such as SQL parse errors or constraint errors, result in the bad data being logged and then discarded.

The gateway also supports atomic and durable updates to the cache and batch files. This protects against data corruption and ensures that the data in the gateway working area is always valid.

Resynchronization

The gateway can perform two types of resynchronization: unidirectional and bidirectional. The **`Gate.Jdbc.ResyncMode`** property provides four resynchronization modes to control how the gateway performs resynchronizations.

The following resynchronization modes are available:

1. None

   The gateway does not perform resynchronization.

2. Unidirectional

   The gateway compares the contents of its cache with the contents of the ObjectServer `alerts.status` table. New, deleted, and updated alerts are forwarded to the target database, thus resynchronizing it with the ObjectServer.

3. Bidirectional

   The gateway seeds its cache with any open alerts stored in the target database. It then compares the contents of its cache with the contents of the ObjectServer `alerts.status` table and resynchronizes the target database accordingly.

   Note : Bidirectional resynchronization results in full table scans on the target database status table. A large amount of event history will result in a long resynchronization.

4. Automatic

   This is the default resynchronization mode. It causes the gateway to operate in unidirectional resynchronization mode by default. However, if the gateway cache is empty on startup (which normally only occurs the first time that the gateway is run), it causes the gateway to operate in bidirectional resynchronization mode. In this mode, only the `alerts.status` and `alerts.journal` ObjectServer tables are resynchronized to the target database. You can manually resynchronize other ObjectServer tables using transfer commands.

Deduplication

The gateway deduplicates updates to alerts and journals. This results in efficient resynchronization, because only alerts that have changed are forwarded to the target database. It also means that updates to fields that are not included in the map definition file, or fields that are not required in the target database (such as the state change field), are ignored.

Audit mode and reporting mode

The gateway can operate in one of two modes, audit mode or reporting mode. In audit mode, the gateway archives events to a target database for auditing purposes. In reporting mode, the gateway archives events to a target database for use with reporting tools such as Tivoli Netcool/Reporter or Tivoli Common Reporting.

Audit mode

If you want to run the gateway in audit mode, you must configure the appropriate database schema. If you are using the Gateway for JDBC as a replacement for the Gateway for Oracle or the Gateway for ODBC, your existing target database configuration might not require any changes. Otherwise, you must install the `gateway-nco-g-jdbc-audit-scripts` library. This library contains the SQL scripts required to create the target database schema.

In audit mode, a new row is created in the database table for every new alert and alert update. The target database can contain multiple rows for each alert, depending on its update history. In audit mode, existing data in the target database is never updated or deleted.

Inserts, updates, and deletes are mapped as follows:

- New alerts cause a new row to be inserted into the target database.
- Alert updates cause a new row containing the updated alert values to be inserted into the target database. Previous alert values are preserved in previously created rows.

- Alert deletes cause a new row containing the final alert values to be inserted into the target database. Most of the alert data is no longer available at deletion time, so the delete row will contain mostly NULL values. Previous alert values are preserved in previously created rows.

Reporting mode

If you want to run the gateway in reporting mode, you must configure the appropriate database schema and install the `gateway-nco-g-jdbc-reporting-scripts` library. If you are using the Gateway for JDBC as a replacement for the Gateway for Oracle or the Gateway for ODBC, your existing target database configuration might not require any changes. Otherwise, you must install the `gateway-nco-g-jdbc-reporting-scripts` library. This library contains the SQL scripts required to create the target database schema.

Note : You must have the database and tablespace already installed and configured before running the SQL scripts.

In reporting mode, the target database contains one row for each new alert and that row is updated whenever the alert is updated. This requires that the target database be configured for reporting, which involves various triggers to generate the reporting data. In reporting mode, the gateway essentially replicates the source status table from the ObjectServer into the target database.

Inserts, updates, and deletes are mapped as follows:

- New alerts cause a new row, unique to the alert, to be inserted into the target database.
- Alert updates cause the existing alert row to be updated with the new alert values.
- Alert deletes cause the existing alert row to be updated with a deletion timestamp.

Table 5. Reporting and Audit mode is supported for the following platforms.

Database	Audit Mode	Reporting
DB2	Yes	Yes
Informix	Yes	Yes
Microsoft SQL Server	Yes	Yes
MySql	Yes	No
Netezza	Yes	No
Oracle	Yes	Yes
Sybase	Yes	Yes

Target database sizing

The main factors that affect database growth (and hence your database sizing requirements) are: the size of an event, the rate at which events are passed to the database, and the combination of the gateway operating mode (audit or reporting) and the mix of event types (insert, update, or delete). The following topics discuss these factors and how to assess your system to determine your optimal sizing requirements.

Tivoli Netcool/OMNIbus event size

The target database row size should correspond closely to the Tivoli Netcool/OMNIbus event size. This is because the supplied database schemas broadly mirror the Tivoli Netcool/OMNIbus schema.

The size of a row, or individual event, depends on the following factors:

- Sizes of individual fields
- Whether custom fields have been added
- Whether default fields have been omitted
- Whether fields are actually populated with data

The maximum allowable field size can be determined by *describing* the ObjectServer tables by running `nco_sql` and executing the following commands:

```
> describe alerts.status
> go
```

The maximum possible size of an event can be determined by a summation of a table's field sizes.

The following table shows the maximum allowable row sizes for a default Tivoli Netcool/OMNIbus 8.1 installation:

Table	Maximum row size
alerts.status	10284 bytes
alerts.journal	4347 bytes
alerts.details	1028 bytes

The following table shows the typical row sizes that you are likely to encounter in most practical applications:

Table	Typical operational row size
alerts.status	2048 bytes
alerts.journal	512 bytes
alerts.details	0 bytes

Event rate

The rate at which events are passed to the gateway for forwarding to the database can be assessed empirically (observed over a period of time), or an assessment can be made of the expected or required throughput.

The extent to which throughput correlates to input from data sources such as probes may be significantly affected by deduplication. Deduplication converts inserts into the ObjectServer (with the same **`Identifier`** values) to updates (incrementally increasing the **`Tally`** field) thus limiting ObjectServer size and potentially limiting the size of the target database. The effect of deduplication may be to reduce data volume by a factor of 10. To calculate the volume reduction that deduplication has on a particular data flow, calculate the ratio of actual events to inserted events in `alerts.status` by dividing the tally **`sum(Tally)`** by the count **`count(*)`**.

Filter conditions applied in gateway configuration will also restrict the gateway's interest to a subset of events placed into the ObjectServer, and so limit the size of the target database.

Operation mode

Database gateways operate in one of two modes:

- Audit: In audit mode, all Tivoli Netcool/OMNIbus event types (inserts, updates and deletes) are forwarded as inserts, thus maintaining an audit trail within the target database, subject to (or restricted by) the granularity of the IDUC cycle.

- Reporting: In reporting mode, both Tivoli Netcool/OMNIbus updates and deletes are forwarded as updates to previously inserted rows.

A gateway operating in reporting mode is likely to populate the database with less data. However, triggers included in the default reporting database schemas populate additional tables with summarized data, from which reports are run. Typically an additional table will be populated and maintained in the database for each report that may be run. These tables are relatively small in comparison to the main status table; see below for further information.

Note : Database gateways do not delete rows from target database tables. Thus target databases will grow in size indefinitely without database pruning or archiving taking place as a separate activity. This is independent of the mode of operation of the gateway.

Monitored tables

The configuration of the gateway determines which Tivoli Netcool/OMNIbus tables are monitored. Most users configure gateways to monitor just the `alerts.status` table, but you can also monitor the `alerts.journal` and `alerts.details` tables. The `alerts.details` table, even if monitored by the gateway, is generally not populated with data in default probe configurations. In general, it is populated only in debugging or setup scenarios. Currently, none of the default database gateway reports depend on data received from the `alerts.details` table.

Although database gateways are primarily used for receiving data from the three main tables (status, journals and details), other tables may be monitored in a custom setup. This would impact on database size requirements.

Journal entries

Journal entries fall into two main categories: those automatically generated by automations, for example when an event is acknowledged or deleted by a right-click tool defined in the event viewer, and those created by users. Automatically generated journals are generally short, and the ratio of the number of journal entries to the number of events is low. The size of user generated journals is determined by the individual user behaviour or policy.

Reporting tables

As mentioned above, the reporting schemas contain other tables in addition to the analogues of the three main Tivoli Netcool/OMNIbus tables (status, journal and details). These fall into two categories: tables from which reports are generated (one per report definition, currently there are four, generally named `rep_audit_`*`fieldname`*, and containing one row per status event) and tables of mostly static or rarely changing data. The second type are generally small and can be ignored in sizing calculations, or absorbed into margins for error. Rows in tables used for generating reports are typically less than 256 bytes.

Target implementation and tuning

Multiplying event rate by event size provides a good rough estimate of the target database size, but database implementation and tuning can easily, and significantly, increase these calculations. One factor to consider is block size and how empty or full you allow data blocks to become when they are updated.

Depending on the database implementation and the level of tuning, you can expect to increase the rough estimate by a factor two to four.

Assessing your system

One way to assess your system would be to run the Flat File Gateway and monitor the growth in its output file. However, note that the output of this gateway will be closer to that of a database gateway running in audit mode rather than reporting mode, because all event types are written (analogous to inserted) to the output file. Nevertheless an analysis of the output file will provide an insight into the mix of inserts and updates encountered in a particular system. For a database gateway running in reporting mode, updates and deletes can be largely discounted from calculations.

Note : 4 byte integers within an ObjectServer will equate to larger amounts of data in the output of the Flat File Gateway due to their representation in character format.

Use the following formula to calculate a rough annual database sizing requirement:

```
<inserts per day> * (<bytes per event> + (<number of report tables> * <bytes
per report table row>) * <52 weeks> * <7 days>) / <bytes per GB>
```

For example, using the following values:

- 10,000 inserts per day, after deduplication
- 2048 bytes per event
- 4 report tables
- 256 bytes per report table row
- 52 weeks
- journals and details not included

The annual database storage requirement would be:

```
(10000 * (2048 + (4 * 256)) * 52 * 7) / 1024^3 = 10.4 GB
```

```
<inserts per day> * (<bytes per event> + (<number of report tables> * <bytes
per report table row>) * <52 weeks> * <7 days>) / <bytes per GB>
```

Installing the gateway

There are separate procedures for installing the gateway on each version of Tivoli Netcool/OMNIbus.

Follow the procedure for the version of Tivoli Netcool/OMNIbus that your site uses.

Installing probes and gateways on Tivoli Netcool/OMNIbus V8.1

From Tivoli Netcool/OMNIbus V8.1 onwards, Tivoli Netcool/OMNIbus probes and gateways can be installed using the IBM Installation Manager. One of the key features of Installation Manager is that all platforms are shipped in a single ZIP file, which means that you do not have to select the platform that you require; Installation Manager does it for you.

Before you can install a probe or gateway, you must have installed and configured Installation Manager and Tivoli Netcool/OMNIbus. To install probes and gateways, you must make sure that the Core Tivoli Netcool/OMNIbus features **Probe Support** and **Gateway Support** respectively are installed.

Installing probes and gateways using the Command Line Tool

To install the probe or gateway using the Command Line Tool, run the following command:

```
installation_manager_location/eclipse/tools/imcl -c install
com.ibm.tivoli.omnibus.integrations.integration_name -repositories
repository_containing_required_integration -installationDirectory
location_of_netcool_omnibus_install_you_are_installing_into
```

Where *integration_name* specifies the name of the probe or gateway that you want to install.

You will be prompted to agree to the terms and conditions of the license as a prerequisite for installing the integration. If you have already reviewed the license and want to skip the manual acceptance, add the `-acceptLicense` option to the `install` command to silently agree to the license.

The following is an example command used to install the SNMP Probe:

```
imcl -c install com.ibm.tivoli.omnibus.integrations.nco-p-mttrapd -
repositories /home/my_home_dir/nco-p-mttrapd_im_package -
installationDirecory /opt/IBM/tivoli/netcool
```

Where /home/my_home_dir/nco-p-mttrapd_im_package contains the unzipped contents of the SNMP Probe Installation Manager package.

Note : The command line tool does not add the repository permanently to the Installation Manager instance. If you subsequently start the Installation Manager GUI, the repositories will not be present in the **Repositories** dialog box.

Uninstalling probes and gateways using the Command Line Tool

To uninstall the probe or gateway using the Command Line Tool, run the following command:

installation_manager_location/eclipse/tools/imcl uninstall com.ibm.tivoli.omnibus.integrations.*integration_name* -installationDirectory *location_of_netcool_omnibus_install_you_are_uninstalling_from*

Where *integration_name* specifies the name of the probe or gateway that you want to uninstall.

The following is an example command used to uninstall the SNMP Probe:

```
imcl uninstall com.ibm.tivoli.omnibus.integrations.nco-p-mttrapd -
installationDirecory /opt/IBM/tivoli/netcool
```

Installing probes and gateways using the GUI

To install the probe or gateway using the GUI, use the following steps:

1. Unzip the IM package that contains the probe or gateway into a directory of your choosing. A file called repository.config will appear after unzipping the IM package.
2. Start Installation Manager using the following command:

   installer_path/IBMIM

   Where *installer_path* is the path to the Installation Manager directory.
3. Perform the following menu actions to display the repository dialog box:

   Files > **Preferences** > **Repositories.**
4. Use the button **Add Repository** in the repository dialog box to point to the repository that contains the unzipped IM package that contains the probe or gateway. This is the repository that contains the repository.config file.
5. Click the **Install software packages** icon.
6. Select the name of the probe or gateway that you want to install.
7. Click **Next**.
8. Click **I accept** when the Licensing panel appears.
9. Highlight **IBM Tivoli Netcool OMNIbus** in the **Package Group Name** field.
10. Click **Next**.
11. Click **Next**.
12. Click **Install**.
13. When the **Install Packages** panel appears indicating that you have successfully installed the probe or gateway, click **Finish**.

Uninstalling probes and gateways using the GUI

To uninstall the probe or gateway, use the following steps:

1. Start Installation Manager using the following command:

   installer_path/IBMIM

   Where *installer_path* is the path to the Installation Manager directory.

2. Click the **Uninstall software packages** icon.
3. Select the name of the probe or gateway that you want to uninstall.
4. Click **Next**.
5. Click **Uninstall**.
6. When the **Install Packages** panel appears indicating that you have successfully uninstalled the probe or gateway, click **Finish**.

Setting environment variables

You might have to set some environment variables to define the gateway's working environment.

Before you run the gateway on a Windows operating system, ensure that the %PATH% environment variable contains the location of the `JVM.DLL` file. The default location of `JVM.DLL` is:

`%NCHOME%/platform/win32/jre_1.x.y/jre/bin/j9vm`

where *x* and *y* are determined by the version of Tivoli Netcool/OMNIbus that you are running.

Configuring communication details

To enable communication between the gateway and the ObjectServer, you must configure communication details for the ObjectServer and the gateway using the Tivoli Netcool/OMNIbus Server Editor (`nco_xigen`) and create an entry for the ObjectServer in the interfaces file (`$NCHOME/etc/omni.dat`).

If the ObjectServer is already configured and the gateway is to run from the same installation, you do not need to configure communication details for the ObjectServer.

On UNIX and Linux operating systems, use the following command to start the Server Editor:

`$NCHOME/omnibus/bin/nco_xigen`

On Windows operating systems, use the following command to start the Server Editor:

Start > Programs > NETCOOL Suite > System Utilities > Servers Editor

You must also add a gateway server entry to the interfaces file. You can do this using the Server Editor. Alternatively, on Unix and Linux operating systems you can edit the interfaces file and regenerate it using the `nco_igen` utility. The default gateway server name is `G_JDBC`.

If there is a firewall between the gateway and the ObjectServer, configure the ObjectServer to use a fixed port for IDUC and ensure that both the main ObjectServer port and the IDUC port are opened in the firewall. By default, the ObjectServer uses a random IDUC port.

For more information about using the Server Editor and the interfaces file, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

Configuring the database schema

Configuring the database schema involves running the appropriate audit or reporting mode SQL scripts for the target database. The scripts provided in the audit and reporting mode libraries are designed to cover general use cases. You will probably need to modify them to work with your database settings or to suit your particular requirements.

The audit and reporting mode libraries contain SQL scripts that create all the database schema objects required to store data processed by the gateway, including the tablespace, temporary tablespace, reporter, tables (`status`, `journal`, and `details`), indexes, and constraints.

Use either the audit mode scripts or the reporting mode scripts to create the database schema objects, as required. See "Audit mode and reporting mode" on page 5 for details of how the gateway operates in each mode.

Running the scripts

The installed scripts are located in subdirectories of the $OMNIHOME/gates/audit and $OMNIHOME/gates/reporting directories, named for the target database that they configure. For example, the scripts for IBM DB2 are located in the $OMNIHOME/gates/audit/db2 and $OMNIHOME/gates/reporting/db2 subdirectories.

Before running the scripts, you should consult your database documentation for instructions about how your database uses SQL scripts. You should also refer to the readme files in the script libraries and the comments in the script files for details of any limitations or constraints specific to individual scripts.

Migrating from an existing gateway

If you are using the Gateway for JDBC as a replacement for the Gateway for Oracle or the Gateway for ODBC, your existing target database configuration might not require any changes. In these cases, there is no need to run the database schema scripts.

Migrating from an existing target database should be done during a quiet period for the ObjectServer. This will ensure that the minimum number of events are lost for archiving while the gateway is not active.

Before migrating from an existing target database, perform a correct shut down of the old gateway to ensure that existing data is forwarded to the new gateway. The new gateway will perform a resynchronization of events from the ObjectServer, so problems with the old gateway shut down should not result in loss of data. However, resynchronization may result in duplicate event data and subsequent error messages if constraint violations in the target database are triggered.

Persistent state

The Gateway for JDBC is designed to facilitate migration from an existing database gateway installation. As it is not practical to migrate the persistent state from an existing database gateway, the Gateway for JDBC has no persistent state by default, unlike the ODBC and Oracle gateways. The Gateway for JDBC seeds its state from the target database before doing further resynchronization. This should prevent existing INSERT alerts from being inserted on resynchronization and instead the alerts should be correctly marked as UPDATE.

Alerts are pulled from the target database that are considered open. In reporting mode, open means the DELETEDAT field is null. This initial bidirectional resynchronization only happens when the Gateway for JDBC determines it has no state. The gateway determines it has no state when the alert count in the cache is 0.

Subsequent resynchronization should not perform a bidirectional resynchronization once there are alerts in the cache.

To replicate the existing Oracle or ODBC gateway functionality of not performing bidirectional resynchronization, you must set the Gateway for JDBC property **Gate.Jdbc.ResyncMode** to "UNI". For further information about this property see “Properties and command line options” on page 28.

Bidirectional resynchronization

The first time you run the gateway, it will detect that there are no alerts in its cache and it will perform a bidirectional resynchronization. This will detect any events in the target database that were still open when the old gateway was last run, and that have subsequently been closed and deleted from the ObjectServer. This enables event deletes that were lost while the gateway was being migrated to be recovered to the target database. On subsequent runs, when there are alerts in its cache, the gateway will perform a unidirectional resynchronization on startup.

Note : Bidirectional resynchronization results in full table scans on the target database status table. The amount of time required for the resynchronization to finish is proportional to the size of the database table. A large amount of event history will result in a long resynchronization.

Configuring the database connection

Configuring the database connection involves configuring the JDBC driver and specifying values for the connection-related properties.

JDBC drivers

You must obtain the JDBC driver for the target database from the database vendor and install it according to the vendor's instructions. The drivers are usually provided as Java archives (`.jar`).

You must copy the JDBC driver `.jar` file to the following directory:

`$OMNIHOME/gates/java`

Database connection properties

To enable the gateway to communicate with the target database, you must specify values for the following properties:

- **`Gate.Jdbc.Connections`**

  This property specifies the number of connections that the gateway makes to the target database. Increasing the number of connections increases the level of parallelism available to the gateway and potentially increases performance. Start with low values and increase as needed to find the desired performance level.

- **`Gate.Jdbc.Driver`**

  This property specifies the JDBC driver. If the **`Gate.Jdbc.Driver`** property is left empty (the default), then the internal null JDBC driver is used. This internal JDBC driver is used for testing purposes to allow the generic portions of the gateway to be run without requiring a database connection to be configured. The null JDBC driver also simulates error insertion and row updating in order to test the error handling code of the gateway. The null JDBC driver is also useful for enabling users to familiarize themselves with the gateway operations before committing updates to their target database. It can be run in parallel to an existing database gateway, to allow this familiarization to be done without downtime of the existing gateway.

  Running the gateway in debug log level allows you to see the logging produced by the gateway and to inspect the resulting SQL that would have been sent to their target database.

  Note : The JDBC gateway persistent state should be deleted after running in test mode to delete alert history used for de-duplication.

- **`Gate.Jdbc.Url`**

  This property specifies the URL of the target database.

- **`Gate.Jdbc.Username`**

  This property specifies the user name for the target database.

- **`Gate.Jdbc.Password`**

  This property specifies the password for the target database.

The following table lists example values for the **`Gate.Jdbc.Driver`** and **`Gate.Jdbc.Url`** properties for use with each database. Consult your driver documentation for more information about setting up database connections. Default values may be different depending on your setup.

Table 6. Example JDBC property values	
DB2 LUW	
`Gate.Jdbc.Driver`	`com.ibm.db2.jcc.DB2Driver`

Table 6. Example JDBC property values (continued)	
Gate.Jdbc.Url	jdbc:db2://*host_name*:*port*/*db_name* Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. For example: jdbc:db2://server.example.ibm.com:9999/REPORTER
DB2 z/OS	
Gate.Jdbc.Driver	com.ibm.db2.jcc.DB2Driver
Gate.Jdbc.Url	jdbc:db2://*host_name*:*port*/*db_name* Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. For example: jdbc:db2://server.example.ibm.com:9999/REPORTER
Informix	
Gate.Jdbc.Driver	com.informix.jdbc.IfxDriver
Gate.Jdbc.Url	jdbc:informix-sqli://*host_name*:*port*/*db_name*:INFORMIXSERVER=*server_name* Where *host_name* is the name of the database host machine, *port* is the port number, *db_name* is the name of the database, and *server_name* is the same as the *host_name*. For example: jdbc:informix-sqli://server.example.ibm.com:1433/REPORTER:INFORMIXSERVER=server.example.ibm.com
Microsoft SQL Server	
Gate.Jdbc.Driver	com.microsoft.sqlserver.jdbc.SQLServerDriver
Gate.Jdbc.Url	jdbc:sqlserver://*host_name*:*port*;databaseName=*db_name*;instanceName=*instance_name*;encrypt=false Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database, ` *instance_name* specifies the instance name of the database and *encrypt* specifies if encryption is enabled for the database. Both *instance_name* and *encrypt* are optional and may be omitted if default values are used to connect to the database. For more details on additional parameters that can be specified, please refer to the Microsoft JDBC Driver for SQL Server website.. The default *port* is 1433. For example: jdbc:sqlserver://server.example.ibm.com:1433;databaseName=REPORTER;instanceName=MSSQLSERVER;encrypt=false

Table 6. Example JDBC property values (continued)	
MySQL	
Gate.Jdbc.Driver	`com.mysql.cj.jdbc.Driver`

Gate.Jdbc.Url	`jdbc:mysql://host_name[,failover_host]:port/db_name[?param1=value1&param2=value2]`
	Where *host_name* is the name of the database host machine, *failover_host* is the name of the optional failover host, *port* is the port number, *db_name* is the name of the database, and *param1* and *param2* are optional parameters. The default *port* is 3306. For example:
	`jdbc:mysql://server.example.ibm.com:3306/alerts`

Oracle	
Gate.Jdbc.Driver	`oracle.jdbc.driver.OracleDriver`

Gate.Jdbc.Url	`jdbc:oracle:thin:@host_name:port:db_name`
	Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. The default *port* is 1521. For example:
	`jdbc:oracle:thin:@server.example.ibm.com:1521:REPORTER`

Sybase	
Gate.Jdbc.Driver	`com.sybase.jdbc4.jdbc.SybDriver`

Gate.Jdbc.Url	`jdbc:sybase:Tds:host_name:port/db_name[?property=value;]`
	Where *host_name* is the name of the database host machine, *port* is the port number, *db_name* is the name of the database, and *property* is an optional parameter. For example:
	`jdbc:sybase:Tds:server.example.ibm.com:1521/REPORTER`

Netezza	
Gate.Jdbc.Driver	`org.netezza.Driver`

Gate.Jdbc.Url	`jdbc:netezza://host_name:port/db_name`
	Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. For example:
	`jdbc:netezza://server.example.ibm.com:5480/ALERTS`

Integrating with an Oracle database

This topic describes how to create and configure an Oracle database to integrate it with Netcool/OMNIbus.

Integrating the JDBC Gateway with Oracle requires the following steps:

Note : The steps in this procedure should be run by the user `oracle`, which is created during the Oracle installation.

Step 1: Setting up the environment variables

Update the `.bash_profile` of the user `oracle` using the following command:

```
export ORACLE_SID=ortestdb
```

Where *ortestdb* is your database instance name.

`ORACLE_HOME` and `ORACLE_SID` have been configured for user oracle.

To check, run the following command:

```
"env | grep ORACLE"
```

`ORACLE_SID` holds the name of the database instance. To create a new instance, or to use an existing instance, the environment variable must be updated first.

Step 2: Preparing an Oracle parameters file

Update the following parameters in the $ORACLE_HOME/dbs/init.ora file.

```
db_name='ortestdb'
memory_target=1G
processes = 150
audit_file_dest='/opt/oracle_BASE/test1/admin/orcl/adump'
audit_trail ='db'
db_block_size=8192
db_domain='my.ibm.com'
db_create_file_dest='/opt/oracle_BASE/test1/oradata'
db_create_online_log_dest_1='/opt/oracle_BASE/test1/u02/oradata'
db_create_online_log_dest_2='/opt/oracle_BASE/test1/u03/oradata'
db_recovery_file_dest='/opt/oracle_BASE/test1/fast_recovery_area'
db_recovery_file_dest_size=2G
diagnostic_dest='/opt/oracle_BASE/test1'
dispatchers='(PROTOCOL=TCP) (SERVICE=ORCLXDB)'
open_cursors=300
remote_login_passwordfile='EXCLUSIVE'
undo_tablespace='UNDOTBSP'
# You may want to ensure that control files are created on separate physical
# devices
control_files = (/opt/oracle_BASE/test1/ctl/u01/prod/control01.ctl,
                 /opt/oracle_BASE/test1/ctl/u01/prod/control02.ctl,
                 /opt/oracle_BASE/test1/ctl/u01/prod/control03.ctl)
compatible ='12.0.0'
OS_AUTHENT_PREFIX=""
```

Note:

db_name is the name of the database instance held by ORACLE_SID.

db_domain is the domain of the server.

undo_tablespace must be set to 'UNDOTBSP'

/opt/oracle_BASE and the sub-directories (before the file names) shown in the sample init.ora must be staged before running the **CREATE SPFILE** command.

It is better to create a bash script for directory creation.

For staging directories, perform the following steps:

1. As root, set full permissions (apply chmod 777) to /opt/oracle_BASE.
2. Use user oracle to create the sub-directories.

You can use a name other than oracle_BASE, but do not use $ORACLE_HOME as the parent directory to the storage location of the database instance files, this is to ease resource management of the database instances. For example, if **Step 3** hit errors, you can remove all files under /opt/oracle_BASE/test1 before rerunning **Step 3**.

The **CREATE DATABASE** command will look for init<ORACLE_SID>.ora. If the **CREATE DATABASE <ORACLE_SID>** command fails reporting: file not found, rename it init.ora to init<ORACLE_SID>.ora (for example initortestdb.ora).

Step 3: Creating a binary parameter file

Run the following command:

```
SQL> CREATE SPFILE FROM PFILE;
```

Verify the creation of the $ORACLE_HOME/dbs/spfile<ORACLE_SID>.ora file.

Note : Do not use a text editor to amend the spfile<ORACLE_SID>.ora file. Consult the Oracle documentation for the appropriate editing procedure.

Step 4: Creating a database instance

To create a database instance, use the following steps:

1. Run the following command to connect to an idle instance:

   ```
   [oracle@klxv0104 bin]$ ./sqlplus / as sysdba
   ```

2. Run the following command:

   ```
   SQL> CREATE SPFILE FROM PFILE;
   ```

   This creates the following file: $ORACLE_HOME/dbs/spfileortestdb.ora

3. Run the following command to start the Oracle instance:

   ```
   SQL> STARTUP NOMOUNT;
   ```

4. Run the following command to create the Oracle database:

   ```
   SQL> CREATE DATABASE ortestdb USER SYS IDENTIFIED BY sys_pass USER SYSTEM IDENTIFIED BY
   system_pass EXTENT MANAGEMENT LOCAL DEFAULT TEMPORARY TABLESPACE temp UNDO TABLESPACE
   UNDOTBSP DEFAULT TABLESPACE users;
   ```

   Note :

The command example in the Oracle document uses a different name for undo table, the value in the command shown here uses UNDOTBSP, which agrees with the `undo_tablespace` parameter in the `init.ora` file. (See the note in "Step 2: Preparing an Oracle parameters file" on page 16).

Verify the database instance file location for the files created.

If the **CREATE DATABASE** command failed at some point, before rerunning the step, perform the following steps:

a. Exit the NOMOUNT state: SHUTDOWN IMMEDIATE

b. Clean up the `db_instance` folder specified in `init.ora` (see "Step 2: Preparing an Oracle parameters file" on page 16).

For the following steps, pass in the designated paths of your setup as the command argument. The path must be staged beforehand.

5. Run the following command to create the tablespace `apps_tbs`.

```
SQL> CREATE TABLESPACE apps_tbs LOGGING DATAFILE '/opt/oracle_BASE/test1/oradata/ORTESTDB/
datafile/apps01.dbf' SIZE 500M REUSE AUTOEXTEND ON NEXT 1280K MAXSIZE UNLIMITED EXTENT
MANAGEMENT LOCAL;
```

6. Run the following command to create the tablespace `indx_tbs`.:

```
SQL> CREATE TABLESPACE indx_tbs LOGGING DATAFILE '/opt/oracle_BASE/test1/oradata/ORTESTDB/
datafile/indx01.dbf' SIZE 100M REUSE AUTOEXTEND ON NEXT 1280K MAXSIZE UNLIMITED EXTENT
MANAGEMENT LOCAL;
```

Step 5: Building the Data Dictionary Views

Before starting, the database instance created previously must be in OPEN state.

To build data dictionary views, perform the following steps:

1. Log in as SYSDBA.
2. Run the following command to start the database in OPEN state:

```
SQL> STARTUP OPEN;
```

3. Run the following command to check the instance status:

```
SQL> select INSTANCE_NAME, STATUS from v$instance;
```

4. As SYSDBA, run the following scripts in the order shown:

```
@?/rdbms/admin/catalog.sql
@?/rdbms/admin/catproc.sql
@?/rdbms/admin/utlrp.sql
```

5. Connect as SYSTEM at the SQL prompt, log in with the SYSTEM password specified in the **CREATE DATABASE** command:

```
SQL> connect system
```

6. Run the following command:

SQL> @?/sqlplus/admin/pupbld.sql

Note : You can safely ignore the DROP errors displayed while the SQL is executing.

Step 6: Preparing a table for the Netcool integration

Preparing REPORTER mode:

Use the scripts bundled in the `nco-g-jdbc-reporting-script` package.

Run the following SQL at the SQL prompt:

```
SQL>@<path>/oracle.reporting.sql
```

For example:

```
SQL> @/tmp/netcool_ora_sql_scripts/reporting_scripts/oracle.reporting.sql
```

Check that the table and the tablespace were created correctly:

```
SQL> select owner, table_name from all_tables where tablespace_name='REPORTER';
```

Preparing AUDIT mode:

Use the scripts bundled in `nco-g-jdbc-audit-script` package.

1. Modify the `orainstall` script:
   a. Change the availability check in `create_tables.audit.sql`:

      ```
      if [ ! -f $OMNIHOME/gates/nco_g_oracle/sql_scripts/create_tables.audit.sql ];
      ```

      to

      ```
      if [ ! -f ./create_tables.audit.sql ];
      ```

   b. Change the location of the `create_tables.audit.sql` file.

      Comment out:

      ```
      #cat $OMNIHOME/gates/nco_g_oracle/sql_scripts/$INPUT_SQL_FILE | sed -e "s/__TABLESPACE__/
      $TABLESPACE/g" -e "s/__STATUS__/$STATUSTAB/g" -e "s/__JOURNAL__/$JOURNALTAB/g" -e "s/
      __DETAILS__/$DETAILSTAB/g" -e "s/__STORAGE__/$STORAGEOPT/g" >> /tmp/nco_oracle.sql
      ```

      Add the following code:

      ```
      cat ./$INPUT_SQL_FILE | sed -e "s/__TABLESPACE__/$TABLESPACE/g" -e "s/__STATUS__/
      $STATUSTAB/g" -e "s/__JOURNAL__/$JOURNALTAB/g" -e "s/__DETAILS__/$DETAILSTAB/g" -e "s/
      __STORAGE__/$STORAGEOPT/g" >> /tmp/nco_oracle.sql
      ```

2. Run `orainstall` to generate the audit script from the template `create_tables.audit.sql` file.

   Note : Run `orainstall` from the same directory storing the audit SQL scripts.

   ```
   ./orainstall
   ```

   This creates the `/tmp/nco_oracle.sql` script.
3. Run the `nco_oracle.sql` script.

   ```
   SQL> @/tmp/netcool_ora_sql_scripts/audit_scripts/nco_oracle.sql
   ```

4. Verify that the tables were created by `nco_oracle.sql`:

   ```
   SQL> select TABLE_NAME from all_tables where TABLESPACE_NAME='SYSTEM' and
   (TABLE_NAME='STATUS' or TABLE_NAME='JOURNAL' or TABLE_NAME='DETAILS');
   ```

Step 7: Creating a password file

This step is only required if users other than the one specified in the table creation scripts in “Step 6: Preparing a table for the Netcool integration” on page 18 were specified.

To create the password file, perform the following steps:

1. Check the password file:

```
SQL> select username, from v$pwfile_users;
```

Note: The default directory where the password file is located is: $ORACLE_HOME/dbs/orapw $ORACLE_SID

2. If the password file is empty or does not exist, create it now:

```
cd $ORACLE_HOME/bin
./orapwd FILE=/opt/oracle/product/19c/dbhome_1 /dbs/orapwortestdb ENTRIES=30
PASSWORD=<password of SYS>
```

Note:

FILE = **$ORACLE_HOME**/dbs/orapw$ORACLE_SID

PASSWORD = SYS password should comply to password requirements (for example, username should not be a substring, and contain at least one numeric and one special character)

If not specified, the password prompt will appear for input.

ENTRIES = the maximum number of distinct SYSDBA, SYSOPER, SYSASM, SYSKM, SYSDG or SYSBACKUP users that can be stored in the password file.

Important:

The password should be set to PASSWORD or the password prompt will override the password originally specified in the **CREATE DATABASE** command.

The password update can be verified using the attempt to connect by the following command:

```
SQL> connect sys as sysdba
```

To update the SYS password, use following command:

```
SQL> ALTER USER sys IDENTIFIED BY "<password>";
```

To use the SYS user to connect to the Oracle database, check whether SYS is in the password file using the following command:

```
SQL> select * from v$pwfile_users;
```

3. Grant the appropriate role to the user.
   a. Create the Oracle user by running the following command:

```
SQL> CREATE USER <user> IDENTIFIED BY <password>;
```

   For example:

```
SQL> CREATE USER jdbcgwoper IDENTIFIED BY jdbcgwoper_pass;
```

   b. Grant a role to user by running the following command:

```
SQL> grant <role> to <user>;
```

   For example:

```
SQL> grant sysoper to jdbcgwoper;
```

   c. Verify the user's role by running the following command:

```
SQL> select SYSDBA, SYSOPER, SYSASM, SYSBACKUP, SYSDG, SYSKM from v$pwfile_users where
USERNAME='SYS';
```

Step 8: Updating the `tnsnames.ora` file

You may get the following error when connecting JDBC to Oracle database:

```
2020-05-07T23:27:36: Error: E-GJA-000-000: [ngjava]: G_JDBC: Thread-7: Listener refused the
connection with the following error:
ORA-12504, TNS:listener was not given the SID in CONNECT_DATA

2020-05-07T23:27:36: Error: E-GJA-000-000: [ngjava]: G_JDBC: Thread-7: java.sql.SQLException:
Listener refused the connection with the following error:
ORA-12504, TNS:listener was not given the SID in CONNECT_DATA
```

If so, use the following steps:

1. Update `$ORACLE_HOME/network/admin/tnsnames.ora` to include the `SID` in the `CONNECT_DATA` definition:

```
ORCLCDB =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = <hostname>)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ORCLCDB)
      (SID = ortestdb)
    )
  )

LISTENER_ORCLCDB =
  (ADDRESS = (PROTOCOL = TCP)(HOST = <hostname>)(PORT = 1521))
```

2. Start the listener using the following commands:

```
$ORACLE_HOME/bin/lsnrctl start
$ORACLE_HOME/bin/lsnrctl status
```

Integrating with a Microsoft SQL Server database

This topic describes how to create and configure a Microsoft SQL Server database to integrate it with Netcool/OMNIbus.

The JDBC Gateway requires the Microsoft JDBC Driver for SQL Server to integrate with Microsoft SQL Server. The JDBC driver can be downloaded at the Microsoft JDBC Driver for SQL Server website.

The JDBC Gateway can integrate with the Microsoft SQL Server using either a secure or a non-secure connection. However, the JDBC Gateway will always authenticate using a secure connection. When the JDBC Gateway begins to connect with Microsoft SQL Server, a secure (TLS) connection is initiated and the expected process of a TLS handshake occurs. Once the handshake is completed successfully, the JDBC Gateway authenticates with Microsoft SQL Server using the provided user credentials. Once the authentication is completed successfully, subsequent communication with Microsoft SQL Server will be performed using either a secure or non-secure connection as per the gateway configuration.

To integrating the JDBC Gateway with Microsoft SQL Server, use the following steps:

- "Step 1: Installing the Microsoft JDBC Driver" on page 21
- "Step 2: Preparing tables for Netcool integration" on page 22
- "Step 3: Configuring the gateway" on page 22

Step 1: Installing the Microsoft JDBC Driver

For OMNIbus installations using JRE 1.7, install the JRE 1.7 version of the JDBC driver. Otherwise for OMNIbus installations using JRE 1.8, install the JRE 1.8 version of the JDBC driver instead. Refer to the Microsoft JDBC Driver for SQL Server Support Matrix website for more information on the currently supported driver versions.

Copy the Microsoft JDBC Driver for example, `mssql-jdbc-8.2.2.jre8.jar` to the `$OMNIHOME/gates/java` directory. Alternatively, the JDBC driver may be copied to the directory defined in

Gate.Java.ClassPath in the properties file. Ensure that only one copy of the JDBC Driver is installed in the directory.

Step 2: Preparing tables for Netcool integration

Preparing REPORTER mode:

Use the scripts bundled in the `nco-g-jdbc-reporting-scripts` package under the directory `mssql`.

At the command prompt, navigate to the `mssql` directory and run the following command to create the REPORTER tables in your database:

```
> sqlcmd -S sqlserver -U netcool -P password -i mssql.reporting.sql
```

Where *`sqlserver`* is the database name, *`netcool`* is the username, *`password`* is the password for the `netcool` user and *`mssql.reporting.sql`* is the database script to execute.

You may check that the tables have been created using the following command:

```
> sqlcmd -S sqlserver -U netcool -P password
1> select table_name from REPORTER.information_schema.tables;
2> go
```

Preparing AUDIT mode:

Use the scripts bundled in the `nco-g-jdbc-audit-scripts` package under the directory `mssql`.

In the command prompt, navigate to the `mssql` directory and run the following command to create the AUDIT tables in your database:

```
> sqlcmd -S sqlserver -U netcool -P password -i mssql.sql
```

Where *`sqlserver`* is the database name, *`netcool`* is the username, *`password`* is the password for the *`netcool`* user and `mssql.sql` is the database script to execute.

You can check that the tables have been created using the following command:

```
>
1> select table_name from information_schema.tables;
2> go
```

Step 3: Configuring the gateway

Note: Microsoft SQL Server 2016 and higher versions strictly require clients to securely connect using TLS v1.2 or higher. To ensure that the JDBC Gateway always initiates a TLS connection using TLS v1.2 or higher, the following argument must be specified in the gateway properties file:

```
Gate.Java.Arguments: '-Dcom.ibm.jsse2.overrideDefaultTLS=true'
```

Specify the correct mode to be used namely, AUDIT or REPORTING:

```
Gate.Jdbc.Mode: 'REPORTING' # STRING (JDBC gateway mode (AUDIT|REPORTING))
```

Specify the JDBC connection properties:

```
Gate.Jdbc.Driver: 'com.microsoft.sqlserver.jdbc.SQLServerDriver' # STRING (JDBC Driver)
Gate.Jdbc.Url: 'jdbc:sqlserver://<hostname or IP address of Microsoft SQL
Server>:1433;databaseName=REPORTER' # STRING (JDBC connection URL)
Gate.Jdbc.Username: 'netcool' # STRING (JDBC username)
Gate.Jdbc.Password: 'password' # STRING (JDBC password)
```

In the above example, the gateway is configured to connect to the REPORTER database using the user `netcool`.

Additional parameters can be specified as needed. In the following example the connection URL specifies the instance name and that a non-secure connection should be used:

```
Gate.Jdbc.Url: 'jdbc:sqlserver://<hostname or IP address of Microsoft SQL
Server>:1433;databaseName=REPORTER;instanceName=MSSQLSERVER;encrypt=false' # STRING (JDBC
connection URL)
```

If a secure connection is required to be used by the gateway, the following should be configured:

```
Gate.Java.Arguments: '-Dcom.ibm.jsse2.overrideDefaultTLS=true -Djavax.net.ssl.trustStore=<path
to truststore file> -Djavax.net.ssl.trustStorePassword=<truststore file password>'
Gate.Jdbc.Url: 'jdbc:sqlserver://<hostname or IP address of Microsoft SQL
Server>:1433;databaseName=REPORTER;instanceName=MSSQLSERVER;encrypt=true' # STRING (JDBC
connection URL)
```

Integrating with a DB2 database

This topic describes how to create and configure a DB2 database to integrate it with Netcool/OMNIbus.

To integrating the JDBC Gateway with Microsoft SQL Server, use the following steps:

- "Step 1: Setting up the environment variables" on page 23
- "Step 2: Preparing tables for Netcool integration" on page 23
- "Step 3: Configuring DB2 parameters" on page 24

Step 1: Setting up the environment variables

Install DB2 software. Follow steps provided on the product user guide.

Step 2: Preparing tables for Netcool integration

Preparing REPORTER mode:

Use the scripts bundled in the `nco-g-jdbc-reporting-scripts` package.

1. Create database REPORTER:

   For example, run the following command:

   ```
   db2 create database reporter
   ```

2. Modify the `db2.reporting.sql` script:

   a. Uncomment the `CREATE DATABASE` line.

   b. Set the default user name and password to match the DB2 installation:

   ```
   CREATE DATABASE reporter @.
              CONNECT TO reporter USER db2inst1 USING db2inst1
   ```

   c. Uncomment the following lines, so that any association journal and details rows are deleted from the database when the corresponding alert are deleted.

   For example:

   ```
   -- Uncomment the line below to enable foreign keys
   -- This helps pruning by only requiring the alert to be
   -- deleted from the status table
   , CONSTRAINT eventref FOREIGN KEY (SERVERNAME, SERVERSERIAL)        REFERENCES
   REPORTER_STATUS(SERVERNAME, SERVERSERIAL) ON DELETE CASCADE
   ```

   Note : This SQL appears twice in the SQL file: once in the details table definition and once in the journal table definition. Uncomment both instances.

3. Run the SQL file against the DB2 database by running the following command as the db2inst1 system user:

```
$ db2 -td@ -vf db2.reporting.sql
```

Preparing AUDIT mode:

Use the scripts bundled in the `nco-g-jdbc-audit-scripts` package.

Run the SQL files in order against the DB2 database by running the following commands as the `db2inst1` system user:

```
$ db2 -td@ -vf db2_status.sql
$ db2 -td@ -vf db2_journal.sql
$ db2 -td@ -vf db2_details.sql
```

Step 3: Configuring DB2 parameters

Run `db2stop` before updating DB2 parameters to prevent issues caused by the security parameters being incompatible with the original settings that `db2start` was running with.

Updates in the DB2 parameters will take effect after `db2start` is run successfully.

Configure DB2 parameters for plaintext user and password authentication

```
db2 update dbm cfg using SRVCON_GSSPLUGIN_LIST NULL
db2 update dbm cfg using AUTHENTICATION SERVER
```

Configure DB2 parameters for Kerberos Authentication

```
db2set DB2ENVLIST=KRB5_KTNAME
db2 update dbm cfg using SRVCON_GSSPLUGIN_LIST IBMkrb5
db2 update dbm cfg using AUTHENTICATION KERBEROS
```

Note: KRB5_KTNAME is an environment variable pointing to the server principal's `keytab` file.

Configure DB2 parameters for SSL connection

```
db2 update dbm cfg using SSL_SVR_KEYDB <db2_ssl_keydb.kdb>
db2 update dbm cfg using SSL_SVR_STASH <db2_ssl_keydb.sth>
db2 update dbm cfg using SSL_SVR_LABEL IBM_CA_signed
db2 update dbm cfg using SSL_SVCENAME <port>
```

Integrating with a MySQL Server database

This topic describes how to create and configure a MySQL Server database to integrate it with Netcool/OMNIbus.

To integrating the JDBC Gateway with MySQL Server, use the following steps:

- "Step 1: Installing the MySQL JDBC Driver" on page 24
- "Step 2: Preparing tables for Netcool integration" on page 24
- "Step 3: Configuring the gateway" on page 25

Step 1: Installing the MySQL JDBC Driver

Refer to the MySQL website for more information on the currently supported driver versions.

Copy the MySQL JDBC Driver for example, `mysql-connector-java-8.0.20.jar` to the `$OMNIHOME/gates/java` directory.

Step 2: Preparing tables for Netcool integration

Preparing AUDIT mode:

Use the scripts bundled in the `nco-g-jdbc-audit-scripts` package under the directory `mysql`.

In the command prompt, navigate to the `mysql` directory and run the following command to create the AUDIT tables in your database:

```
> run ./mysqlinstall
```

Run `mysqlinstall` to install the database tables.

You can check that the tables have been created using the following command:

```
Login to mysql -u <user_name> -p <password>
 mysql> show databases;
 mysql> use alerts;
 mysql> show tables;
```

Note : Mysql is only supported in AUDIT Mode.

Step 3: Configuring the gateway

Specify the correct mode to be used (namely, AUDIT):

```
Gate.Jdbc.Mode: 'AUDIT' # STRING (JDBC gateway mode (AUDIT|REPORTING))
```

Specify the JDBC connection properties:

```
Gate.Jdbc.Driver: 'com.mysql.cj.jdbc.Driver' # STRING (JDBC Driver)
Gate.Jdbc.Url: 'jdbc:mysql://<hostname or IP address of the MySQL Server>:3306/alerts' # STRING
(JDBC connection URL)
Gate.Jdbc.Username: 'netcool' # STRING (JDBC username)
Gate.Jdbc.Password: 'password' # STRING (JDBC password)
```

In the above example, the gateway is configured to connect to the ALERTS database using the user `netcool`.

If a secure connection is required to be used by the gateway, the following should be configured:

```
Gate.Java.Arguments: '-Djavax.net.ssl.trustStore=<path to truststore file> -
Djavax.net.ssl.trustStorePassword=<truststore file password>' # STRING (JDBC connection URL)
```

Configuring SSL connections

If you are using an HTTPS/SSL connection between the gateway and the target database, you must create a truststore to store the JDBC digital certificate and point the gateway to the location of the truststore.

You can generate the truststore file using the Java keytool utility. The keytool utility is located in the following directory:

`$OMNIHOME/platform/`*`arch`*`/`*`jre_directory`*`/jre/bin/`

Where *arch* is the operating system you are running and *jre_directory* is the installation directory of your Java Runtime Environment (JRE).

Use the following steps to enable SSL connections between the gateway and JDBC:

1. Obtain the JDBC certificate files.

   Note : You must obtain the CA root certificate and any intermediate CA certificates that your JDBC configuration uses.
2. Use the Java keytool utility to create a truststore and password.
3. Add the JDBC SSL certificate to the truststore.

   For example, run the following keytool command from the `$NCHOME/platform/`*`arch`*`/jre_`*`version`*`/bin` directory to import the certificate and create the keystore:

```
keytool -import -trustcacerts -alias CAROOTCERT -file PATH_TO_CERTIFICATE
CA.cer -keystore $NCHOME/etc/security/jdbccerts
```

Note : You will be prompted to create a truststore password.

4. Update the **Gate.Jdbc.Url** property to include the URL location.

   For example:

   ```
   'jdbc:db2://targethost:50004/REPORTER:sslConnection=true;
   sslTrustStoreLocation=/netcool/etc/cert/cacerts;'
   ```

   Note : The *targethost* variable must match the hostname specified in the JDBC server certificate.

5. Update the **Gate.Java.Arguments** property to include the following arguments:

   - `-Djavax.net.ssl.trustStore=`*truststore_path*

     Where *truststore_path* is the full path and name of the truststore file.

   - `-Djavax.net.ssl.trustStorePassword=`*truststore_password*

   For example:

   ```
   Gate.Java.Arguments : '-Djavax.net.ssl.trustStore=/opt/IBM/tivoli/netcool/
   platform/arch/jre_1.6.7/jre/lib/security/jdbccerts -
   Djavax.net.ssl.trustStorePassword=changeit'
   ```

   Note : The **Gate.Java.Arguments** property can be encrypted to hide the trust store password from casual inspection.

Configuring the gateway

After installing the gateway, you must configure it to work with your operating environment.

The gateway installation package contains the configuration files required to run the gateway. The default configuration files configure the gateway for operation in reporting mode.

The following table lists the configuration files installed with the gateway installation package.

Table 7. Gateway configuration files

Configuration file	Description
`$OMNIHOME/etc/G_JDBC.props`	The default properties file used by the gateway. This file configures the gateway for operation in reporting mode.
`$OMNIHOME/gates/jdbc/audit.G_JDBC.props`	The properties file used to configure the gateway for operation in audit mode.
`$OMNIHOME/gates/jdbc/audit.jdbc.map`	The map definition file used to configure the gateway for operation in audit mode.
`$OMNIHOME/gates/jdbc/G_JDBC.props`	A copy of the default properties file used by the gateway. This file configures the gateway for operation in reporting mode.
`$OMNIHOME/gates/jdbc/jdbc.map`	The default map definition file used by the gateway. This file configures the gateway for operation in reporting mode.
`$OMNIHOME/gates/jdbc/jdbc.rdrwtr.tblrep.def`	The default table replication definition file used by the gateway.

Table 7. Gateway configuration files (continued)	
Configuration file	**Description**
`$OMNIHOME/gates/jdbc/jdbc.startup.cmd`	The default startup command file used by the gateway.
`$OMNIHOME/gates/jdbc/reporting.G_JDBC.props`	The properties file used to configure the gateway for operation in reporting mode. It is identical to the default properties file `G_JDBC.props`.
`$OMNIHOME/gates/jdbc/reporting.jdbc.map`	The map definition file used to configure the gateway for operation in reporting mode. It is identical to the default map definition file `jdbc.map`.
`$OMNIHOME/gates/jdbc/jdbc_conn.properties`	The file contains sample JDBC parameters for Oracle and DB2 connections.
`$OMNIHOME/gates/jdbc/jdbc_javasys.properties`	The file contains sample Java system properties for Oracle and DB2 connections.
`$OMNIHOME/gates/jdbc/jaas.conf`	The file contains sample JAAS client configuration used in Kerberos-authenticated database connections.

The following topics contain more information about configuring the gateway.

- "Properties file" on page 27
- "Properties and command line options" on page 28
- "Map definition file" on page 46
- "Startup command file" on page 46
- "AfterIDUC and Filter functions" on page 49
- "Using a partitioning field" on page 50
- "Filtering resynchronization data" on page 50
- "Message log file" on page 51
- "FIPS mode and encryption" on page 52

For more information about using gateway configuration files, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Properties file

The properties file is a text file that contains a set of gateway-specific and generic Tivoli Netcool/OMNIbus properties and their corresponding values. You can edit the properties file to suit your operating environment.

The default properties file installed with the gateway, `$OMNIHOME/etc/G_JDBC.props`, configures the gateway to operate in reporting mode.

To configure the gateway to operate in audit mode, use the following steps:

1. Copy `$OMNIHOME/gates/jdbc/audit.G_JDBC.props` to `$OMNIHOME/etc/G_JDBC.props`.
2. Copy `$OMNIHOME/gates/jdbc/audit.jdbc.map` to `$OMNIHOME/gates/jdbc/jdbc.map`.

Properties and command line options

You use properties to define the operational environment of the gateway. You can override the default property values by editing the properties file or using the property's command line options.

The following tables describe the main properties required to configure the Gateway for JDBC. For information about additional generic Tivoli Netcool/OMNIbus gateway properties and command line options, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

The following sections describe the properties used to configure the gateway:

Common Tivoli Netcool/OMNIbus properties

The Common Tivoli Netcool/OMNIbus lists the available common properties.

Table 8. Common Tivoli Netcool/OMNIbus properties

Property name	Command line option	Description
ConfigCryptoAlg *string*	`-configcryptoalg` *string*	Use this property to specify the encryption algorithm that the gateway uses. Use this property in conjunction with the **ConfigKeyFile** property and the `nco_aes_crypt` utility supplied with Tivoli Netcool/OMNIbus. The default is AES.
ConfigKeyFile *string*	`-configkeyfile` *string*	Use this property to specify the encryption key used with the encrypted data. Use this property in conjunction with the **ConfigCryptoAlg** property and the `nco_aes_crypt` utility supplied with Tivoli Netcool/OMNIbus. The default is `""`.
Connections *integer*	`-connections` *integer*	Use this property to specify the maximum number of client connections that can be made to the gateway server. The default is `30`.
MaxLogFileSize *integer*	`-maxlogfilesize` *integer*	Use this property to specify the size (in bytes) that the gateway allocates for the log file. When the log file reaches this size, the gateway renames the log file by appending the suffix `.old` and creates a new log file. The default is `1024`.

Table 8. Common Tivoli Netcool/OMNIbus properties (continued)

Property name	Command line option	Description
MessageLevel *string*	`-messagelevel` *string*	Use this property to specify the reporting level of the log file messages. The default is `warn`.
MessageLog *string*	`-messagelog` *string*	Use this property to specify the location of the message log file. The default is `'$OMNIHOME/log/G_JDBC.log'`.
Name *string*	`-name` *string*	Use this property to specify the name of the current gateway instance. If you want to run multiple gateways on one machine, you must use a different name for each instance. The default is `'G_JDBC'`.
Props.CheckNames *boolean*	No command line equivalent.	Use this property to instruct the gateway to shut down if any property in the properties file is set to an invalid value. The default is TRUE.
Props.LiveUpdate *boolean*	No command line equivalent.	Use this property to specify whether the JDBC Gateway monitors its properties file for any changes made while running, and whether it updates those property values that have changed. This property takes the following values: FALSE: The gateway does not monitor its properties file while running. TRUE: The running process monitors its properties file for any changes and updates those property values that have changed. **Note :** Whether a property value becomes effective depends on what the property defines and also on how it is consumed internally by the application. For example, the JDBC Gateway retrieves the **Gate.Jdbc.Username** and **Gate.Jdbc.Password** properties only when there is a write failure. If the property file was modified but there is no write failure, the **Gate.Jdbc.Username** and **Gate.Jdbc.Password** properties will not be reloaded regardless of whether **Props.LiveUpdate** is set to TRUE. Property values that were originally set using the command line will not be altered even if their values are updated in the properties file. This is because values specified on the command line always override those set in the properties file.

Table 8. Common Tivoli Netcool/OMNIbus properties (continued)		
Property name	**Command line option**	**Description**
PropsFile *string*	`-propsfile` *string*	Use this property to specify the location of the gateway properties file. The default is `$OMNIHOME/etc/G_JDBC.props`.
UniqueLog *boolean*	`-uniquelog` *boolean*	Use this property to specify that log file names are made unique by adding the Process ID (PID) of the gateway to the file name. The default is `FALSE`.

JDBC Gateway properties

Table 9 on page 30 lists the available JDBC gateway properties.

Table 9. JDBC Gateway properties		
Property name	**Command line option**	**Description**
Gate.Jdbc.ActionCodeField *string*	`-jdbcactioncodefield` *string*	Use this property to specify the column name for action code information when the gateway is in audit mode. The default is `ACTIONCODE`.
Gate.Jdbc.ActionTimeField *string*	`-jdbcactiontimefield` *string*	Use this property to specify the column name for the action time information when the gateway is in audit mode. The default is `ACTIONTIME`.
Gate.Jdbc.Connections *integer*	`-jdbcconnections` *integer*	Use this property to specify the number of connections to the target database. The default is 3.
Gate.Jdbc.DeletedAtField *string*	`-jdbcdeleteatfield` *string*	Use this property to specify the field in the target table that is used to record the time of deletion of an alert. This property is only required when running the gateway in reporting mode. The default is `DELETEDAT`.
Gate.Jdbc.DetailsTableName *string*	`-jdbcdetailstablename` *string*	Use this property to specify the name of the target database table for storing data from the `alerts.details` ObjectServer table. The default is `REPORTER_DETAILS`.
Gate.Jdbc.Driver *string*	`-jdbcdriver` *string*	Use this property to specify the JDBC driver. The default is `""`.

Table 9. JDBC Gateway properties (continued)

Property name	Command line option	Description
Gate.Jdbc.DriverPasswordPropertyName *string*	-jdbcdriverpasswordpropname *string*	Use this property to specify the driver property key to the value configured in the **Gate.Jdbc.Password** property for the JDBC connection. If no value is specified for this property, the gateway will use the default value. The default is `password`.
Gate.Jdbc.DriverUserPropertyName *string*	-jdbcdriveruserpropname *string*	Use this property to specify the driver property key to the value configured in the **Gate.Jdbc.Username** property for the JDBC connection. If no value is specified for this property, the gateway will use the default value. The default is `user`.
Gate.Jdbc.DupIgnore *string*	`-jdbcdupignore` *string*	Use this property to specify what fields to ignore when de-duplicating alert updates. This property can take multiple values, each separated by a space. The default is `LastModified StateChange`.
Gate.Jdbc.FatalErrors *string*	`-jdbcfatalerrors` *string*	Use this property to specify which SQLSTATE prefixes are considered fatal when processing a batch of alerts. This property can take multiple values, each separated by a space. The default is `0A 42`.
Gate.Jdbc.InitializationString *string*	-jdbcinitializationstring *string*	Use this property to specify an SQL initialization string to execute on connection to the target database. The default is `""`.
Gate.Jdbc.InitializationTimeout *integer*	-initializationtimeout *integer*	Use this property to configure the timeout (in seconds) for the **Gate.Jdbc.InitializationString** operation. The default is `10`.

Table 9. JDBC Gateway properties (continued)

Property name	Command line option	Description
Gate.Jdbc.InitializeAllSessions *string*	`-jdbcinitallsessions` (This is equivalent to **Gate.Jdbc.InitializeAllSessions** with a value of `true`.) `-jdbcinitonesession` (This is equivalent to **Gate.Jdbc.InitializeAllSessions** with a value of `false`.)	Use this property to configure the scope of **Gate.Jdbc.InitializationString** operation in all sessions or just one session. The initialization that the amend table schema should be executed in one session, because duplicating action of such is a violation in most databases, the SQL exception will trigger the gateway to exit. The default is `true`.
Gate.Jdbc.JavaSystemPropsFile *string*	`-jdbcjavasystempropsfile` *string*	Use this property to specify the file containing the Java system properties used in JDBC connection. The default is `""`. **Notes :** 1. Do not use this file to hold Java System properties that are meant to be immediately applied when the JVM starts up. Use instead the **Gate.Java.Arguments** property. 2. Do not configure the same property in the **Gate.Java.Arguments** property and as the Java system properties file name, otherwise the value in the file will supersede it. See "Supporting configuration files" on page 44.
Gate.Jdbc.JdbcPropsFile *string*	`jdbcjdbcpropsfile` *string*	Use this property to specify the file containing driver properties for the JDBC connection. See Supporting Configuration Files. The default is `""`.
Gate.Jdbc.JournalTableName *string*	`-jdbcjournaltablename` *string*	Use this property to specify the name of the target database table for storing data from the `alerts.journal` ObjectServer table. The default is `REPORTER_JOURNAL`.
Gate.Jdbc.MaxBatchSize *integer*	`-jdbcmaxbatchsize` *integer*	Use this property to specify the maximum number of rows to process in a batch. The default is `250`.

Table 9. JDBC Gateway properties (continued)		
Property name	**Command line option**	**Description**
Gate.Jdbc.Mode *string*	`-jdbcmode` *string* `-jdbcaudit` (This is equivalent to **Gate.Jdbc.Mode** with a value of `AUDIT`.) `-jdbcreporter` (This is equivalent to **Gate.Jdbc.Mode** with a value of `REPORTING`.)	Use this property to specify the mode of operation of the gateway. This property takes the following values: `AUDIT`: The gateway runs in audit mode. `REPORTING`: The gateway runs in reporting mode. The default is `REPORTING`.
Gate.Jdbc.OrderedWrites *integer*	`-jdbcorderedwrites` *integer*	Use this property to insert status table rows before the journal and details rows. The default is `1`.
Gate.Jdbc.Partitioning Field *string*	`-jdbcpartitioningfield` *string*	Use this property to specify the field to use for partitioning. The default is `""`.
Gate.Jdbc.Password *string*	`-jdbcpassword` *string*	Use this property to specify the password for the target database. The default is `""`. **Note :** If you want to encrypt this password, use the `nco_aes_crypt` utility supplied with Tivoli Netcool/OMNIbus. For more information, see “FIPS mode and encryption” on page 52.
Gate.Jdbc.Preconnection Wait *integer*	`-jdbcpreconnectionwait` *integer*	Use this property to configure the wait interval (in milliseconds) for consecutive JDBC connection attempt to avoid spurious login issue caused by resource contention in the target’s authentication service. The default is `1000`.
Gate.Jdbc.Reconnect Timeout *integer*	`-jdbcreconnecttimeout` *integer*	Use this property to specify the time (in seconds) that the gateway waits before attempting to reconnect to the target database after losing the connection. The default is `30`.
Gate.Jdbc.ResyncFilter *string*	`-jdbcresyncfilter` *string*	Use this property to specify a filter for restricting open events in the target database when the gateway is operating in bidirectional resynchronization mode. The default is `""`.

Table 9. JDBC Gateway properties (continued)

Property name	Command line option	Description
Gate.Jdbc.ResyncMode *string*	-jdbcresyncmode *string* -jdbcresyncnone (This is equivalent to **Gate.Jdbc.ResyncMode** with a value of NONE.) -jdbcresyncuni (This is equivalent to **Gate.Jdbc.ResyncMode** with a value of UNI.) -jdbcresyncbi (This is equivalent to **Gate.Jdbc.ResyncMode** with a value of BI.) -jdbcresyncauto (This is equivalent to **Gate.Jdbc.ResyncMode** with a value of AUTO.)	Use this property to specify a resynchronization mode for the gateway. This property takes the following values: NONE: The gateway does not perform resynchronization. UNI: The gateway operates in unidirectional resynchronization mode. BI: The gateway operates in bidirectional resynchronization mode. AUTO: The gateway operates in unidirectional resynchronization mode by default. If its alert cache is empty on startup, which normally only occurs the first time it is run, the gateway operates in bidirectional resynchronization mode. The default is AUTO.
Gate.Jdbc.RetryErrors *string*	-jdbcretryerrors *string*	Use this property to specify which SQLSTATE prefixes will cause the current batch of alerts to be retried. This property can take multiple values, each separated by a space. The default is 08 28 40 HYT.
Gate.Jdbc.SerialField *string*	-jdbcserialfield *string*	Use this property to specify the serial field in the target database when the gateway is running in audit mode. The value that will populate the target database comes from the ServerSerial field. This is for delete records when the serial field is set to NOT NULL. The default gateway behaviour is that SERIAL is allowed to be NULL and this property does not need to be specified. The default is "".
Gate.Jdbc.Server NameField *string*	-jdbcservernamefield *string*	Use this property to specify the field in the target table that contains the server name. The default is SERVERNAME.
Gate.Jdbc.Server SerialField *string*	-jdbcserverserialname *string*	Use this property to specify the field in the target table that contains the server serial. The default is SERVERSERIAL.

Table 9. JDBC Gateway properties (continued)

Property name	Command line option	Description
Gate.Jdbc.SqlTimeout *integer*	`-sqltimeout` *integer*	Use this property to configure the timeout (in seconds) for SQL operations. The default is `600`. **Note :** 600 seconds is used for backward compatibility as the legacy implementation was using the same hardcoded timeout value.
Gate.Jdbc.Status TableName *string*	`-jdbcstatustablename` *string*	Use this property to specify the name of the target database table for storing data from the `alerts.status` ObjectServer table. The default is `REPORTER_STATUS`.
Gate.Jdbc.SuppressDeletes *boolean*	`-jdbcsuppressdeletes` *boolean*	Use this property to suppress deleted events from the ObjectServer. The following values are valid: • `TRUE` : The gateway suppresses deleted events. • `FALSE` : The gateway does not suppress deleted events. The default is `FALSE`.
Gate.Jdbc.TestCount *integer*	`-jdbctestcount` *integer* `-jdbcnotest`	Use this property to specify the number of internal events the gateway generates to test the JDBC connection. This property specifies the number of test alerts to create, update, journal then delete, populating the target database in the process. The default is `0`. You can use the `-jdbcnotest` command line option to set the **Gate.Jdbc.TestCount** value to 0. This turns off test alerts. **Note :** The simulated alerts use the server name `JDBC_TEST`.
Gate.Jdbc.Url *string*	`-jdbcurl` *string*	Use this property to specify the URL of the target database. The default is `" "`.

Table 9. JDBC Gateway properties (continued)		
Property name	**Command line option**	**Description**
Gate.Jdbc.Username *string*	-jdbcusername *string*	Use this property to specify the user name for the target database. The default is " ". **Note :** For more information about configuring this property see the following configuration section: "Configuring the database connection" on page 13.
Gate.Jdbc.UnknownErrors *string*	-jdbcunknownerrors *string*	Use this property to specify how the gateway handles unknown SQL errors. This property takes the following values: ABORT: The gateway aborts the operation. IGNORE: The gateway ignores the error message. RECONNECT: The gateway attempts to reconnect to the target database. RETRY: The gateway retries the operation that caused the error message. The default is RECONNECT.

Generic gateway properties

Table 10 on page 36 lists the available generic gateway properties.

Table 10. Generic gateway properties		
Property name	**Command line option**	**Description**
Gate.CacheHashTblSize *integer*	-cachehtblsize *integer*	Use this property to specify the number of elements the gateway allocates for the hash table cache. The default is 5023.
Gate.MapFile *string*	-mapfile *string*	Use this property to specify the mapping file for the gateway to use. The default is $OMNIHOME/gates/jdbc/jdbc.map.
Gate.NGtkDebug *boolean*	-ngtkdebug *boolean*	Use this property to enable the logging of NGTK library debug messages. The default is TRUE.

Table 10. Generic gateway properties (continued)

Property name	Command line option	Description
Gate.PAAware *integer*	-paaware *integer*	This property indicates whether the gateway is PA aware. The default is 0 (not PA aware). **Note :** This property is maintained by the PA server and is included in the properties file for information only.
Gate.PAAwareName *string*	-paname *string*	Use this property to specify the name of the Process Agent controlling the gateway. The default is "". **Note :** This property is maintained by the PA server and is included in the properties file for information only.
Gate.ResyncTables *string*	-gateresynctables *string*	Use this property to specify the name of the target database table for storing data from a secondary dynamic ObjectServer table. The default is "". **Note :** This property accepts spaces, tabs, colons, and commas as table name separators. For more information see, "Table replication definition file" on page 47.
Gate.StartupCmdFile *string*	-startupcmdfile *string*	Use this property to specify the location of the startup command file. The default is $OMNIHOME/gates/jdbc/jdbc.startup.cmd.
Gate.Transfer.FailoverSyncRate *integer*	-fsyncrate *integer*	Use this property to specify the rate (in seconds) of the failover synchronization. The default is 60.
Gate.UnixAdminGrp *string*	-unixadmingrp *string*	Use this property to specify the administration group to which the gateway must belong if standard UNIX authentication is used. The default is ncoadmin.

Table 10. Generic gateway properties (continued)		
Property name	**Command line option**	**Description**
Gate.UsePamAuth *boolean*	-usepamauth *boolean*	Use this property to specify whether PAM authentication is used. The default is FALSE. **Note :** To run the gateway in FIPS 140-2 mode, you must set this property to TRUE.

Java properties

Table 11 on page 38 lists the available Java properties.

Table 11. Java properties		
Property name	**Command line option**	**Description**
Gate.Java.Arguments *string*	-javaarguments *string*	Use this property to specify the arguments to use when starting Java. The default is "".
Gate.Java.ClassPath *string*	-javaclasspath *string*	Use this property to specify the environment variable used to store the location of the Java libraries. The default is $CLASSPATH.
Gate.Java.Debug *boolean*	-javadebug *boolean*	Use this property to enable the logging of Java debug messages. The default is TRUE.
Gate.Java.LibraryPath *string*	-javalibrarypath *string*	Use this property to specify the location of the Java libraries that will be set in the environment variable specified by the **Gate.Java.ClassPath** property. The default is "".

Mapping properties

Table 12 on page 39 lists the available mapping properties.

Table 12. Mapping properties		
Property name	**Command line option**	**Description**
Gate.Mapper.Debug *boolean*	`-mapperdebug` *boolean*	Use this property to enable the logging of mapper debug messages. The default is TRUE.
Gate.Mapper.Forward HistoricDetails *boolean*	`-mapperforhistdtls` *boolean*	Use this property to specify whether the gateway forwards all historic details on converted update. The default is FALSE.
Gate.Mapper.Forward HistoricJournals *boolean*	`-mapperforhistjrnl` *boolean*	Use this property to specify whether the gateway forwards all historic journals on converted update. The default is FALSE.

Gateway Reader-Writer properties

Table 13 on page 39 lists the available gateway reader-writer properties.

Table 13. Gateway Reader-Writer properties		
Property name	**Command line option**	**Description**
Gate.RdrWtr.BufferSize *integer*	`-rdrwtrbufsize` *integer*	Use this property to specify the number of entries that the gateway stores in the buffer before flushing, if buffering is enabled. This property can be used to fine-tune the efficiency of the gateway. The default is 25. **Note :** The gateway flushes the buffer when the end of a batch of SQL statements has been reached regardless of the buffer size.
Gate.RdrWtr.CommonNames *string*	`-rdrwtrcommonnanes` *string*	Use this property to specify a list of common names. The default is `""`.
Gate.RdrWtr.Debug *boolean*	`-rdrwtrdebug` *boolean*	Use this property to specify whether the gateway includes gateway reader debug messages in the debug log. The default is TRUE.

Table 13. Gateway Reader-Writer properties (continued)

Property name	Command line option	Description
Gate.RdrWtr.DeleteIfNo Dedup *boolean*	`-rdrwtrdeleteifnodedup` *boolean*	Use this property to specify whether the gateway deletes duplicate events. The default is FALSE.
Gate.RdrWtr.Description *string*	`-rdrwtrdescription` *string*	Use this property to specify the application description for the reader connection. This description is used in triggers and allows you to determine which component of the gateway attempted to perform an action. The default is "`Gateway Reader/Writer`".
Gate.RdrWtr.DetailsTable Name *string*	`-detailstblname` *string*	Use this property to specify the name of the status table that the gateway reads. The default is `alerts.details`.
Gate.RdrWtr.Failback Enabled *boolean*	`-rdrwtrfailbackenabled` *boolean*	Use this property to specify whether the gateway attempts to fail back to the primary ObjectServer following an ObjectServer failover. The default is `False`. **Note :** The gateway attempts to fail back with the frequency specified by the **Gate.RdrWtr. FailbackTimeout** property.
Gate.RdrWtr.Failback Timeout *integer*	`-readerfailbacktimeout` *integer*	Use this property to specify the frequency (in seconds) with which the gateway attempts to fail back to the primary system following a system failover. The default is `30`. **Note :** The gateway attempts to fail back to the primary ObjectServer only if the **Gate.RdrWtr. FailbackEnabled** property is set to TRUE.

Table 13. Gateway Reader-Writer properties (continued)

Property name	Command line option	Description
Gate.RdrWtr.IducFlushRate *integer*	`-iducflushrate` *integer*	Use this property to specify the rate (in seconds) of the granularity of the reader. If you set this property to `0`, the reader gets its updates at the same granular rate as that of the ObjectServer to which it is connected. The default is `0`. **Note :** If you set this property to a value greater than 0, the reader issues automatic IDUC flush requests to the ObjectServer with this frequency. This enables the reader to run at a faster granularity than that of the ObjectServer, thus enabling the gateway to capture more detailed event changes in systems where the ObjectServer itself has high granularity settings.
Gate.RdrWtr. JournalTableName *string*	`-journaltblname` *string*	Use this property to specify the name of the status table that the gateway reads. The default is `alerts.journal`.
Gate.RdrWtr.LogOSSql *boolean*	`-logossql` *boolean*	Use this property to specify whether the gateway logs all SQL commands sent to the ObjectServer in debug mode. The default is FALSE.

Table 13. Gateway Reader-Writer properties (continued)		
Property name	**Command line option**	**Description**
Gate.RdrWtr.Password *string*	-password *string*	Use this property to specify the password associated with the user specified by the **Gate.RdrWtr.Username** property. You must specify this password when connecting to the ObjectServer in secure mode. The default is "". **Note :** When connecting to the ObjectServer in secure mode the password may be in plain text, but if you want to encrypt this password, use the nco_aes_crypt utility supplied with Tivoli Netcool/OMNIbus. For more information, see "FIPS mode and encryption" on page 52.
Gate.RdrWtr.ReconnectTimeout *integer*	-reconntimeout *integer*	Use this property to specify the time (in seconds) between each reconnection poll attempt that the gateway makes if the connection to the ObjectServer is lost. The default is 30.
Gate.RdrWtr.RefreshCacheOnUpdate *boolean*	-rdrwtrrefresh cacheonupdate *boolean*	Use this property to specify whether the gateway refreshes the cache after each update. The default is FALSE.
Gate.RdrWtr.Server *string*	-rdrwtrserver *string*	Use this property to specify the name of the ObjectServer from which the gateway reads alerts The default is NCOMS.
Gate.RdrWtr.StatusTableName *string*	-rdrwtrstatustablename *string*	Use this property to specify the name of the target database table from which the ObjectServer reads data. The default is alert.status.

Table 13. Gateway Reader-Writer properties (continued)		
Property name	**Command line option**	**Description**
Gate.RdrWtr.TblReplicateDefFile *string*	`-tblrepdeffile` *string*	Use this property to specify the path to the table replication definition file. The default is `$OMNIHOME/gates/jdbc/jdbc.rdrwtr.tblrep.def`.
Gate.RdrWtr.Username *string*	`-username` *string*	Use this property to specify the user name used to authenticate the ObjectServer connection. This property is used with the **Gate.RdrWtr.Password** property. The default is `root`. **Note :** When connecting to the ObjectServer in secure mode you must specify this property.
Gate.RdrWtr.UseBulkInCmd *boolean*	`-usebulkincmd` *boolean*	Use this property to enable SQL commands to be inserted in bulk into the gateway. The default is `FALSE`.

Connection properties

Table 14 on page 43 lists the available mapping properties.

Table 14. Connection properties		
Property name	**Command line option**	**Description**
Gate.StartupCmdFilePath *string*	`-startupcmdfilepath` *string*	Use this property to specify the file path to the gateway start up file. The default is `$OMNIHOME/gates/jdbc/jdbc.startup.cmd`.
Gate.Transfer.FailoverSyncRate *integer*	`-transferfailoversyncrate` *integer*	Use this property to specify the rate (in seconds) at which the re-synchronization takes place on failover. The default is `60`.
Gate.UnixAdminGrp *string*	`-unixadmingrp` *string*	Use this property to specify the UNIX authentication administration group name The default is `NCOADMIN`.

Table 14. Connection properties (continued)		
Property name	**Command line option**	**Description**
Gate.UsePamAuth *boolean*	`-usepamauth` *boolean*	Use this property to enable the gateway to use PAM authentication. This property takes two values: TRUE : The gateway uses PAM authentication. FALSE : The gateway uses UNIX authentication. The default is FALSE.

Supporting configuration files

The gateway package bundles supporting configuration files to hold properties for secure JDBC connection modes, such as SSL, Kerberos, and data integrity.

Table 15. Supporting configuration files	
Configuration file	**Purpose**
`jdbc_conn.properties`	JDBC connection properties for the database connection. Configure the path to this file using the **Gate.Jdbc.JdbcPropsFile** property.
`jdbc_javasys.properties`	Java system properties working in conjunction with other JDBC connection parameters to establish a database connection. Configure the path to this file using the **Gate.Jdbc.JavaSystemPropsFile** property.

File format

The supporting configuration files should be in key-value pair format, namely: `key=value`

The gateway ignores lines starting with #.

Date security

The gateway can process value field in the form of encrypted data.

Use `nco_aes_crypt` with AES_FIPS algorithm to encrypt configuration value. When deployed, **ConfigCryptoAlg** and **ConfigKeyFile** must contain AES_FIPS and the key file respectively.

Environment Variables

The gateway can expand defined environment variables in value fields.

Properties for the various database connection modes

Table 16. Oracle database		
Mode	**JDBC Properties**	**Java System properties**

Table 16. Oracle database (continued)		

SSL	`oracle.net.authentication_services`	`javax.net.ssl.keyStore`
		`javax.net.ssl.keyStoreType`
		`javax.net.ssl.keyStorePassword`
		`javax.net.ssl.trustStore`
		`javax.net.ssl.trustStoreType`
		`javax.net.ssl.trustStorePassword`
		
		`oracle.net.ssl_server_dn_match`
		`oracle.net.ssl_cipher_suites`
		`oracle.net.tns_admin`

Kerberos	`oracle.net.authentication_services`
	`oracle.net.kerberos5_cc_name`
	`oracle.net.kerberos5_mutual_authentication`
		`java.security.krb5.conf`

Data Integrity	`oracle.net.crypto_checksum_client`
	`oracle.net.crypto_checksum_types_client`
	`oracle.net.encryption_client`
	`oracle.net.encryption_types_client`
		None

Note : For Kerberos-SSL-combined connections, `oracle.net.authentication_services` must contain SSL and KERBEROS5, for example:

`oracle.net.authentication_services=(KERBEROS5, SSL)`

Table 17. IBM DB2		
Mode	**JDBC connection properties**	**Java System properties**
Kerberos	`kerberosServerPrincipal` `securityMechanism=11`	`java.security.auth.login.config`

SSL	None	`javax.net.ssl.trustStore`
		`javax.net.ssl.trustStoreType`
		`javax.net.ssl.trustStorePassword`

Configure the following parameters to enable IBM DB2 Kerberos-authenticated connections:

`java.security.auth.login.config=$OMNIHOME/gates/jdbc/jaas.conf`

`kerberosServerPrincipal=<`*`serverPrincipal`*`>@<`*`REALM`*`>`

`securityMechanism=11` (11 is the value for Kerberos security).

Notes :

`kerberosServerPrincipal` must be the server principal in `KRB5_KTNAME`.

`securityMechanism=11` denotes using Kerberos authentication.

A sample of Java Authentication and Authorization Service (JAAS) Client configuration is available in the `$OMNIHOME/gates/jdbc/jaas.conf` file.

Configurations for passing credentials from the gateway to the DB2 Kerberos service

There are two ways in which the gateway can pass credentials to the DB2 Kerberos service:

- Using the Kerberos cache file
- Not using the Kerberos cache file

Configurations for both methods are described in the following table.

Table 18. Configurations for passing credentials to the DB2 Kerberos service

Using Kerberos Cache Files	Not Using Kerberos Cache Files

`JaasClient{`
 	
`com.ibm.security.auth.module.Krb5LoginModule required`
`principal=<kerberos_principal>`
`credsType=initiator`
`useCcache=<kerberos_cache_file>`
`debug=true`
`};`
 	
Configure the `principal` and `useCache` fields accordingly.
 	
As the login credentials are obtained from Kerberos cache, the corresponding gateway properties must be empty. Namely:
 	
`Gate.Jdbc.Username: ''`
`Gate.Jdbc.Password: ''`
	`JaasClient{`
	
	`com.ibm.security.auth.module.Krb5LoginModule required`
	`debug=true`
	`useDefaultCcache=false;`
	`};`
	
	The absence of the Kerberos cache requires the gateway properties to have the login credentials specified in the following properties:
	
	`Gate.Jdbc.Username: '<principal>'`
	`Gate.Jdbc.Password: '<password>'`

Deploying the gateway with non-IBM Java

On secure connections, some vendor JDBC drivers may require certain class packages tied to specific brand and version of Java.

For example, in Kerberos mode, the Oracle JDBC driver invokes the methods from the `sun.security.krb5.*` classes, which are available only in Oracle Java.

Consult the driver user guide or the vendor support to ascertain the Java requirement.

To run the gateway using non-IBM Java, perform the steps accordingly.

UNIX:

Update the `NCO_GATEWAY_JRE` environment variable with the Java home directory.

Note: `NCO_GATEWAY_JRE` is referenced in `nco_g_jdbc.env`.

Map definition file

The mapping definition file defines how the gateway maps data received from the ObjectServer to tables in the target database.

The default map definition file installed with the gateway, `jdbc.map`, configures the gateway to operate in reporting mode. If you want to run the gateway in audit mode, copy `$OMNIHOME/gates/jdbc/audit.jdbc.map` to `$OMNIHOME/gates/jdbc/jdbc.map`.

The default map definition file contains example mappings. It is advisable to make a backup copy of the default file for future reference.

Startup command file

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

You can specify the location of the startup command file using the generic Netcool/OMNIbus **`Gate.StartupCmdFile`** property.

The default startup command file is located in the following directory: `$OMNIHOME/gates/jdbc/jdbc.startup.cmd`

The default startup command file contains example commands. You should make a copy of the default file for future reference.

You can use the following commands within the startup command file:

- SHOW PROPS - Use this command to display the current configuration of the gateway by listing all properties and their values.
- GET PROPERTY '*property_name*' - Use this command to return the value of the property specified in *property_name* from the gateway properties file.
- SET PROPERTY '*property_name*' TO ('*string*' | *integer* | TRUE | YES | FALSE | NO); - Use this command to set the value of the property specified in *property_name* in the gateway properties file.
- SET LOG LEVEL TO - Use this command to set the level of message logging for the gateway. This command can take the following values: fatal, error, warn, info or debug. The default logging level is warn.

These commands can also be entered using the SQL interactive interface (nco_sql). For more information about using the SQL interactive interface, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

For more information about the startup command file, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

TRANSFER command

The TRANSFER command transfers data from an ObjectServer table to a target table using a transfer map. The data can also be filtered. This command is most useful is cases where once populated, the target table contents are unlikely to change.

The TRANSFER command takes the following syntax:

TRANSFER FROM '*source*' [TO '*target*']

The TRANSFER command is usually specified in the startup command file. To run the TRANSFER command from the startup command file use the following example:

```
TRANSFER FROM 'source_table' TO 'target_table'
            VIA FILTER 'Colname != \'Severity\''
            WITH DELETE VIA 'Column_Name <> \'Severity\''
            USING TRANSFER_MAP GatewayTablesMap;
```

Note : To replicate data from dynamic secondary tables you must use the **Gate.ResyncTables** property in the table replication definition file.

This command can also be entered using the SQL interactive interface (nco_sql). For more information about using the SQL interactive interface, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

For more information about the startup command file and the TRANSFER command, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table replication definition file

The gateway replicates data between ObjectServer tables and the gateway target. The table replication definition file is used to define which tables and event types are monitored in Tivoli Netcool/OMNIbus and forwarded to the target that the gateway is configured to send data to.

You can specify the location of the table replication definition file using following generic Tivoli Netcool/OMNIbus property.

Gate.Reader.TblReplicateDefFile

The default table replication definition file is in the following directory: $OMNIHOME/gates/jdbc/jdbc.rdrwtr.tblrep.def

The default table replication definition file contains example commands. You should make a backup copy of the default file for future reference.

Note : You should use the REPLICATE command to replicate data from the primary tables (`alerts.status`, `alerts.journal`, `alerts.details`) and dynamic secondary tables (if required).

You can add one or more optional clauses to the REPLICATE command to further process the data during replication. The available commands are listed in the following syntax example. Use the optional clauses in the order in which they are listed in the syntax. For example, when using both the

FILTER WITH and AFTER IDUC DO clauses, the FILTER WITH clause must precede the AFTER IDUC DO clause.

```
REPLICATE ALL | (INSERTS, UPDATES, DELETES)
FROM TABLE sourcetable
USING MAP mapname
[FILTER WITH filter]
[INTO targettable]
[ORDER BY order, ... ]
[AFTER IDUC DO afteriduc] ;
```

Table 19. Optional replication commands

Command	Description
FILTER WITH '*filter*'	Filters the database rows selected for replication, where *filter* defines the filter that the gateway uses in the WHERE clause of the SQL SELECT. Filtering is positive by default, which means that only those events that match the filter definition are replicated. You can use a negative filter by putting an exclamation mark (!) before the equals sign (=) in the filter clause. For example, the following filter clause replicates all events whose severity is not 5: FILTER WITH 'Severity !=5'
ORDER BY '*order*'	Order results by the SQL SELECT ORDER BY clause used to get data. A potential use case might be to order by first occurrence, so that alerts are processed in chronological order, in which case the value specified for *order* would be 'FirstOccurrence'.
AFTER IDUC DO '*afteriduc*'	Updates replicated rows, where *afteriduc* specifies which field to update with what value. This uses the SQL UPDATE action to execute on rows retrieved by the SQL SELECT action used to get data, e.g. 'SentToCRM=1'.

Using the `Gate.ResyncTables` command to resynchronize data on startup from dynamic tables

To replicate data from dynamic secondary tables you must use the **`Gate.ResyncTables`** property. You must configure this property before starting the gateway. Any tables in addition to the main `alerts.status`, `alerts.details`, and `alerts.journal` tables, which are resynchronized by default, should be specified in this property. To configure this property, use the following steps:

1. Specify the table using the **`Gate.ResyncTables`** command. The command accepts spaces, tabs, colons and commas as table name separators. For example,

   Gate.ResyncTables: 'alerts.conversions *custom.name.one custom.name.two*'
2. Add an entry to the table replication definition file for the dynamic table.

Note : Using this method to resynchronize tables may cause constraint errors in the target database.

Converting alerts into a more readable format in a reporter table

This topic describes how to convert alerts into a more readable format in a reporter table.

The `alerts.conversions` table is used to provide an easy conversion from a numeric value to a string for any column.

Conversions are associated with the columns in the ObjectServer `alerts.status` table, and they map integer values and UTC(time) values for the columns to string values. The conversions that are configured in the Netcool/OMNIbus Administrator are used in the event list, to translate integer values into strings for readability. For example, default conversions exist for event severities; if an event has a severity of 4, the text `Major` is displayed for the event severity in the event list.

To configure the conversions table, use the following steps:

1. Add a definition to $OMNIHOME/gates/jdbc/jdbc.rdrwtr.tblrep.def:

   ```
   REPLICATE ALL FROM TABLE 'alerts.conversions' USING MAP 'ConversionsMap' INTO
   'reporter_conversions';
   ```

2. Add a transfer request to provide initial values for such tables in `jdbc.startup.cmd`:

   ```
   TRANSFER FROM 'alerts.conversions' TO 'reporter_conversions' DELETE USING TRANSFER_MAP
   ConversionsMap;
   ```

3. Go into OMNIbus and do: `update alerts.conversions set Value=Value;`
4. On the database server, verify whether the database user has access to the `reporter_conversions` table.
5. Grant the user access if necessary. For example:

   ```
   GRANT ALL ON REPORTER.REPORTER_CONVERSIONS TO "KUSER1@FYRE.IBM.COM";
   ```

6. Start the gateway.

AfterIDUC and Filter functions

The Gateway for Oracle and the Gateway for ODBC implemented `AfterIDUC` and filter functions using the **`Gate.ReaderAfterIDUC`** and **`Gate.ReaderFilter`** properties. The Gateway for JDBC uses the table replication definition file, **`Gate.RdrWtr.TblReplicateDefFile`**, to perform the same functions.

Use the `AFTER IDUC DO` command to update replicated rows. In the following example, the `AFTER IDUC DO` clause instructs the gateway to set the `Archived` column to 1 for all replicated rows:

```
REPLICATE ALL FROM TABLE 'alerts.status'
    USING MAP 'StatusMap'
    AFTER IDUC DO 'Archived=1';
```

Use the `FILTER WITH` command to filter the data that is replicated to the target database. In the following example, the `FILTER WITH` clause instructs the gateway to only replicate alerts that originate from the NCOMS_US and NCOMS_CA ObjectServers:

```
REPLICATE ALL FROM TABLE 'alerts.status'
    USING MAP 'StatusMap'
    FILTER WITH 'ServerName IN (\'NCOMS_US\',\'NCOMS_CA\')';
```

Note : You must use backslash characters (\) to escape the quote characters (') in the query string.

An important consideration when filtering replicated data is that the filter should match a characteristic of the alerts that does not change over time. This can be the alert source, as in the example above. You could also use the Class and Manager elements of alerts as a filter. Elements of alerts such as `Severity` make for bad filter criteria because they can change over the lifecycle of an alert.

Using a partitioning field

You can use the **Gate.Jdbc.PartitioningField** property to qualify the identification of alerts for the purpose of partitioning the target database.

By default, alerts are identified using the `ServerName` and `ServerSerial` fields. You can use the **Gate.Jdbc.PartitioningField** property to specify an additional field that can be used to identify alerts for the purpose of database partitioning. The `FirstOccurrence` field is a typical choice for this purpose.

Filtering resynchronization data

You can use the **Gate.Jdbc.ResyncFilter** property to specify a filter for restricting open events in the target database. The events are filtered when the gateway is operating in bidirectional resynchronization mode.

When you use the **Gate.Jdbc.ResyncFilter** property, the gateway filters open events from the target database using the server name of the ObjectServer as the filter criterion. The filter that you specify becomes part of an SQL `WHERE` clause. The `WHERE` clause is then appended to the `SELECT` query that the gateway uses to retrieve events from the target database.

The following example illustrates a typical use case for the **Gate.Jdbc.ResyncFilter** property.

Example

Assume that you are using two Netcool/OMNIbus ObjectServers, one for production and one for testing, each with its own gateway. The server name of the production ObjectServer is PROD and the server name of the test ObjectServer is TEST. Both gateways are archiving data into the same target database.

When the TEST gateway is performing a bidirectional resynchronization between the TEST ObjectServer and the database, you want to prevent the TEST gateway from closing events that came from the PROD ObjectServer. To do this, you can specify the following filter value for the **Gate.Jdbc.ResyncFilter** property of the TEST gateway:

```
ServerName LIKE 'TEST%'
```

This filter causes the TEST gateway to ignore open events in the target database that came from the PROD ObjectServer. It only resynchronizes open events that came from the TEST ObjectServer.

Note : When specifying values for the **Gate.Jdbc.ResyncFilter** property in the properties file, you must use backslash characters (\) to escape the single quote characters ('). For example:

```
Gate.Jdbc.ResyncFilter: 'ServerName LIKE \'TEST%\''
```

SQL executed by the gateway

This sections describes the SQL that gateway the executes over the target database during resync on startup if enabled.

Note : In the following examples <> brackets enclose variables determined by gateway properties as follows: <variablename> is determined by the **Gate.Jdbc.<variablename>** property. For example <ServerNameField> is determined by the **Gate.Jdbc.ServerNameField** property.

When the gateway is running in Reporter Mode

Gate.Jdbc.ReysncFilter set to default empty string value:

```
select <ServerNameField>, <ServerSerialField> from <StatusTableName> where
<DeletedAtField> is null
```

Gate.Jdbc.ReysncFilter defined by user:

```
select <ServerNameField>, <<ServerSerialField>> from <StatusTableName> where
<DeletedAtField> is null and <ResyncFilter>
```

When the gateway is running in Audit Mode

Gate.Jdbc.ReysncFilter set to default empty string value:

```
select distinct <ServerNameField>, <ServerSerialField> from <StatusTableName> o
where o.<ActionCodeField> = 'I' and not exists

(select * from <StatusTableName> d where d.<ServerNameField>
=o.<ServerNameField> and d.<ServerSerialField> = o.<ServerSerialField> and
d.<ActionCodeField> = 'D')
```

Gate.Jdbc.ReysncFilter defined by user:

```
select distinct <ServerNameField>, <ServerSerialField> from <StatusTableName> o
where o.<ActionCodeField> = 'I' and not exists

(select * from <StatusTableName> d where d.<ServerNameField>
=o.<ServerNameField> and d.<ServerSerialField> = o.<ServerSerialField> and
d.<ActionCodeField> = 'D') and <ResyncFilter>
```

Note : A complex ResyncFilter may need to include enclosing brackets: (and) to ensure the correct order of precedence in the SQL definition.

Message log file

The gateway creates a message log file to store all messages that it generates while running.

You can use the **MessageLog** property to specify a name for the message log file. The default log file `G_JDBC.log` is located in the following directory:

```
$OMNIHOME/log
```

You can specify the maximum size of the log file using the **MaxLogFileSize** property. The default is 1024 KB. When the log file reaches the specified maximum size, the ObjectServer archives it using the extension `.log_old` and starts a new log file with the extension `.log`. When the new log file reaches the maximum size, it is archived in turn and overwrites the first archived log file.

You can specify the level of message logging using the **MessageLevel** property. The default is `warn`. The default logging level is sufficient to identify most configuration problems. If you require more information about how the gateway is running, set the **MessageLevel** property to debug. This option produces a lot of detailed output, resulting in large log files.

Log file environment variables

You can control the log file size and log file rotation using the following environment variables:

- `NDE_LOGFILE_MAXSIZE` sets the maximum log file size.

  The following example sets the maximum log file size to 1024000 bytes (1024 KB):

  ```
  setenv NDE_LOGFILE_MAXSIZE 1024000
  ```

- `NDE_LOGFILE_ROTATION_FORMAT` enforces daily log file rotation, regardless of the maximum log file size specified by `NDE_LOGFILE_MAXSIZE`. It also specifies the format of the archived log file name.

  You can use one of the following types of parameter to set this variable: any literal string (for example, `rotation`), a POSIX timestamp format, or a Unicode Locale Data Markup Language (LDML) timestamp format. The literal string or the timestamp is appended to the archived log file, for example, `nco_g_odbc.log_rotation`.

  The following commands enable daily log file rotation:

Parameter	Command
Literal string	`setenv NDE_LOGFILE_ROTATION_FORMAT \'`*`literal_string`*`\'`
POSIX timestamp	`setenv NDE_LOGFILE_ROTATION_FORMAT %Y%m%d-%H%M`

Parameter	Command
LDML timestamp	`setenv NDE_LOGFILE_ROTATION_FORMAT yyyyMMdd-HHmm`

`NDE_LOGFILE_ROTATION_TIME` specifies the time at which log file rotation occurs each day. The following example causes the log file to be rotated at 00:00 hours each day:

```
setenv NDE_LOGFILE_ROTATION_TIME 0000
```

For more information about using log file environment variables, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

FIPS mode and encryption

This gateway complies with Federal Information Processing Standard 140-2 (FIPS 140-2). It can be run in FIPS mode on any currently supported version of Tivoli Netcool/OMNIbus.

You can use encryption algorithms to secure string value entries made in the properties file, including passwords. You must use the generic Tivoli Netcool/OMNIbus **ConfigCryptoAlg** property to specify the encryption method and the generic Tivoli Netcool/OMNIbus **ConfigKeyFile** property to specify the encryption key file, amongst a number of other required settings.

For more information about running the gateway in FIPS mode, and encrypting properties and passwords, see *Running the ObjectServer in secure mode*, *Running the proxy server in secure mode*, and *Encrypting plain text passwords in routing definitions* in the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

Also see, *Configuring FIPS 140-2 support for the server components* in the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

Also see *SSL and FIPS 140-2 support* in the *IBM Tivoli Netcool/OMNIbus Event Integration Facility Reference*.

Also see *Appendix C. WAAPI security* in the *IBM Tivoli Netcool/OMNIbus Web GUI Administration API (WAAPI) User's Guide*.

Note : If you run the gateway in FIPS mode, you must either use no encryption, or if you do use encryption, you must use `nco_aes_crypt` with the cipher (`-c`) option `AES_FIPS`. The cipher option used here must match the option specified by the **ConfigCryptoAlg** property. For example:

```
$NCHOME/omnibus/bin/nco_aes_crypt -c AES_FIPS -k key_file string_value
```

Gateway statistics

The gateway logs Reader and Writer statistics to its log file. You can use these statistics to monitor the gateway's performance.

For the Reader component, the gateway logs the time taken to read and write an entire work batch to disk, measured in milliseconds. This measurement is also expressed as the number of database rows processed per second.

For the Writer component, the gateway logs the number of outstanding (unprocessed) batches and the running row rate, measured as database rows per second processed.

When the gateway is performing adequately, the number of outstanding batches should be 0 or 1. If the number of outstanding batches is greater than 1, this indicates that the gateway is not performing adequately.

The running row rate is a weighted average of the historical running row rate and the row rate of the last processed batch. The running row rate is calculated as 0.625 of the last measured running row rate plus 0.375 of the last batch row rate ((0.625 * last measured running row rate) + (0.375 * last batch row rate)).

The gateway also logs the time taken to write the persistent cache, measured in milliseconds. This time is approximately proportional to the number of open alerts being managed by the gateway.

The gateway statistics are written to the default log file `G_JDBC.log`. To enable easy retrieval of the data, every line of statistics output in the log file contains the string "STATS".

Note : Gateway statistics information is logged at the `information` log level, which is off by default. To enable logging of statistics, you must set the **`MessageLevel`** property to `information`.

Example log file

The following is an example of the statistics information logged by the gateway.

```
11/02/11 23:26:23: Information: I-GJA-000-000: [ngjava]: G_JDBC:
Thread-3: STATS: 9fbb3866-bf81-4fa6-9944-ae99eaafee45
Batch write time 11 ms (1000.0 rows/second)
```

```
11/02/11 23:26:23: Information: I-GJA-000-000: [ngjava]: G_JDBC:
Thread-3: STATS: Cache write time 7 ms
```

```
11/02/11 23:26:24: Information: I-GJA-000-000: [ngjava]: G_JDBC:
pool-1-thread-1: STATS: 9fbb3866-bf81-4fa6-9944-ae99eaafee45
Batch execution time: 473
```

```
11/02/11 23:26:24: Information: I-GJA-000-000: [ngjava]: G_JDBC:
pool-1-thread-1: STATS:
Running row rate 18.907563025210084 rows/second
```

```
11/02/11 23:26:24: Information: I-GJA-000-000: [ngjava]: G_JDBC:
pool-1-thread-1: STATS: Outstanding batches 0
```

Error messages

Error messages provide information about problems that have occurred during the operations of the gateway. You can use the information that they contain to resolve such problems.

The following table describes the error messages that the gateway generates:

Table 20. Error messages

Error	Description	Action
`Batch creation failed` *`exception`*	The gateway failed to create a new batch of work in store and forward.	Check the exception text for an indication of the cause of the error. If the cause of the problem is not clear from the error message, contact IBM Software Support.
`Data UUID of recovery file (`*`uuid`*`) does not match` *`uuid`*	A persistent file has been corrupted. Persistent files are differentiated by universally unique identifiers (UUID). The UUID is used to name a persistent file and is also embedded in the contents of that file. The error message is indicating that, for a particular persistent file, the embedded UUID does not match the file name UUID. The file has been corrupted.	Locate, and save a copy of, the file named *uuid* in the `$OMNIHOME/var/G_JDBC/` directory. Contact IBM Software Support.

Table 20. Error messages (continued)		
Error	**Description**	**Action**
`DML type unknown: type`	A data manipulation language (DML) command of an unknown type was requested. Inserts, updates, and deletes are the only types supported by the gateway.	Contact IBM Software Support.
`Failed to rename file to uuid`	The gateway could not write a persistent file named *uuid*.	Verify that the gateway has write permission for the `$OMNIHOME/var/G_JDBC` directory.
`SQL State is null`	A database processing error has occurred and the JDBC driver did not set a valid SQL error state. The gateway cannot determine the cause of the error.	Check the database log for any indication of errors. Check that the database can still be reached. If there are no other obvious error indicators, restart the gateway. If the problem persists, contact IBM Software Support.

JDBC error messages are described in the following topic:

- "JDBC error messages" on page 54

JDBC error messages

JDBC error messages are typically generated by the JDBC driver and the content of such messages depends on the driver that you are using.

The following table describes the typical Java `ClassNotFound` exception and the three main types of JDBC errors. Consult your JDBC driver documentation for specific information about the error messages it generates.

Table 21. JDBC error messages		
Error	**Description**	**Action**
`java.lang.ClassNotFound Exception: driver class`	The JDBC driver class cannot be found.	Verify that the JDBC driver class name is correct and that the driver `.jar` files and any dependencies have been installed to the `$OMNIHOME/gates/java` directory.
Any exception thrown by the `GWJdbcNullPrepared Statement` class. Usually containing the following error message: `java.sql.SQLException: Simulated transaction error`.	This indicates that the **`Gate.Jdbc.Driver`** property is empty and the gateway in using an internal test JDBC driver to discard data and simulate errors.	Correctly configuring the **`Gate.Jdbc.Driver`** property with the correct driver class name.

Table 21. JDBC error messages (continued)		
Error	**Description**	**Action**
JDBC authentication errors	Authentication errors are usually caused by incorrect or invalid user names or passwords.	Verify that you are using correct and valid user names and passwords for the **Gate.Jdbc.Username** and **Gate.Jdbc.Password** properties, and wherever else authentication is required.
JDBC connection errors	Connection errors are typically caused by incorrect or invalid connection string parameters, such as an invalid host name or database name.	Verify that you are using correct and valid connection string parameters for the **Gate.Jdbc.Url** property.
JDBC runtime errors	Runtime errors have various causes. A typical error of this type is a constraint violation. These usually occur when the gateway is performing the initial resynchronization.	Check the exception text for the cause of the error. Constraint violations indicate that the gateway is trying to insert duplicate data into the database. Because the data is already in the database, this is not a problem. If the cause of the problem is not clear from the error message, contact IBM Software Support.

Running the gateway

You can start the gateway from the command line or run it as a Windows service.

To start the gateway on UNIX and Linux operating systems, run the following command:

```
$OMNIHOME/bin/nco_g_jdbc
```

To start the gateway on Windows operating systems, run the following command:

```
%OMNIHOME%\bin\nco_g_jdbc.exe
```

For information about putting the gateway under process control, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.

Running the gateway as a Windows service

To run the gateway as a Windows service, use the following steps:

1. To run the gateway on the same host as the ObjectServer, use the following command to register it as a service:

   ```
   %OMNIHOME%\bin\nco_g_jdbc.exe -install -depend NCOObjectServer
   ```

2. To run the gateway on a different host to the ObjectServer, use the following command to register it as a service:

   ```
   %OMNIHOME%\bin\nco_g_jdbc.exe -install
   ```

3. Start the gateway using the Microsoft Services Management Console.

Known issues

At the time of release, some issues were reported that you should be aware of when running the gateway.

This section covers the following known issues:

Gateway core dump when shutting it down

A slow connection can cause problems when the gateway is shutting down.

When shutting down, the gateway waits up to 30 seconds for database connections to finish processing. If any connection is slow, and the gateway times out waiting for the connection to close, there is a small possibility of a crash if that connection subsequently logs any messages to the log file.

Custom table labelling

The gateway and ObjectServer relationship is unable to handle labelling a custom table or a generic table as the status table.

A custom table cannot be treated as the status table because the gateway and ObjectServer relationship is unable to handle it.

Labelling a custom status table as a generic table is also not supported by the gateway, as it depends on a status table being marked as a status table by the core gateway libraries.

Note : The **Gate.RdrWtr.StatusTableName** property should not be used to specify a custom table or a generic table as a status table.

SQL error states

When the JDBC driver that you are using does not indicate a valid SQL error state for an error, the gateway will make the best effort it can to recover.

In cases where the gateway cannot recover from such an error, data will be discarded and lost. The following is an example of an error that is logged when the gateway cannot determine a valid SQL error state:

```
11/26/10 14:28:20: Error: E-GJA-000-000: [ngjava]: G_JDBC:
pool-1-thread-2: SQL State is null
```

Sybase naming schemes

The names of Sybase database objects, such as tables and columns, are case-sensitive. You might need to review your database schema to ensure that your naming scheme is compatible with your gateway configuration.

Your naming scheme for database objects can affect all columns in your maps, as well as gateway-specific properties such as **Gate.Jdbc.ServerSerialField** and **Gate.Jdbc.StatusTableName**.

Frequently asked questions

Various questions arise as users work with the JDBC Gateway. The following answer regarding how the gateway transfers the status of an event in a single gateway period is provided for your reference.

How does the gateway transfer the status of an event in a single gateway period?

The gateway can transfer only the last status of an event in a single gateway period.

For example, when an event is raised and cleared, and the **Generic Clear** and **Delete** automations run on the same event within a single 60 second granularity period, a single delete message is recorded on the external database. The result is a single delete message in the external database.

Appendix A. Notices and Trademarks

This appendix contains the following sections:

- Notices
- Trademarks

Notices

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

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

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

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

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

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

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

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

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

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

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

IBM Corporation
Software Interoperability Coordinator, Department 49XA

3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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

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

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

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

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

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

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

Trademarks

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

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

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

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

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

Part Number:

(1P) P/N:

SC22-5408-11