# IBM® Tivoli® Netcool/OMNIbus Java Gateway for ServiceNow 4.0

*Reference Guide*
*December 11, 2020*

SC27-8720-05

**Notice**

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 69.

**Edition notice**

This edition (SC27-8720-05) applies to version 4.0 of IBM Tivoli Netcool/OMNIbus Java Gateway for ServiceNow and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces (SC27-8720-04).

# Contents

**About this guide..... v**
Document Control Page..... v
Conventions used in this guide..... vi

**Chapter 1. Java Gateway for ServiceNow..... 1**
Summary..... 1
Overview..... 3
Features of the IBM Tivoli Netcool/OMNIbus Java Gateway for ServiceNow..... 3
Installing the gateway..... 4
Installing probes and gateways on Tivoli Netcool/OMNIbus V8.1..... 4
Configuring the gateway..... 6
Authentication..... 7
Enabling the gateway to read alerts from the ObjectServer..... 8
Importing custom CA certificates..... 8
Connecting to ServiceNow..... 9
Using the environment file to define Java options..... 10
Using the alert functions defined in servicenow.notification.js..... 11
Using the functions defined in servicenow.generic.js..... 15
Using the gateway log manager service to manage log messages..... 16
Table Replication..... 19
Mapping..... 21
Historic journal forwarding..... 25
Store and forward capability..... 25
FIPS mode and encryption..... 25
AES encryption..... 26
ServiceNow encryption..... 29
Configuring the PollingAdjustment property..... 29
Identifying ServiceNow ticket reclaim candidates..... 30
ServiceNow incident resolution..... 31
Gateway operation..... 33
Controlling synchronization between the gateway and ServiceNow..... 35
Controlling which ServiceNow target table the gateway uses..... 35
Controlling how the gateway handles the alternate index policy..... 36
Controlling how the gateway handles outbound data management..... 37
Controlling how the gateway handles first notifications of created tickets..... 38
Controlling where the gateway stores error codes..... 38
Controlling how the gateway performs checks before it creates a ticket..... 39
Controlling how the gateway handles processing of ServiceNow tickets..... 40
Controlling how the gateway handles operations related to ServiceNow ticket fields..... 40
Controlling how the gateway handles resynchronization..... 41
Running the gateway..... 41
Properties and command line options..... 43
Error messages..... 64
GatewayWatch messages..... 68

**Appendix A. Notices and Trademarks..... 69**
Notices..... 69
Trademarks..... 70

# 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 Java Gateway for ServiceNow documentation is provided in softcopy format only. To obtain the most recent version, please 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 |
|---|---|---|
| SC27-8720-00 | March 10, 2016 | First IBM publication. |
| SC27-8720-01 | June 10, 2016 | Updates were made to the following sections:<br>"Overview" on page 3<br>"Features of the IBM Tivoli Netcool/OMNIbus Java Gateway for ServiceNow" on page 3<br>"Authentication" on page 7<br>"Enabling the gateway to read alerts from the ObjectServer" on page 8<br>"Connecting to ServiceNow" on page 9<br>"Using the alert functions defined in servicenow.notification.js" on page 11<br>"Table Replication" on page 19<br>"Mapping" on page 21<br>"Historic journal forwarding" on page 25 |
| SC27-8720-02 | March 14, 2017 | Updates were made to the following sections:<br>Identifying ServiceNow ticket reclaim candidates on page 36 added.<br>Properties and command line options updated on page 46 to describe the following new property:<br>**`Gate.ServiceNow.Proxy`**<br>This version addresses the following enhancement requests:<br>• RFE 85941: Enhancement to add HTTP proxy support. New property updated to specify the HTTP proxy with optional port.<br>• RFE 88118: Enhancement to allow the gateway to accommodate a change in REST API behavior between ServiceNow versions Eureka/Fiji compared to Geneva. |

| *Table 1. Document modification history (continued)* | | |
|---|---|---|
| **Document version** | **Publication date** | **Comments** |
| SC27-8720-03 | December 14, 2017 | This version addresses APAR IJ01322.<br>"Importing custom CA certificates " on page 8 added. |
| SC27-8720-04 | February 28, 2019 | This version addresses **APAR IV99584**: Log file rolling and management.<br>Description for the following properties were added to "Properties and command line options" on page 43:<br>• **MessageLogRollKeep**<br>• **MessageLogRollProcess** |
| SC27-8720-05 | December 11, 2020 | Updated for Version 4.0.<br>"Summary" on page 1 updated.<br>The gateway now supports the REST v2 Table API.<br>Description for the **Gate.ServiceNow.NoRetryCodes** property added to "Properties and command line options" on page 43.<br>Functionality for providing ServiceNow Incident Resolution data added. See "ServiceNow incident resolution" on page 31.<br>Version 4.0 also addresses the following APARs:<br>• **IJ22756**: Configure the gateway to not retry queries against ServiceNow that return certain HTTP response codes (for example, 404).<br>• **IJ00363**: Remove fields marked INTERNAL ONLY from journal entries, as they shouldn't be forwarded to ServiceNow.<br>• **IV92988**: Handle malformed alternate index values without throwing an exception. |

# 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. Java Gateway for ServiceNow

The IBM Tivoli Netcool/OMNIbus Java Gateway for ServiceNow works with ServiceNow, which is a platform-as-a-service (PaaS) provider of enterprise Service Management (SM) software. In addition to providing ITSM (IT Service Management) applications, ServiceNow provides a PaaS for forms-based workflow application development. The ServiceNow platform automates common business processes.

The Java Gateway for ServiceNow is a bidirectional gateway that operates with the ServiceNow platform in the following IT operations management areas:

- Automated ticket creation
- Ticket life cycle monitoring
- Ticket management

This guide contains the following sections:


- "Summary" on page 1
- "Overview" on page 3
- "Features of the IBM Tivoli Netcool/OMNIbus Java Gateway for ServiceNow" on page 3
- "Installing the gateway" on page 4
- "Configuring the gateway" on page 6
- "Gateway operation" on page 33
- "Running the gateway" on page 41
- "Properties and command line options" on page 43
- "Error messages" on page 64
- "GatewayWatch messages" on page 68


## Summary

Use this summary information to learn about the Java Gateway for ServiceNow.

The following table provides a summary of the gateway:

*Table 3. Summary*

| | |
|---|---|
| Gateway target | The gateway target system is ServiceNow.<br>**Note :** The gateway has been certified against ServiceNow from versions Istanbul to Paris. |
| Gateway target access API | The gateway uses the REST (REpresentational State Transfer) API that ServiceNow provides.<br>Specifically, the gateway supports ServiceNow REST v2 Table API. The REST URIs include the following path: `/api/now/v2/table`. |
| Gateway jar file | `$OMNIHOME/gates/java/nco_g_servicenow.jar` |
| Command to execute the gateway | `$OMNIHOME/bin/nco_g_servicenow` |
| Additional gateway jar files | Additional gateway jar files are installed in the following directory: `$OMNIHOME/gates/java` |
| Package Version | 4.0 |

| *Table 3. Summary (continued)* | |
|---|---|
| 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=swg21978428 |
| Configuration files | Map definition file:<br>`$OMNIHOME/gates/servicenow/servicenow.map`<br>Properties file:<br>`$OMNIHOME/gates/servicenow/G_SERVICENOW.props`<br>Table replication definition file:<br>`$OMNIHOME/gates/servicenow/servicenow.rdrwtr.tblrep.def`<br>JavaScript files:<br>`$OMNIHOME/gates/servicenow/servicenow.notification.js`<br>`$OMNIHOME/gates/servicenow/servicenow.generic.js`<br>Environment (`.env`) file:<br>`$OMNIHOME/gates/servicenow/nco_g_servicenow.env`<br>SQL file:<br>`$OMNIHOME/gates/servicenow/servicenow.sql` |
| Requirements | A currently supported version of IBM Tivoli Netcool/OMNIbus.<br>Java runtime environment (JRE) 1.8.0 or later (32-bit or 64-bit)<br>The following jar files from Tivoli Netcool/OMNIbus:<br>• `$OMNIHOME/java/jars/jconn3.jar` (Sybase JConnect JDBC driver)<br>• `$OMNIHOME/java/jars/niduc.jar`<br>**Note :** The previously listed jar files and driver require the installation of the 'NCODesktop' feature in Tivoli Netcool/OMNIbus versions prior to version 8.1. The previously listed jar files require the installation of the 'GatewaySupport' feature in Tivoli Netcool/OMNIbus version 8.1.<br>The following packages are provided with the gateway:<br>• `common-libibmjuno-2`<br>• `common-jnetcool-7`<br>• `gateway-nco-g-java-10` |
| Remote Connectivity | Available |
| SSL support | Available |
| Multicultural support | Available<br>For information about configuring multicultural support, including language options, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*. |

| *Table 3. Summary (continued)* | |
|---|---|
| IP environment | IPv4 and IPv6 |
| Federal Information Processing Standards (FIPS) | IBM Tivoli Netcool/OMNIbus uses the FIPS 140-2 approved cryptographic provider: IBM Crypto for C (ICC) certificate 384 for cryptography. This certificate is listed on the NIST website at http://csrc.nist.gov/groups/STM/cmvp/documents/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*.<br>**Note :** FIPS is not supported on the target system. |

# Overview

The Gateway for ServiceNow is a bidirectional gateway that creates tickets in ServiceNow from alerts present in the ObjectServer.

In a typical environment, users define workflows within ServiceNow to perform actions on the tickets that the gateway creates. These tickets are updated, according to a predefined mapping, throughout the life-span of the alert. For each ticket that the gateway creates, ServiceNow sends back an associated unique ServiceNow system ID.

ServiceNow also passes back to the gateway any changes to the tickets so that the gateway can update the original ObjectServer alert.

The following list describes the functions that the gateway supports:

- Flexible event filtering that allows you to specify which alerts the gateway uses to create the corresponding tickets in ServiceNow.
- Reporting within Tivoli Netcool/OMNIbus of ServiceNow operation failures.
- Forwarding updates to ServiceNow tickets when the associated events in Tivoli Netcool/OMNIbus are updated or deleted.
- The ability for ServiceNow to pass back any changes to the tickets, so that the gateway can update the original ObjectServer alert.
- The gateway can forward journal entries from alerts in Tivoli Netcool/OMNIbus to ServiceNow, when the ticket table supports a journaling mechanism.
- Conversion of ObjectServer values and ServiceNow values. These conversions are bidirectional.
- The gateway can perform unidirectional resynchronization.
- The gateway provides JavaScript files for processing bidirectional updates (that is, updates from ServiceNow to the ObjectServer).
- The gateway provides a store and forward capability to minimize any data loss should the gateway lose communication with ServiceNow or the ObjectServer.

# Features of the IBM Tivoli Netcool/OMNIbus Java Gateway for ServiceNow

The IBM Tivoli Netcool/OMNIbus Java Gateway for ServiceNow has various features that enable you to create an interface between ServiceNow and the ObjectServer.

## Event forwarding

The gateway uses the Insert, Delete, Update, or Control (IDUC) communication protocol to retrieve events from ObjectServer tables (specifically, alerts.status and alerts.journal). The gateway can replicate the data from those tables to the destination server. Details of the tables to be replicated are stored in the

table replication definition file. The events retrieved from these tables are based on the table replication definition file configuration, including their filtering. Retrieved events are then passed through a mapper to assign values to target fields that the gateway is required to populate and update on the ServiceNow system. The mappers are specified in the table replication definition file and defined in the map definition file.

"Table Replication" on page 19 contains more information on replicating data between the ObjectServer and ServiceNow.

See the *IBM Tivoli Netcool/OMNIbus Administration Guide* and the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for more information on IDUC.

### Mapping table data

The gateway writes the alerts received from the various tables in the ObjectServer onto ServiceNow in a format defined by the mappers defined in the `servicenow.map` map definition file. Each mapping in the different maps defines how to map alert data to a field within the ticket table of the ServiceNow target system.

"Mapping" on page 21 contains more information on the `servicenow.map` map definition file.

### Unidirectional resynchronization

The gateway can perform unidirectional resynchronization.

"Gateway operation" on page 33 provides additional information about unidirectional resynchronization.

# 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**.

# Configuring the gateway

After installing the gateway you need to make various configuration settings to suit your environment.

Table 4 on page 6 lists gateway configuration tasks. For each configuration task, the table lists the properties you use with that task, and the section in this guide that shows you how to complete the configuration task.

Some configuration tasks are mandatory for all installations. For those configuration tasks set the properties to the correct values or verify that their default values are suitable for your environment. The remaining configuration tasks are optional depending on which ones you want to use.

**Note :** The table references the following categories of properties that are defined in the `G_SERVICENOW.props` properties file:

- Gateway Framework properties
- ServiceNow gateway specific properties

For reference information and command line options on these properties, see "Properties and command line options" on page 43.

*Table 4. Configuring the gateway*

| Configuration tasks | Properties | See |
|---|---|---|
| **Required configuration tasks:** | | |
| **Authentication configuration tasks** | | |
| Authenticate the gateway with ServiceNow. | **`Gate.ServiceNow.Username`**<br>**`Gate.ServiceNow.Password`** | "Authenticating the gateway with ServiceNow" on page 7 |
| Authenticate the gateway with the ObjectServer.<br>**Note :** The gateway needs to authenticate with the ObjectServer only when the ObjectServer runs in secure mode. | **`Gate.RdrWtr.Password`**<br>**`Gate.RdrWtr.Username`** | "Authenticating the gateway with the ObjectServer" on page 8 |
| **Connection Configuration Tasks** | | |
| Enabling the gateway to read alerts from the ObjectServer | **`Gate.RdrWtr.Server`** | "Enabling the gateway to read alerts from the ObjectServer" on page 8 |

| *Table 4. Configuring the gateway (continued)* | | |
|---|---|---|
| **Configuration tasks** | **Properties** | **See** |
| Connect the gateway to the ServiceNow server by establishing the connection to ServiceNow. | **Gate.ServiceNow.Host**<br>**Gate.ServiceNow.Username**<br>**Gate.ServiceNow.Password** | “Connecting to ServiceNow” on page 9 |
| **Optional configuration tasks:** | | |
| **Table replication configuration task** | | |
| Define the tables and event types that are replicated between the ObjectServer and ServiceNow. | **Gate.RdrWtr.TblReplicate DefFile** | “Table Replication” on page 19 |
| **Data mapping configuration task** | | |
| Define how fields in ObjectServer tables map to fields in ServiceNow. | **Gate.Mapfile** | “Mapping” on page 21 |
| **Improve gateway performance configuration task** | | |
| Specify the number of simultaneous connections the gateway makes to ServiceNow. | **Gate.ServiceNow. Connections** | “Improving the performance of the gateway” on page 10 |
| **Journal updates configuration task** | | |
| Specify whether the gateway forwards historic journal information. | **Gate.HistoricResync** | “Historic journal forwarding” on page 25 |
| **Store and forward configuration task** | | |
| Controls the operation of the store and forward functionality. | **None** | “ Store and forward capability” on page 25 |

# Authentication

The gateway needs to authenticate itself with ServiceNow at all times. The gateway needs to authenticate itself with the ObjectServer only when the ObjectServer runs in secure mode.

## Authenticating the gateway with ServiceNow

It is recommended to create a service account user on ServiceNow that acts as the dedicated gateway service account for the gateway. This account should be set up for **Web Service Account** only. Accounts set up for Web Service Access cannot log into the ServiceNow user interface to perform other actions.

By using a service account you can:

- Increase security by assigning the minimum privileges required for the gateway to perform the tasks.
- Improve troubleshooting as a dedicated service account makes it easier to distinguish and trace the actions performed by the gateway in logs.

Set the gateway's **Gate.ServiceNow.Username** and **Gate.ServiceNow.Password** properties to the user name and password of the gateway's ServiceNow account.

The dedicated gateway service account created on the ServiceNow system needs to have specific roles to support required actions. The following list identifies these roles. See the ServiceNow documentation for more information on the meaning of these roles:

- itil
- personalize_dictionary
- rest_service

**Note :** The gateway needs read access to the ServiceNow data dictionary in order to inspect the field types of the target table. The personalize_dictionary role comes as default with a ServiceNow system and provides a super set of the required access rights needed to read the ServiceNow data dictionary.

**Note :** The **Gate.ServiceNow.Username** and **Gate.ServiceNow.Password** properties must be unique for each gateway instance.

### Authenticating the gateway with the ObjectServer

When the ObjectServer runs in secure mode, it requires each gateway that connects to it to supply a user name and password. Set the gateway's **Gate.RdrWtr.Username** and **Gate.RdrWtr.Password** to the user name and password of the gateway's ObjectServer account.

**Note :** A gateway user account needs to be created within Tivoli Netcool/OMNIbus so that the gateway can supply the username and password to identify itself to the ObjectServer.

## Enabling the gateway to read alerts from the ObjectServer

To enable the gateway to read alerts from the ObjectServer, use the **Gate.RdrWtr.Server** property.

The **Gate.RdrWtr.Server** property specifies the name of the ObjectServer from which the gateway reads alerts. The name can either be an interface name (for example, NCOMS) or the <host>:<port> details of the ObjectServer.

## Importing custom CA certificates

Importing a custom CA certificate entails exporting the CA root certificate (AEP CA) from the Firefox browser and then importing that certificate into the JRE cacerts file. Once imported, the JRE can use the certificate to validate a TLS connection.

To import a custom CA certificate, use the following steps:

1. Export the CA root certificate from the Firefox browser.
   a. Open the SN page in the browser.
   b. Right-click on the page and select **View Page Info**.

      Firefox will display a dialog box with the page information.
   c. Select the **Security** icon tab.
   d. Click on the **View Certificate** button on the right of the dialog box.

      Firefox will display another dialog box.
   e. Select the **Details** tab.
   f. In **Certificate Hierarchy**, select the root certificate at the top of the certificate tree hierarchy.
   g. Click the **Export...** button to export the selected certificate to a file.
   h. Accept the default name and format (PEM).
   i. Make a note of the name used.
2. Import this certificate into the JRE cacerts file.
   a. From a command line, run the following command:

```
cd $NCHOME/platform/linux2x86/jre64_1.8.0/jre/bin
```

This directory contains keytool, the tool that you will use to import the certificate into the cacerts file.

b. Run the following command:

```
./keytool -import -file cert.crt -keystore
```

Where cert.crt is the the name you specified in Step 1.8.

c. Run the following command:

```
../lib/security/cacerts -storepass "changeit"
```

Where `changeit` is the store password.

keytool will display the certificate details and prompt you to add the certificate.

d. Enter Y (yes).

With cacerts updated to include the AEP CA certificate, the TLS negotiation can now verify the server, the connection should complete, and allow the gateway to run.

# Connecting to ServiceNow

The connection sequence that the gateway uses has two phases: establishing the connection to ServiceNow and validating the data.

## Establishing the connection to ServiceNow

At start up, the gateway sends an HTTP GET request on the ServiceNow target table defined by the **Gate.ServiceNow.TableName** property and validates the table. In the same action, the gateway validates the connection details to ServiceNow.

Use the following properties defined in the `G_SERVICENOW.props` properties file to establish connection to ServiceNow:

- Set the **Gate.ServiceNow.Host** property to the name (or IP address) of the server on which ServiceNow is running and to which the gateway connects. The gateway will connect to this ServiceNow server to create tickets and to read updates from it.
- Set the **Gate.ServiceNow.Username** property to the username for the user account that the gateway uses to access the ServiceNow instance. This user account must be created in ServiceNow and consist of a username and password. Use this property in conjunction with the **Gate.ServiceNow.Password** property.
- Set the **Gate.ServiceNow.Password** property to the password for the user account that the gateway uses to access the ServiceNow instance. This user account must be created in ServiceNow and consist of a username and password. Use this property in conjunction with the **Gate.ServiceNow.Username** property.

“Authentication” on page 7 contains information on creating a dedicated gateway user on the ServiceNow system and using those credentials for specifying the user name and password for the **Gate.ServiceNow.Username** and **Gate.ServiceNow.Password** properties.

When the gateway starts, it checks the previously described properties and, if valid, uses them as part of every REST request issued to the target ServiceNow instance. If one or more of the previously described properties has invalid values or is empty, the gateway shuts down and writes an appropriate message to the log.

If the gateway starts up successfully and subsequently loses its connection with ServiceNow, the gateway goes into store and forward mode.

See “ Store and forward capability” on page 25 for information on the store and forward feature.

### Validating the gateway properties file

When the gateway starts, but before it connects to the ServiceNow server, it validates the gateway properties file, `G_SERVICENOW.props`, to ensure that each property has the correct syntax and correct value type. After the gateway successfully connects to the ServiceNow server, it performs a number of checking operations including the following:

- Validates the mapping file as defined in the **`Gate.MapFile`** property. Specifically, the gateway:
  - Ensures that the mapping file contains at least the status map (`StatusMap`). If the status map is not present, the gateway shuts down.
  - Checks that every ServiceNow field residing in all maps is a valid ServiceNow field name.
- Validates a number of attribute names with ServiceNow. These attribute names are used to set gateway specific properties, for example, **`Gate.ServiceNow.UpdateFields`** and **`Gate.ServiceNow.AlternateIndexField`**.

### Improving the performance of the gateway

To improve the performance of the gateway use the **`Gate.ServiceNow.Connections`** property (defined in the `G_SERVICENOW.props` properties file). The **`Gate.ServiceNow.Connections`** property takes an integer value that specifies the number of simultaneous connections the gateway makes with ServiceNow. In general, the more connections specified, the faster the gateway can create requests in ServiceNow from alerts sent by the ObjectServer.

In the following gateway properties file example, the **`Gate.ServiceNow.Connections`** property is set to the value 6, which means the gateway will make six simultaneous connections with ServiceNow:

```
.
.
.
# ServiceNow gateway specific properties
.
.
.
Gate.ServiceNow.Connections:6
.
.
.
```

**Note :** If specified, the value of the **`Gate.ServiceNow.Connections`** property should be greater than 2. Also, the underlying hardware architecture must align with the number of connections.

## Using the environment file to define Java options

On Linux, the `nco_g_servicenow.env` environment file to define Java options that you want the Java Gateway for ServiceNow to pick up at run time. On Windows, the `nco_g_servicenow.bat` script defines the Java options for the gateway.

### Overview

The *JRE_OPTS* environment variable defined in the `nco_g_servicenow.env` file defines Java options specific to the Java Gateway for ServiceNow. The *JRE_OPTS* environment variable is supplied with Java options that define three system properties. At start-up, the gateway picks up the contents of the *JRE_OPTS* environment variable.

The following sections discuss in more detail the `nco_g_servicenow.env` file and the three defined system properties.

### The nco_g_servicenow.env file

The `nco_g_servicenow.env` file supplied with the gateway resides in the `$OMNIHOME/gates/servicenow` directory and contains an environment variable called *JRE_OPTS*. The *JRE_OPTS*

environment variable is supplied with the following Java options that make use of the -D switch to define the following system properties that get executed at run time:

- **org.apache.commons.logging.Log** - This system property is set to the value of org.apache.commons.logging.impl.SimpleLog, which identifies the logging mechanism that the gateway uses. This logging mechanism is the Simple Log package that is distributed under The Apache Software Foundation License.
- **org.apache.commons.logging.simplelog.showdatetime** - This system property is set to the value true to include the current date and time in output messages. The default value is true, which includes the current date and time in output messages.
- **org.apache.commons.logging.simplelog.log.org.apache.http** - This system property specifies the logging detail level for a SimpleLog instance named log.org.apache.http. In this case, the value WARN indicates that the logging detail level used is warn.
- **com.ibm.jsse2.overrideDefaultTLS** - This system property instructs the IBM Java Secure Socket Extension to match the behavior of SSLContext.getInstance("TLS") in the IBM SDK with the Oracle implementation. Setting this to true enables the IBM Java to enable TLSv1.2, 1.1 and 1.0 for secure communications.

To define additional Java options specific to the gateway, add them to the *JRE_OPTS* environment variable in the nco_g_servicenow.env file.

### The nco_g_servicenow.bat script

The nco_g_servicenow.bat script is the Windows equivalent of the nco_g_servicenow.bat. The script resides in the $OMNIHOME/gates/win32 directory and defines the GW_JRE_OPTS environment variable which sets the system properties mentioned above. To define additional Java options for the gateway on Windows, add them to the GW_JRE_OPTS variable in the nco_g_servicenow.bat script.

## Using the alert functions defined in servicenow.notification.js

Use the alert functions defined in the servicenow.notification.js file to perform a variety of operations on Tivoli Netcool/OMNIbus ObjectServer alerts. The alert functions are called by the Java Gateway for ServiceNow.

### Overview

The servicenow.notification.js file defines a variety of alert functions that operate on ObjectServer alerts and the journal table (default value alerts.journal). The file also imports functionality from several modules. The alert functions supplied in the servicenow.notification.js file define a basic default behavior for each of those functions. The servicenow.notification.js file is written in JavaScript and you can modify these alert functions to suit your requirements.

The following sections discuss each of the imported modules and alert functions defined in the servicenow.notification.js file.

### Imported modules

The servicenow.notification.js file uses the require function to load the following modules:

- Gateway log manager service -- Made available through the *logger* variable
- Simple OMNIbus Gateway (SOG) interface module -- Made available through the *sog* variable
- dateformat module -- Made available through the *dateformat* variable

The SOG interface module exposes the following method:

- newrow -- Returns an empty row that is ready to be populated.

## The addJournal function

The gateway calls the `addJournal` function to add a journal entry in the ObjectServer `alerts.journal` table for the ObjectServer alert specified in *alert*.

The `addJournal` function is defined as follows:

```
function addJournal(alert, message)
```

- *alert* -- Specifies the ObjectServer alert for which you want to add a journal entry in the `alerts.journal` table.
- *message* -- Specifies a message to be concatenated after the following string:

```
Generated by ServiceNow Gateway: message
```

## The logDebug function

The gateway calls the `logDebug` function to log debug messages associated with the ObjectServer alert specified in *alert*. The `logDebug` function actually calls the `debug` method, which is implemented as part of the Tivoli Netcool/OMNIbus `Logger` interface. The gateway writes debug messages to the gateway log file.

The `logDebug` function is defined as follows:

```
function logDebug(alert, msg)
```

- *alert* -- Specifies the ObjectServer alert to which the debug message specified in *msg* applies.
- *msg* -- Specifies a message to be concatenated after the following string:

```
Alert [ server_name, server_serial ] : msg
```

Where:

- *server_name* -- Specifies the name of the server on which the ObjectServer alert was generated.
- *server_serial* -- Specifies the serial number of the alert in the `alerts.status` table.

## The logError function

The gateway calls the `logError` function to log errors associated with the ObjectServer alert specified in *alert*. The `logError` function actually calls the `error` method, which is implemented as part of the Tivoli Netcool/OMNIbus `Logger` interface. The gateway writes errors to the gateway log file.

The `logError` function is defined as follows:

```
function logError(alert, msg)
```

- *alert* -- Specifies the ObjectServer alert to which the error message specified in *msg* applies.
- *msg* -- Specifies a message to be concatenated after the following string:

```
Alert [ServerName, ServerSerial]: msg
```

Where:

- *server_name* -- Specifies the name of the server on which the ObjectServer alert was generated.
- *server_serial* -- Specifies the serial number of the alert in the `alerts.status` table.

## The update functions

The `servicenow.notification.js` file defines three update functions:

- update -- The gateway calls the update function whenever it:
  - Receives an IDUC (specifically, an Insert or Update) notification from the ObjectServer
  - Polls ServiceNow and needs to perform an update operation on an alert

  If the gateway receives an IDUC (specifically, an Insert or Update) notification from the ObjectServer, the update function provides the entry point to invoke the internalUpdate function. If the gateway performs an update operation on an alert following a ticket modification in ServiceNow, the update function provides the entry point to invoke the inboundUpdate function. Thus, in effect, the update function is a wrapper function that calls either the internalUpdate or the inboundUpdate function.
- internalUpdate -- Processes the IDUC (specifically, the Insert or Update) notification from the ObjectServer.
- inboundUpdate -- The update function calls the inboundUpdate function as a result of the gateway performing an update operation on an alert following a ticket modification in ServiceNow.

The following sections describe each of these functions.

### *The update function*

The gateway calls the update function whenever it:

- Receives an IDUC (specifically, an Insert or Update) notification from the ObjectServer
- Polls ServiceNow and needs to perform an update operation on an alert

The update function is defined as follows:

```
function update(alert, inputs)
```

- *alert* -- Specifies the alert within Tivoli Netcool/OMNIbus that is being updated.
- *inputs* -- Determines which update function to call. If *inputs* contains the value ServiceNowErrorCode, then the gateway knows that a set of data was provided in the IDUC notification from the ObjectServer. The update function then calls the internalUpdate function to process the IDUC (specifically, the Insert or Update) notification.

  Otherwise, if the *inputs* parameter does not contain the value ServiceNowErrorCode, then the gateway knows that this is an update operation resulting from the gateway reading a ticket update in ServiceNow. In this case, the update function calls the inboundUpdate function to perform the update operation.

### *The internalUpdate function*

The update function calls the internalUpdate function as a result of the processing of the IDUC (Insert or Update) notification in the main gateway code. More specifically, the internalUpdate function sets the ServiceNowErrorCode value in the ObjectServer alerts.status table when it gets an error trying to perform an action on ServiceNow. For example, the gateway would update the ServiceNowErrorCode in the ObjectServer if it tried to update a ticket in ServiceNow and received an error back indicating that the ticket is not there anymore. In this example, the gateway would update the ServiceNowErrorCode to the value 2 (unrecoverable error) in the alerts.status table.

The internalUpdate function is defined as follows:

```
function internalUpdate(alert, inputs)
```

- *alert* -- Specifies the ObjectServer alert that is being updated with a ServiceNow error code.
- *inputs* -- Specifies a name value pair as follows:
  - The name of the ObjectServer column in the alerts.status table to be used for storing ServiceNow error codes. The default name of this column is ServiceNowErrorCode.
  - The ServiceNow error code to be written to the specified column in the alerts.status table.

This is the example provided in the `servicenow.notification.js` file. The function can handle other items.

**Note :**

The reason that the `internalUpdate` function creates a new row (by calling the `sog.newrow` method) is to set only the `ServiceNowErrorCode` and not set all the other variables that might be present in the *inputs* parameter.

The `internalUpdate` function calls the `logDebug` function to log the following debug message in the gateway log file:

```
Alert [ server_name, server_serial ] :
       updating with ServiceNowErrorCode ServiceNowErrorCode
```

Where:

- *server_name* -- Specifies the name of the server on which the ObjectServer alert was generated.
- *server_serial* -- Specifies the serial number of the alert in the `alerts.status` table.
- *ServiceNowErrorCode* -- Specifies a ServiceNow error code. For example: error code 1 (`TRANSIENT_ERROR`) and error code 2 (`NON_RECOVERABLE_ERROR`).

The `internalUpdate` function calls the `logError` function to log the following error message in the gateway log file if the gateway cannot update the ObjectServer alert specified in *alert*:

```
Alert [ServerName, ServerSerial]:
       Unable to update alert in OMNIbus: err
```

- *err* -- Specifies a message indicating why the gateway could not update the ObjectServer alert specified in *alert* with the ServiceNow error code.

### *The inboundUpdate function*

The gateway calls the `inboundUpdate` function to process an update as a result of a ticket modification on ServiceNow. Note that the gateway does not receive notifications from ServiceNow. Instead, the gateway regularly polls ServiceNow for updates. An update is the result of a ticket modification in ServiceNow.

**Note :** You specify the frequency with which the gateway polls ServiceNow by specifying an appropriate value in the **`Gate.ServiceNow.PollingInterval`** property defined in the `G_SERVICENOW.props` properties file.

The `inboundUpdate` function is defined as follows:

```
function inboundUpdate(alert, inputs)
```

- *alert* -- Specifies the alert within Tivoli Netcool/OMNIbus that is being updated.
- *inputs* -- Specifies the actual update from the target system (in this case, ServiceNow). This parameter takes a name/value map of updated values. These names are in the target system namespace, which might not correspond to the names of columns in the Tivoli Netcool/OMNIbus, so values need to be converted from one namespace to the other. This requires creating an empty update row (by calling the `sog.newrow` method).

**Note :** It is possible to customize the `inboundUpdate` function to provide logic to cater for a ticket state being set to `Resolved`.

## The clear function

The gateway does not currently make use of the `clear` function. The main reason is that when a ticket gets closed or deleted in ServiceNow, the gateway does not get notified. By default, the only way for the gateway to discover when a ticket gets closed or deleted in ServiceNow is when the gateway tries to update a ticket and gets back an error indicating that the ticket cannot be found.

### The error function

The gateway calls the `error` function whenever an error occurs when the gateway tries to manipulate the corresponding ticket in the target system (in this case, ServiceNow). The `error` function is defined as follows:

```
function error(alert, inputs, message)
```

- *alert* -- Specifies the alert within Tivoli Netcool/OMNIbus.
- *inputs* -- Specifies the actual values from the ServiceNow ticket.
- *message* -- Specifies the error message text. This error message text can be added to a journal entry for the alert (by calling the `addJournal` method), for example, to identify the problem to the end user.

### The setTargetId function

The gateway calls the `setTargetId` function whenever the alert is associated with a destination target system ticket. Typically, the gateway calls `setTargetId` after the user creates a ServiceNow ticket. The `setTargetId` function sets the target ID in the underlying alert and adds the target ID details to a journal entry for the alert (by calling the `addJournal` method).

The `setTargetId` function is defined as follows:

```
function setTargetId(alert, targetid)
```

- *alert* -- Specifies the alert within Tivoli Netcool/OMNIbus.
- *targetid* -- Specifies a string representation of the id. In the case of ServiceNow, this string is the sys_id associated with the ticket.

## Using the functions defined in servicenow.generic.js

Use the alert functions defined in the `servicenow.generic.js` script to perform customizations on Tivoli Netcool/OMNIbus ObjectServer alerts.

### Overview

The `servicenow.generic.js` script defines functions that are intended to modify processed event data from the gateway before the alert is replicated into ServiceNow. This script is written in JavaScript so JavaScript syntax and notations are to be used.

### Imported modules

The `servicenow.generic.js` file uses the `require` function to load the following module:

- Gateway log manager service: Made available through the logger variable.

### The updatePayload function

After reading the event data from the object server, the gateway calls the `updatePayload` function before an event is sent to ServiceNow. This function is intended to allow for the overriding or insertion of values into the event. The implementation provided in `servicenow.generic.js` provides values for the `close_code` and `close_notes` fields when the event `Severity` equals to `0` (zero).

The `addJournal` function is defined as follows:

```
function updatePayload(inserts, operation)
```

- *inputs* -- Specifies a map containing key-pair values of the event data.
- *operation* -- Specifies the operation applied to the event, namely: `insert`, `update` or `delete`.

# Using the gateway log manager service to manage log messages

The gateway log manager service is a static class which enables log messages to be sent to the lower level gateway architecture. Any log messages of a level higher than that set by the **MessageLevel** property in the gateway's properties file will appear in the gateway's log file.

The following topics describe how to use the gateway log manager service.

## Preparing to use the gateway log manager service

Typically, the gateway makes the gateway log manager service available through a JavaScript file. The name of the JavaScript file has the following format:

```
gatewayname.notification.js
```

Where: *gatewayname* specifies a string that maps to the name of the gateway.

The JavaScript file uses the `require` function to load the gateway log manager service and make it available through the *logger* variable, as follows:

```
.
.
.
var logger = require("logger");
.
.
.
```

The gateway log manager service exposes the following methods:

- `error` -- Logs an error message.
- `warning` -- Logs a warning message.
- `info` -- Logs an informational message.
- `debug` -- Logs a debug message.

## Logging messages of type ERROR

Use the `error` method to log messages of type ERROR into the specified gateway log file.

Using the `error` method requires you to understand:

- The definition of the `error` method
- How to call the `error` method

**Note :** Use the **MessageLevel** property to specify the level of messages to receive for this gateway. Use the **MessageLog** property to specify the path, including the name of the log file, to which the `error` method writes messages of type ERROR for this gateway.

### *Understanding the definition of the error method*

The `error` method is defined in the gateway log manager service.

The `error` method is defined as follows:

```
public void error( Object message );
```

Where:

- *message* specifies the message string to be logged.

The `error` method inserts the message specified in the *message* parameter into the log file specified in the **MessageLog** property for this gateway.

Upon completion, the `error` method returns no value.

### *Calling the error method*

You can call the `error` method in any of the gateway's source files whenever you want to insert a log message of type ERROR into the log file specified in the **MessageLog** property for this gateway.

The following example shows a call to the `error` method:

```
.
.
.
logger.error("Target application unable to find the ticket. Please
             double-check if the ticket exists in the target
             application."); 1
.
.
.
```

1. Calls the `error` method, which writes the specified message to the log file specified in the **MessageLog** property for this gateway.

## Logging messages of type WARNING

Use the `warning` method to log messages of type `WARNING` into the specified gateway log file.

Using the `warning` method requires you to understand:

- The definition of the `warning` method
- How to call the `warning` method

**Note :** Use the **MessageLevel** property to specify the level of messages to receive for this gateway. Use the **MessageLog** property to specify the path, including the name of the log file, to which the `warning` method writes messages of type `WARNING` for this gateway.

### *Understanding the definition of the warning method*

The `warning` method is defined in the gateway log manager service.

The `warning` method is defined as follows:

```
public void warning( Object message );
```

Where:

- *message* specifies the message string to be logged.

The `warning` method inserts the message specified in the *message* parameter into the log file specified in the **MessageLog** property for this gateway.

Upon completion, the `warning` method returns no value.

### *Calling the warning method*

You can call the `warning` method in any of the gateway's source files whenever you want to insert a log message of type `WARNING` into the log file specified in the **MessageLog** property for this gateway.

The following example shows a call to the `warning` method:

```
.
.
.
logger.warning("Unable to determine hostname"); 1
.
.
.
```

1. Calls the `warning` method, which writes the specified message to the log file specified in the **MessageLog** property for this gateway.

## Logging messages of type INFO

Use the `info` method to log messages of type `INFO` into the specified gateway log file.

Using the `info` method requires you to understand:

- The definition of the `info` method
- How to call the `info` method

**Note :** Use the **`MessageLevel`** property to specify the level of messages to receive for this gateway. Use the **`MessageLog`** property to specify the path, including the name of the log file, to which the `info` method writes messages of type `INFO` for this gateway.

### *Understanding the definition of the info method*

The `info` method is defined in the gateway log manager service.

The `info` method is defined as follows:

```
public void info( Object message );
```

Where:

- *message* specifies the message string to be logged.

The `info` method inserts the message specified in the *message* parameter into the log file specified in the **`MessageLog`** property for this gateway.

Upon completion, the `info` method returns no value.

### *Calling the info method*

You can call the `info` method in any of the gateway's source files whenever you want to insert a log message of type `INFO` into the log file specified in the **`MessageLog`** property for this gateway.

The following example shows a call to the `info` method:

```
.
.
.
logger.info("Ticket on the target system found:"); 1
.
.
.
```

1. Calls the `info` method, which writes the specified message to the log file specified in the **`MessageLog`** property for this gateway.

## Logging messages of type DEBUG

Use the debug method to log messages of type DEBUG into the specified gateway log file.

Using the debug method requires you to understand:

- The definition of the debug method
- How to call the debug method

**Note :** Use the **`MessageLevel`** property to specify the level of messages to receive for this gateway. Use the **`MessageLog`** property to specify the path, including the name of the log file, to which the debug method writes messages of type DEBUG for this gateway.

### *Understanding the definition of the debug method*

The debug method is defined in the gateway log manager service.

The debug method is defined as follows:

```
public void debug( Object message );
```

Where:

- *message* specifies the message string to be logged.

The debug method inserts the message specified in the *message* parameter into the log file specified in the **MessageLog** property for this gateway.

Upon completion, the debug method returns no value.

### *Calling the debug method*

You can call the debug method in any of the gateway's source files whenever you want to insert a log message of type DEBUG into the log file specified in the **MessageLog** property for this gateway.

The following example shows a call to the debug method:

```
.
.
.
logger.debug("Form validated by target application"); 1
.
.
.
```

1. Calls the debug method, which writes the specified message to the log file specified in the **MessageLog** property for this gateway.

# Table Replication

Use the table replication definition file to define which tables the gateway replicates and the types of events that it monitors in the ObjectServer.

## Overview

The gateway creates and updates tickets in ServiceNow when alerts are created and updated in the ObjectServer. The gateway populates and updates the fields in the ticket by replicating data from the ObjectServer. The gateway uses a table replication definition file called `servicenow.rdrwtr.tblrep.def` to determine which ObjectServer tables to replicate. Specifically, the `servicenow.rdrwtr.tblrep.def` file specifies REPLICATE command clauses that define how the gateway replicates data between an ObjectServer and a target database or application, in this case ServiceNow.

See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for descriptions of the syntax associated with the REPLICATE command clauses. See the *IBM Tivoli Netcool/OMNIbus Administration Guide* for a description of the ObjectServer SQL interface for defining and manipulating relational database objects such as tables and views. The `servicenow.rdrwtr.tblrep.def` file makes use of the ObjectServer SQL interface. The *IBM Tivoli Netcool/OMNIbus Administration Guide* also describes the column names of the `alerts.status` and `alerts.journal` ObjectServer tables.

This topic describes the specific REPLICATE command and `FILTER WITH` clauses specified in the delivered `servicenow.rdrwtr.tblrep.def` file. The topic also provides examples of other possible `FILTER WITH` clauses.

The information that follows describes the `servicenow.rdrwtr.tblrep.def` file. See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for more information, including information on the structure of the table replication definition file, the REPLICATE command, and the available clauses.

## The table replication definition file

The table replication definition file defines which tables the gateway replicates and the types of events that it monitors in the ObjectServer. The table replication definition file also identifies the mapping between data fields in the ObjectServer and those in a ServiceNow ticket.

The table replication definition file supplied with the gateway is named `servicenow.rdrwtr.tblrep.def`, and resides in the `$OMNIHOME/gates/servicenow` directory. Use the **Gate.RdrWtr.TblReplicateDefFile** property defined in the `G_SERVICENOW.props` properties file to specify the path to (including the name of ) the table replication definition file. The default value specified in `G_SERVICENOW.props` is as follows:

```
.
.
.
# Gate.RdrWtr.TblReplicateDefFile :
'${OMNIHOME}/gates/servicenow/servicenow.rdrwtr.tblrep.def'
.
.
.
```

The table replication definition file contains one or more REPLICATE command clauses and as supplied contains the following:

```
REPLICATE ALL FROM TABLE 'alerts.status' 1
    USING MAP 'StatusMap'; 2

REPLICATE ALL FROM TABLE 'alerts.journal' 3
    USING MAP 'JournalMap'; 4
  FILTER WITH "Text1 NOT LIKE 'Generated by ServiceNow Gateway*'"; 5
```

The following descriptions explain each of the numbered items in the supplied `servicenow.rdrwtr.tblrep.def` file:

1. Specifies that the gateway replicates all insert, update, and delete operations from the source ObjectServer table (`alerts.status`) to the target table.
2. Specifies that the gateway uses the `StatusMap` defined in the `servicenow.map` map definition file to map Tivoli Netcool/OMNIbus alert fields to their corresponding ServiceNow ticket fields.

   **Note :** A separate data mapping file called `servicenow.map` contains the definitions of the data mapping.

   See “Mapping” on page 21 for details of the data mapping definitions.
3. Specifies that the gateway replicates all insert, update, and delete operations from the ObjectServer source table (`alerts.journal`) to the target table.
4. Specifies that the gateway uses the `JournalMap` defined in the `servicenow.map` map definition file to map Tivoli Netcool/OMNIbus `alerts.journal` fields to their corresponding ServiceNow ticket fields.
5. Specifies how the gateway should filter the rows that are selected for replication. The NOT LIKE operator instructs the gateway to perform a string comparison between the string `'Generated by ServiceNow Gateway*'` and the table column represented by the string `Text1`. The result is the gateway replicates all rows in the target table in which `Text1` does not contain the string `'Generated by ServiceNow Gateway*'`.

   This ensures that rows containing journal entries created by the gateway are not replicated.

The following example shows how to use a `FILTER WITH` clause to filter out updates when the `ServiceNowErrorCode` has a non zero value:

```
REPLICATE ALL FROM TABLE 'alerts.status'
     USING MAP 'StatusMap'
     FILTER WITH "Manager LIKE 'ServiceNow*' AND ServiceNowErrorCode=0";
```

## FILTER WITH examples

The `servicenow.rdrwtr.tblrep.def` table replication definition file is supplied with the following `FILTER WITH` clause:

```
FILTER WITH "Text1 NOT LIKE 'Generated by ServiceNow Gateway*'";
```

The following examples show other possible FILTER WITH clauses. Create FILTER WITH clauses that are suitable for your environment. See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for information on the FILTER WITH clause.

**Note :** When considering which characteristics of alerts to use as a filter, use those fields from the ObjectServer alerts.status table whose values do not change over time. For example, ServerName, Class, Manager, and so forth.

```
FILTER WITH 'Acknowledged>0'; 1
FILTER WITH "Text1 NOT LIKE 'GW%'" 2
FILTER WITH 'Text1 NOT LIKE \'GW%\' ' 3
FILTER WITH 'Node != \'localhost\' 4
FILTER WITH 'ServerName IN (\'NCOMS_US\',\'NCOMS_CA\')' 5
```

The following descriptions explain each of the numbered items in the FILTER WITH clause examples. The gateway uses these FILTER WITH clauses to filter the database rows that are selected for replication as follows:

1. Uses the > (greater than) comparison operator to test the ObjectServer alerts.status table field called Acknowledged to determine if the value is greater than 0 (zero). The result is the gateway selects only those alerts that have been acknowledged (that is, Acknowledged is equal to the value 1).

   **Note :** The Acknowledged field is either 0 (Not Acknowledged) or 1 (Acknowledged).
2. Uses the NOT LIKE operator to instruct the gateway to perform a string match between the pattern GW % and the table column represented by the string Text1. (The % (percent) operator is used as a wildcard in matching operations.) The result is the gateway replicates all rows in the target table in which Text1 does not contain the string GW%.
3. Uses the != (not equal to) comparison operator to test the ObjectServer alerts.status table field called Node to determine if the value is not equal to 'localhost'. The result is the gateway selects all Node alerts except those that originate from the local host.
4. Uses the IN list comparison operator to compare the ObjectServer alerts.status table field called ServerName to the list of values NCOMS_US and NCOMS_CA. The result is the gateway selects only those alerts that originate from the ObjectServers named NCOMS_US and NCOMS_CA.

   **Note :** This FILTER WITH clause shows that if a filter contains quotation marks (single or double), an escape (the \ (backslash)) character must be specified before each quotation mark to escape it.

# Mapping

Mapping defines how the Java Gateway for ServiceNow maps fields in ObjectServer tables to fields in a ServiceNow ticket. The mapping is used during replication of ObjectServer to ServiceNow data.

## The map definition file

The mapping of fields in ObjectServer tables to fields in a ServiceNow ticket is defined in the map definition file. The gateway is supplied with a map definition file named servicenow.map in the directory $OMNIHOME/gates/servicenow. You can modify this file to tailor the data mapping to suit your environment.

The map definition file contains a number of CREATE MAPPING commands each of which maps specific ObjectServer table fields to fields in a ServiceNow ticket.

The information that follows describes the servicenow.map map definition file. See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for more information, including information on the structure

of the map definition file, the CREATE MAPPING command, and the optional CREATE MAPPING command clauses.

## DEDUPLICATION

Deduplication is a mechanism by which the gateway reduces the amount of redundant updates sent to ServiceNow.

The gateway computes a cryptographic hash (SHA1) based on fields selected for deduplication in the specified map, and stores the resulting hash in the history of the alert in the gateway cache. If subsequent data results in a hash already stored in the alert history, the gateway drops this data because it exists in the cache.

The gateway maintains the main alert values (values from the ObjectServer alerts.status table) as a current hash, with no other history maintained.

The gateway maintains alert journals (from the ObjectServer alerts.journal table) as a history, with multiple hashes per alert. This allows the gateway to record multiple journals and details for each alert.

**Note :** Without a DEDUPLICATE clause, all fields in a map are considered for deduplication purposes. The DEDUPLICATE clause specifies the fields to be included in the row hash used for deduplication in the specified map. Conversely, the DO NOT DEDUPLICATE clause means ignore the listed fields when computing the row hash used for deduplication.

See "Explanation of the StatusMap" on page 23 for an example of how to use the DEDUPLICATE clause.

## The CREATE MAPPING command

The CREATE MAPPING command creates a map that defines how the gateway maps fields in ObjectServer tables to fields in a ServiceNow ticket. The CREATE MAPPING command has the following syntax:

```
CREATE MAPPING mappingname
(
'ServiceNowFieldId' = 'value' [ ON INSERT ONLY ] [ CONVERT TO type ]
[ , 'ServiceNowFieldId' = 'value' [ ON INSERT ONLY ] [ CONVERT TO type ] ]...
) ;
```

where:

- *mappingname* specifies the name of the map to be created for a specific mapping of ObjectServer table fields to fields in a ServiceNow ticket.
- '*ServiceNowFieldId*' specifies the destination field in the ServiceNow ticket.
- '*value*' specifies the name of a field in the ObjectServer table.

**Note :** It is assumed that you are familiar with the fields in the ObjectServer tables and the corresponding fields in a ServiceNow ticket.

At a minimum, the gateway must pass across the @Serial and @Severity fields to ServiceNow in order for the replication to take place.

The optional ON INSERT ONLY controls the updating of the ServiceNow field during the life of its associated ObjectServer alert; when omitted, the ServiceNow field is updated for any change in the state of the alert. When included, the ServiceNow field is only set when the alert is created.

The optional CONVERT TO *type* allows the mapping to define a forced conversion for situations where a source field may not match the type of the destination field. The *type* can be INTEGER, STRING, or DATE to force the source field to be converted to an integer, string, or date type.

## Default maps supplied with the map definition file

The servicenow.map map definition file delivered with the gateway contains the following default maps:

- StatusMap - Is the map for the ObjectServer alerts.status table, and maps Tivoli Netcool/OMNIbus alert fields to their corresponding ServiceNow ticket fields.
- JournalMap - Contains the mapping from the ObjectServer alerts.journal table entries to ServiceNow work_notes. The JournalMap also contains meta-data fields for the journal entry (user and timestamp of the journal entry in Tivoli Netcool/OMNIbus) for the purpose of deduplication.

**Note :** The map definition file (servicenow.map) defines the maps, but does not determine whether the maps are used. You specify which maps to use in the table replication definition file (servicenow.rdrwtr.tblrep.def).

See "Table Replication" on page 19 for more information.

## Explanation of the StatusMap

The StatusMap is one of the maps contained in the servicenow.map map definition file delivered with the gateway. As discussed previously, the StatusMap is the main map for ObjectServer alerts.status table entries, and maps Tivoli Netcool/OMNIbus alert fields to their corresponding ServiceNow ticket fields. The StatusMap is defined as follows:

```
CREATE MAPPING StatusMap 1
(
    'opened_at' = '@FirstOccurrence' CONVERT TO DATE ON 2
INSERT ONLY,
'short_description' = '@Summary' ON INSERT ONLY, 3
'state'             = Lookup(StateTable, coalesce
                      ('@Severity','1')),4
'impact'            = Lookup(ImpactTable, '@Severity') ON 4
INSERT,UPDATE ONLY,
'urgency'           = Lookup(UrgencyTable, '@Severity') ON 4
INSERT,UPDATE ONLY,
'sys_id'            = '@ServiceNowSysId' INTERNAL ONLY, # DO
NOT UPDATE OR REMOVE 5
severity'          = '@Severity' INTERNAL ONLY,5
'lastoccurrence'    = '@LastOccurrence' INTERNAL ONLY,5
'tally'             = '@Tally' INTERNAL ONLY, 5
'acknowledged'      = '@Acknowledged' INTERNAL ONLY, 5
'serverName'        = '@ServerName' INTERNAL ONLY, # DO NOT
UPDATE OR REMOVE 5
'serverSerial'      = '@ServerSerial' INTERNAL ONLY, # DO NOT
UPDATE OR REMOVE 5
'work_notes'        = 'Alert deleted in the Object Server' ON
DELETE ONLY, 6
) DEDUPLICATE
('severity','lastoccurrence','tally','acknowledged'); 7
```

The following list describes each of the numbered items in the StatusMap:

1. Uses the CREATE MAPPING command to create a map called StatusMap. As described previously, the StatusMap is the main map for ObjectServer alerts.status table entries, and maps Tivoli Netcool/OMNIbus alert fields to their corresponding ServiceNow ticket fields.
2. Maps the ServiceNow field identifier (the string value opened_at) to the ObjectServer table field called FirstOccurrence.

   The string value on the right side of the expression in each mapping clause refers to a field in the ObjectServer table. The string values on the right side of the expressions use the format @fieldname. For example, in this mapping clause @FirstOccurrence is used.

   The use of ON INSERT ONLY instructs the gateway to set the 'opened_at' field of the ServiceNow ticket only when the gateway creates the ticket.

   The use of CONVERT TO DATE instructs the gateway to force a conversion if a source field may not match the type of the destination field.
3. Maps the ServiceNow field identifier (the string value short_description) to the ObjectServer table field called @Summary. Again the ON INSERT ONLY command instructs the gateway to set the 'short_description' field of the ServiceNow ticket only when the gateway creates the ticket.
4. Uses the Lookup function to reference the following tables defined in the servicenow.map map definition file:

- StateTable: The coalesce function means that if '@Severity' evaluates to null, set it to the value 1 (one). The coalesce function is needed because when the gateway receives an IDUC DELETE, all the fields are null. The coalesce function will assume the first non-null value from its arguments.
- ImpactTable: The gateway sends this field to ServiceNow only when the gateway receives an IDUC INSERT or an UPDATE from the ObjectServer. The gateway should not send this field to ServiceNow when it receives an IDUC DELETE from the ObjectServer.
- UrgencyTable: The gateway sends this field to ServiceNow only when the gateway receives an IDUC INSERT or an UPDATE from the ObjectServer. The gateway should not send this field to ServiceNow when it receives an IDUC DELETE from the ObjectServer.

5. A number of fields are specified as INTERNAL ONLY. This means that the gateway will see the values, but those values will not be sent across to ServiceNow. For a number of these fields, the gateway needs to see them because they are being used in the DEDUPLICATE clause (that is, 'severity', 'lastoccurrence', 'tally' and 'acknowledged'). For some other fields (sys_id, serverName, and serverSerial), the gateway needs them for its internal processing.
6. Ensures that for each IDUC DELETE received by the gateway, the gateway will send across the work_notes field with the value Alert deleted in the Object Server. Thus, it will be obvious in ServiceNow that the alert has been deleted.
7. Maps the ServiceNow fields close_code and close_notes to the ObjectServer table fields CloseCode and CloseNotes respectively.
8. The DEDUPLICATE clause specifies the fields to be included in the row hash used for deduplication. As specified in the DEDUPLICATE clause, these fields are severity, lastoccurrence, tally, and acknowledged.

**Note :** Do not remove or update fields identified with the with the # DO NOT UPDATE OR REMOVE comment.

## Explanation of the JournalMap

The JournalMap is one of the maps contained in the servicenow.map map definition file delivered with the gateway. As discussed previously, the JournalMap contains the mapping from ObjectServer alerts.journal table entries to ServiceNow work_notes fields. The JournalMap is defined as follows:

```
CREATE MAPPING JournalMap 1
(
    'work_notes' = replace(JOURNAL.TEXT,"\n","\\n"), 2
    'uid'               = JOURNAL.UID INTERNAL ONLY, 3
    'chrono'            = JOURNAL.CHRONO INTERNAL ONLY, 3
    'serverName'        = STATUS.SERVER_NAME INTERNAL ONLY, # DO
NOT UPDATE OR REMOVE 3
    'serverSerial'      = STATUS.SERVER_SERIAL INTERNAL ONLY # DO
NOT UPDATE OR REMOVE 3

);
```

The following list describes each of the numbered items in the JournalMap:

**Note :** Do not remove or update fields identified with the with the # DO NOT UPDATE OR REMOVE comment.

1. Uses the CREATE MAPPING command to create a map called JournalMap. As described previously, the JournalMap contains fields that are included for deduplication purposes.
2. Replaces all "\n" in the journal entry with "\\n" to avoid the backslash getting lost in the mapping.
3. Fields marked INTERNAL ONLY are not sent to ServiceNow, but included for deduplication. If not included, journal entries with the same text, but made at different times, will be deduplicated and the latter entries dropped.

   **Note :** Some field are listed INTERNAL ONLY (like serverName and serverSerial) because the gateway needs them for its internal processing.

**Note :** Without a `DEDUPLICATE` clause, all fields in a map are considered for deduplication purposes. The `DEDUPLICATE` clause specifies the fields to be included in the row hash used for deduplication in the specified map. Conversely, the `DO NOT DEDUPLICATE` clause means ignore the listed fields when computing the row hash used for deduplication.

### Specifying alternative values in the map definition file

When specifying alternative values for fields in the `servicenow.map` map definition file, ensure that the field does not exceed the maximum length of the target field in the ServiceNow ticket.

## Historic journal forwarding

You can configure how the gateway handles historic journal entries for an ObjectServer alert by setting the **`Gate.HistoricResync`** property.

By default, the gateway forwards historic journals (those created in the ObjectServer before the alert is sent for request creation) when the alert is created. Setting the **`Gate.HistoricResync`** property to `false` prevents the gateway from sending historic journals to ServiceNow. Note, however, that the gateway sends any subsequent journals to ServiceNow after the ticket is created.

## Store and forward capability

The Java Gateway for ServiceNow provides a store and forward capability to minimize any data loss should the gateway lose communication with ServiceNow. The store and forward capability is permanently enabled and goes from the ObjectServer to ServiceNow.

### Operation of store and forward

The gateway creates a batch file for the following operations (depending on what is configured in the `servicenow.rdrwtr.tblrep.def` file):

- Alert (`alerts.status` table) insert, update, or delete
- Alert journals (`alerts.journal` table) insert

When the gateway reestablishes communication with ServiceNow, it sends the data that it has stored in the batch files.

### Startup behavior

When the gateway starts, it sends an HTTP request to ServiceNow that either succeeds or fails. If the HTTP request succeeds, the gateway determines if there are any store-and-forward batch files from a previous run and, if present, processes these batch files.

If the HTTP request to ServiceNow fails for any reason, the gateway shuts down. The gateway does not attempt to gather data from the ObjectServer and store it in store-and-forward batch files.

## 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
```

# AES encryption

AES encryption can be used to encrypt any string within the gateway properties file. It is used by the gateway to prevent sensitive data from being available in readable format in the gateway properties file.

**Note :** AES encryption is supported on all supported versions of Tivoli Netcool/OMNIbus on all UNIX and Linux operating systems.

## nco_aes_crypt

You can encrypt strings in the gateway properties file using the `nco_aes_crypt` tool (supplied with Tivoli Netcool/OMNIbus). The syntax of encrypted data is as follows:

```
@datalength:encrypted_data@
```

Where *datalength* is the length of the data in bytes (expressed as a decimal) and the data itself is base64 encoded. The at sign (`@`) indicates the start and end of the encrypted data definition. The colon (`:`) acts as a field separator.

The encrypted values appear in single quotes in the properties file. The following example shows the server password in encrypted format for the **`Gate.ServiceNow.Password`** property in the `G_SERVICENOW.props` properties file:

```
# ServiceNow gateway specific properties
.
.
.
Gate.ServiceNow.Password  :
'@64:lHBLuIPLNye8zCWhykFVFY7y90V9kCjGK5GSWu5VBdSlgQOqarq6T4UK4xk5Vqix@'
.
.
.
```

**Note :** You can obtain the `nco_aes_crypt` tool from the IBM Passport Advantage website: http://www-306.ibm.com/software/howtobuy/passportadvantage/pao_customers.htm. Access the Software Downloads section and search for `Netcool/OMNIbus Gateway configuration encryption library`.

## Using the nco_aes_crypt tool

Property values in the properties file must be encrypted using the `nco_aes_crypt` tool.

This is a command line tool which takes the following format:

```
nco_aes_crypt [-d] [-o outfile] [-c cipher] -k keyfile -f filename
nco_aes_crypt [-d] [-o outfile] [-c cipher] -k keyfile data
```

The output of this command will be the encrypted string to be used in the properties file.

The following table describes the options available with `nco_aes_crypt`:

| *Table 5. nco_aes_crypt command line options* | |
|---|---|
| **Command line option** | **Description** |
| `-d` | Use this option to specify the mode in which the `nco_aes_crypt` tool runs:<br>d - decrypt mode<br>The default is encrypt mode. |
| `-o` *string* | Use this option to specify the output file to which the encrypted or decrypted data will be written. |
| `-c` *string* | Use this option to specify the cipher to use:<br>• AES - Specifies the non-FIPS cipher.<br>• AES_FIPS - Specifies the FIPS cipher.<br>The default is AES (non-FIPS). |
| `-k` *string* | Use this option to specify the path of the file containing the key data. This option is mandatory. |
| `-f` *string* | Use this option to specify the path of the file containing data requiring encryption or decryption. |
| *data* | Use this option to specify the data to be encrypted or decrypted. |

## Encryption key file

The encryption key is stored in a flat file alongside the encrypted data. The key storage file has an ASCII numeric key length indicator followed by a colon and the key in binary form.

The format of the key file is as follows:

```
key_length:key_data
```

Where `key_length` is the length of the key in bits and the `key_data` is the key in binary form. Valid length values are 128, 192 and 256.

For example:

```
128:1234567812345678
```

In this case, `key_length` is 128 since the ASCII string 1234567812345678 has 16 bytes (128 bits).

You can generate random or pre-defined keys of varying lengths using `nco_keygen`. To generate a key file, use the following command:

```
nco_keygen -o outfile[-l length|-k key] [-h] [-?]
```

The following table gives the descriptions of the above command line options.

| *Table 6. Encrytion key file command line options* | |
|---|---|
| **Command line option** | **Description** |
| `-o` *outfile* | Use this option to specify the output file name. |

| *Table 6. Encrytion key file command line options (continued)* | |
|---|---|
| **Command line option** | **Description** |
| -l *length* | Use this option to specify the length (in bits) of the key to write out.<br>The default is 128.<br>**Note :** The value that you specify must be divisible by 8. |
| -k *key* | Use this option to specify the key to be written out, expressed as hex digits.<br>**Note :** This option bypasses automatic key generation. |
| -h /-? | Use this option to print the help information and exit |

**Note :** AES encryption is used as the initial encryption method for sensitive data. However, this does not mean that the data can be considered to be secure purely due to AES encryption; the security of the data depends on the restriction of access to the key file used for AES encryption. Access to this file is controlled using UNIX or Windows file permissions.

## Using encrypted data

To use encrypted data, you set the **ConfigKeyFile** property in the G_SERVICENOW.props properties file to the path of the file that contains the encryption key. For example:

```
# Generic Omnibus Properties
#
ConfigKeyFile : 'key_file_path'
.
.
.
```

Where *key_file_path* is the path to the file containing the encryption key.

## Running the ObjectServer in a secure mode

When the gateway connects to the ObjectServer running in secure mode, it needs to authenticate with a username and password. This username and password can be encrypted using the nco_aes_crypt tool.

To enable the encryption, the location of the key file must be specified using the **ConfigKeyFile** property in the G_SERVICENOW.props properties file, as described previously. You also need to specify the encrypted username and password required for authentication using the **Gate.RdrWtr.Username** and **Gate.RdrWtr.Password** properties in the G_SERVICENOW.props properties file.

The following example shows the three fields that need to be specified in the G_SERVICENOW.props properties file when the ObjectServer runs in a secure mode:

```
# Generic Omnibus Properties
#
ConfigKeyFile  : '/HOME/74/solaris/omnibus/keyfile_name'
.
.
.
# Gateway Framework properties
.
.
.
Gate.RdrWtr.Password   : '@44:mdyEb8VTh+2wALnNlR7dnGnxRZ3BkMOQbR5IgxLlHuc=@'
.
.
```

```
.
Gate.RdrWtr.Username  : '@44:2yXgd6fp9q1Ey4sSAb2RibzA3+PpCZmhAZXo6nNdkvQ=@'
```

# ServiceNow encryption

Use this information to learn how the Java Gateway for ServiceNow interacts with the standard security encryption that is built into the ServiceNow API.

Encryption between the gateway and ServiceNow is provided by the HTTPS protocol as defined within the REST URLs.

The following are some configuration tasks that you need to perform on both the ServiceNow server and the gateway server to configure the standard security encryption that is built into the ServiceNow API.

## Configuring the standard security encryption on the ServiceNow server

To configure the standard security encryption on the ServiceNow server, follow the steps in the ServiceNow documentation.

# Configuring the PollingAdjustment property

The **PollingAdjustment** property (measured in seconds), is used for adjusting the ServiceNow update time scope (relative to the gateway's clock) used in polling.

## Range of PollingAdjustment

Since the gateway is run remotely from ServiceNow, the clocks at both sides are bound to be out of sync to some degree. This clock difference has an effect on gateway polling for ServiceNow updates.

You cannot rely on the default **PollingAdjustment** value. Gateway polling might become pointless if the gateway clock is leading without the aid of the **PollingAdjustment** setting to reverse the gap.

The range for the **PollingAdjustment** property is determined by which clock is leading.

**Case 1:**

GW_Clock <= ServiceNow_Clock (ServiceNow_Clock is leading)

**PollingAdjustment** = X, where 0 <= X < (ServiceNow_Clock – GW_Clock)

**Case 2:**

GW_Clock > ServiceNow_Clock (Gateway clock is leading)

**PollingAdjustment** = -X, where X >= (GW_Clock – ServiceNow_Clock)

To keep the gateway operationally healthy and efficient in performing polling for a substantial period, you should use a **PollingAdjustment** value that makes polled time scope slightly behind the ServiceNow's clock by between 10 seconds and 30 seconds.

## Calculation of polled time scope

The gateway polls ServiceNow for updates periodically. Every iteration's polled time scope is based on the gateway's current time offset (forward or backward) as defined by the **PollingAdjustment** value.

If applied consistently to the clock rules as described above, when the **PollingAdjustment** property has a positive value, the resultant polled time scope will always be ahead of the gateway’s clock.

In the gateway log, the polled time scope is expressed in UTC. The polled time scope has a lower_bound and an upper_bound). The following example is the first polled time scope. The **PollingAdjustment** was configured as 200 seconds, and the time local to the gateway (the log timestamp) is UTC+8.

Lower bound

```
16/09/01 12:07:08: Debug: [Notifications] Encoded value '2016-09-01 04:10:28':
'2016-09-01%2004%3A10%3A28'
```

Upper bound

```
16/09/01 12:07:08: Debug: [Notifications] Encoded value '2016-09-01 04:10:28':
'2016-09-01%2004%3A10%3A28'
```

For first polled time scope, the difference between upper_bound and lower_bound is zero seconds. For all subsequent polled time scope, the difference between upper_bound and lower_bound is the length of polling interval. The lower_bound of a polled time scope is the upper_cound of its predecessor. The upper bound value is the gateway's current time plus polling adjustment value.

Example:

**1st polling**

```
16/09/01 12:07:08: Debug: [Notifications]
Polling for updates in ServiceNow from Thu Sep 01 12:10:28 MYT 2016
16/09/01 12:07:08: Debug: [Notifications]
Encoded value '2016-09-01 04:10:28': '2016-09-01%2004%3A10%3A28'.
16/09/01 12:07:08: Debug: [Notifications]
Encoded value '2016-09-01 04:10:28': '2016-09-01%2004%3A10%3A28'
```

**2nd polling**

```
16/09/01 12:07:39: Debug: [Notifications]
Polling for updates in ServiceNow from Thu Sep 01 12:10:28 MYT 2016
16/09/01 12:07:39: Debug: [Notifications]
Encoded value '2016-09-01 04:10:28': '2016-09-01%2004%3A10%3A28'.
16/09/01 12:07:39: Debug: [Notifications]
Encoded value '2016-09-01 04:10:59': '2016-09-01%2004%3A10%3A59'.
```

### Problems in polling operations

Progressive clock drift or an inappropriate **`PollingAdjustment`** value may lead to these possibilities:

1. **Latent update**

   Caused by the resultant polled time scope being behind the ServiceNow's clock. The further the polled time scope is behind the ServiceNow's clock, the more iterations will the gateway need to take to reach subsequent time scopes that may be containing updates.

   As a result ObjectServer's reflection of updates in ServiceNow tickets would appear lagging. If there were series of changes in a ticket's fields over time, the history (save the latest update) would be oblivious to ObjectServer because of latent polling.

2. **Skipping update**

   Caused the resultant polled time scope being ahead of the ServiceNow's clock. This is the worst scenario for the gateway, because there can be no updates in the future time frame that has yet to come in ServiceNow.

## Identifying ServiceNow ticket reclaim candidates

In normal case, at the last stage of ticket creation the gateway will update TargetId to the alert with the value extracted from ServiceNow data (by the field name configured in **`Gateway.ServiceNow.TargetId`**) as to establish the linkage between a ticket and its origin alert for future reference.

If this step did not happen, then the alert is not associated to the ticket (supposedly created). From gateway perspective the alert has lost its ServiceNow ticket. To maintain alert coherence gateway will reclaim lost tickets.

When **Gateway.ServiceNow.ReclaimIncident** is set to 1 or 2, ticket reclaim will be performed at appropriate points in gateway operation life cycle. Having been marked with **CreationErrorCode** or not, any alert without **TargetId** in sight becomes a candidate for ticket reclaim.

The action of reclaiming ticket is performed at two areas of processing:

1. Outbound – ticket creation, update, and deletion.

   Reclaim is controlled by **Gateway.ServiceNow.ReclaimIncident**. If reclaim is enabled, the gateway will determine whether an alert is new or is for update by querying ServiceNow by alternate index field for a corresponding ticket.

   Upon confirmation of a ticket found in ServiceNow the gateway will update TargetId to the alert and then sent the alert data out as update. Otherwise the gateway will send the alert for ticket creation.

2. Inbound – polled update parsing

   Reclaim is controlled by **Gateway.ServiceNow.AlternateIndexPolicy**. When alternate index policy is 'internal', then ticket reclaim is enabled. When processing the returned data of a ticket for update in ObjectServer, the gateway will need to associate the ticket to its origin alert through TargetId.

   If no alert was found by this mean, then the gateway will attempt reclaim by search using the keys: ServerName and ServerSerial, decomposed from the combined key of internal alternate index field obtained from ticket data.

   To reclaim ticket the newly-associated alert will be updated with **TargetID**. Returned ticket data without its origin alert identified is skipped for update.

## ServiceNow incident resolution

This version of the Gateway for ServiceNow provides functionality for providing ServiceNow incident resolution data.

By default, ServiceNow instances enforce a rule that requires the Resolution Code field (table field name: `close_code`) and Resolution Notes field (table field name: `close_notes`) to be populated when the incident is set to `Resolved` or `Closed`. These fields correspond to the `CloseCode` and `CloseNotes` fields in the ObjectServer respectively.

The `Resolution Code` field must be set to one of the defined options in the ServiceNow instance. This field is a text string and can be customized by defining additional options. The `Resolution Notes` field must also be populated when the `Resolution Code` is provided, but can be set to any value.

The gateway maps events in the ObjectServer that have `Severity` set to `Clear` into ServiceNow as incidents with `State` set to `Resolved`. The gateway provides a method to specify values for these fields so that the event is inserted/updated into ServiceNow correctly with the `Resolution Code` and `Resolution Notes` fields populated appropriately.

### Updating the gateway to support ServiceNow Incident Resolution

If you are upgrading from a previous version of the Gateway for ServiceNow, use the following steps to enable the gateway to support ServiceNow Incident Resolution:

1. Update the `alerts.status` table to add the new columns: `CloseCode` and `CloseNotes`:

   ```
   ```
   alter table alerts.status add column CloseCode varchar(64);
   go

   alter table alerts.status add column CloseNotes varchar(64);
   go
   ```
   ```

2. Edit the `updatePayload()` function in the `servicenow.generic.js` JavaScript file to specify the default values for `CloseCode` and `CloseNotes`.

```
//
// This function is executed before an event is inserted/updated into
// ServiceNow and is intended to allow for the overriding of values in the event.
// The functionality defined below allows for the close_code and close_notes
// fields to be overridden with new values.
//
// @param inputs
//      A Map of values for the event being processed by the gateway.
//
function updatePayload(inputs, operation) {

    if (operation === "insert" || operation === "update") {
        var severity = inputs.get("severity");
        if (severity === 0) {
            var new_close_code = "Closed/Resolved by Caller";
            inputs.put("close_code", new_close_code);
            var new_close_notes = "Event Severity set to 0 at object server\.";
            inputs.put("close_notes", new_close_notes);

            logger.debug("generic.js returning close_code=" + new_close_code + " and close_notes=" + new_close_notes
                + " for " + operation + " to the gateway.");
        }
    }
    else if (operation === "delete") {
        // At this point, the event is already deleted at the object server
        // So we set the values for the fields required to be replicated to ServiceNow
        var new_close_code = "Closed/Resolved by Caller";
        inputs.put("close_code", new_close_code);
        var new_close_notes = "Event deleted at object server\.";
        inputs.put("close_notes", new_close_notes);
        var state_closed = 7;
        inputs.put("state", state_closed);

        logger.debug("generic.js returning close_code=" + new_close_code + " and close_notes=" + new_close_notes + " and
state="
            + state_closed + " for " + operation + " to the gateway.");
    }

    return inputs;
}```
```

The function above provides `Closed/Resolved by Caller` for the `close_code` field and `Provided by Netcool/OMNIbus ServiceNow Gateway` for events with `Severity=0 (Clear)`.

You should update the function to specify the correct values for the fields as needed. Be aware that as the script is written in JavaScript, JavaScript syntax applies here.

Once the changes have been performed, restart the gateway for the changes to take effect.

3. Ensure the `inboundUpdate` function in the `servicenow.notification.js` file includes the `CloseCode` and `CloseNotes` fields:

```
```
//
// Update an alert with data received from ServiceNow
//
// Params:
//   alert - The alert being updated
//   inputs - Input values, as a name/value map, for the update
function inboundUpdate(alert,inputs)
{
    try {
        // Dates can be parsed using dateformat.parse.
        // The format used is as used by the Java SimpleDateFormat class.
        // http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html
        //
        // Example:
        // var resolvetime = dateformat.parse("dd/MM/yyyy' 'HH:mm:ss", inputs.get("ResolveTime"));
        values = sog.newrow();
        values.put("ServiceNowState", inputs.get("state"));
        values.put("TTNumber", inputs.get("number"));
        values.put("CloseCode", inputs.get("close_code"));
        values.put("CloseNotes", inputs.get("close_notes"));
        alert.update(values);
        if( null == inputs.get("sys_id") ) {
            addJournal(alert, "Ticket updated in ServiceNow with TTNumber: " + inputs.get("number") + " and state: " +
inputs.get("state"));
        } else {
            addJournal(alert, "Ticket updated in ServiceNow with SysId: " + inputs.get("sys_id") + " and TTNumber: " +
inputs.get("number") + " and state: " + inputs.get("state"));
        }
    } catch(err) {
        logError(alert, "Unable to update alert in OMNIbus: " + err);
    }
}
```
```

## Removing support for the ServiceNow Incident Resolution

If you are working with a `ServiceNow` instance that does not enforce the rule requiring `Resolution Code for Resolved Incidents`, remove support for ServiceNow Incident Resolution using the following steps:

1. Edit the `updatePayload()` function in the `servicenow.generic.js` JavaScript file to remove the functionality to provide the `CloseCode` and `CloseNotes` data:

```
```
function updatePayload(inputs, operation) {
    return inputs;
}
```
```

2. Remove the `CloseCode` and `CloseNotes` fields from the `inboundUpdate` function in `servicenow.notification.js`:

```
```
//
// Update an alert with data received from ServiceNow
//
// Params:
//   alert - The alert being updated
//   inputs - Input values, as a name/value map, for the update
function inboundUpdate(alert,inputs)
{
    try {
        // Dates can be parsed using dateformat.parse.
        // The format used is as used by the Java SimpleDateFormat class.
        // http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html
        //
        // Example:
        // var resolvetime = dateformat.parse("dd/MM/yyyy' 'HH:mm:ss", inputs.get("ResolveTime"));
        values = sog.newrow();
        values.put("ServiceNowState", inputs.get("state"));
        values.put("TTNumber", inputs.get("number"));
        alert.update(values);
        if( null == inputs.get("sys_id") ) {
            addJournal(alert, "Ticket updated in ServiceNow with TTNumber: " + inputs.get("number") + " and state: " +
inputs.get("state"));
        } else {
            addJournal(alert, "Ticket updated in ServiceNow with SysId: " + inputs.get("sys_id") + " and TTNumber: " +
inputs.get("number") + " and state: " + inputs.get("state"));
        }
    } catch(err) {
        logError(alert, "Unable to update alert in OMNIbus: " + err);
    }
}
```
```

3. Optional: remove the `CloseCode` and `CloseNotes` fields from the ObjectServer:

```
```
alter table alerts.status drop column CloseCode;
go

alter table alerts.status drop column CloseNotes;
go
```
```

After making the changes, restart the gateway for the changes to take effect.

# Gateway operation

Use this information to learn how the Java Gateway for ServiceNow interacts with ServiceNow. This includes information on how the gateway interacts with ServiceNow with regard to ticket operations.

Table 7 on page 34 identifies some of the operations related to how the gateway interacts with ServiceNow, including those related to ticket operations. For each operation, the table lists the properties you use to control how the gateway performs the operation, and the section of the guide that provides details about the operation and the valid values for the associated properties. For each operation, set the properties to the correct values or verify that their default values are suitable for your environment.

**Note :** The table references the following categories of properties that are defined in the `G_SERVICENOW.props` properties file:

Gateway Framework properties
ServiceNow gateway specific properties

For reference information and command line options on these properties, see “Properties and command line options” on page 43.

*Table 7. Configuring gateway operation*

| Configure gateway operation tasks | Properties | See |
|---|---|---|
| Controlling synchronization between the gateway and ServiceNow | None | “Controlling synchronization between the gateway and ServiceNow” on page 35 |
| Controlling which ServiceNow target table the gateway uses | **Gate.ServiceNow.TableName** | “Controlling which ServiceNow target table the gateway uses” on page 35 |
| Controlling how the gateway queries the ServiceNow instance database | **Gate.ServiceNow.QueryPolicy** | “Understanding how the gateway queries the ServiceNow instance database” on page 36 |
| Controlling when and how many times the gateway should retry an operation that fails | **Gate.ServiceNow.RetryLimit**<br>**Gate.ServiceNow.RetryWait** | “Controlling when and how many times the gateway should retry an operation that fails” on page 35 |
| Controlling how the gateway handles the alternate index policy | **Gate.ServiceNow.AlternateIndexField**<br>**Gate.ServiceNow.AlternateIndexPolicy** | “Controlling how the gateway handles the alternate index policy” on page 36 |
| Controlling how the gateway handles outbound data management | **Gate.ServiceNow.DateAndTimeFormat**<br>**Gate.ServiceNow.OutboundManagedTypes**<br>**Gate.ServiceNow.OutboundManagement** | “Controlling how the gateway handles outbound data management” on page 37 |
| Controlling how the gateway handles notifications | **Gate.ServiceNow.DropFirstNotification**<br>**Gate.ServiceNow.ModifCounterField**<br>**Gate.ServiceNow.PollingInterval** | “Controlling how the gateway handles first notifications of created tickets” on page 38 |
| Controlling where the gateway stores error codes | **Gate.ServiceNow.ErrorCodeColumn** | “Controlling where the gateway stores error codes” on page 38 |
| Controlling how the gateway performs checks before it creates a ticket | **Gate.ServiceNow.PreCreateChecks** | “Controlling how the gateway performs checks before it creates a ticket” on page 39 |
| Controlling how the gateway handles ServiceNow tickets | **Gate.ServiceNow.PollingInterval** | “Controlling how the gateway handles processing of ServiceNow tickets” on page 40 |

| *Table 7. Configuring gateway operation (continued)* | | |
|---|---|---|
| **Configure gateway operation tasks** | **Properties** | **See** |
| Controlling how the gateway handles operations related to ticket fields | **Gate.ServiceNow.SysIdField**<br>**Gate.ServiceNow.TicketIdField**<br>**Gate.ServiceNow.UpdateFields**<br>**Gate.ServiceNow.UpdateQueryStartTime** | "Controlling how the gateway handles operations related to ServiceNow ticket fields" on page 40 |

The topics that follow make use of the following terms:

- Outbound -- Specifies the data flow sent from the gateway to the ServiceNow instance database.
- Inbound -- Specifies the data flow received by the gateway from the ServiceNow instance database.

## Controlling synchronization between the gateway and ServiceNow

You need to ensure that the server on which the gateway is running and the server on which ServiceNow is running are on the same system time. Specifically, the clocks on the gateway server and the ServiceNow server need to be synchronized.

Also, to prevent miscommunication of ticket details, the time zones of the ObjectServer, the gateway server, and the ServiceNow server must be synchronized at all times.

See your operating system documentation for information on how to synchronize the server clocks.

## Controlling which ServiceNow target table the gateway uses

The gateway communicates with a ServiceNow instance database to create tickets and to retrieve data about these tickets. The ServiceNow instance database consists of tables each of which is a collection of records. Specifically, the gateway needs to know which table in the ServiceNow instance database it should use to create tickets and to retrieve data about these tickets. The gateway uses the **Gate.ServiceNow.TableName** property to identify the table (referred to as the target table) in the ServiceNow instance database it should use to create tickets and to retrieve data about these tickets. The default name for the target table in the **Gate.ServiceNow.TableName** property is `incident`.

Edit this property if you want the gateway to access a different target table in the ServiceNow instance database.

**Note :** In addition to editing the **Gate.ServiceNow.TableName** property to target an alternative table, you will also need to configure accordingly the `servicenow.rdrwtr.tblrep.def` table definition file, the `servicenow.map` map definition file, the `notification.js` file, and other properties.

### Controlling when and how many times the gateway should retry an operation that fails

Following successful completion of the initial REST GET requests during the gateway startup sequence, you can control how the gateway handles failed operations by using the **Gate.ServiceNow.RetryLimit** and **Gate.ServiceNow.RetryWait** properties as follows:

- Use the **Gate.ServiceNow.RetryLimit** property to specify the maximum number of retries the gateway should make on an operation that failed. The default value of `0` (zero) means there is no limit to the number of retries that the gateway makes on a failed operation.

- Use the **Gate.ServiceNow.RetryWait** property to specify the number of seconds the gateway should wait before retrying an operation that failed. The default value is 7 seconds.

The following example shows that the **Gate.ServiceNow.RetryLimit** property is set to its default value of 0 and the **Gate.ServiceNow.RetryWait** property is set to the value of 5 seconds:

```
.
.
.
Gate.ServiceNow.RetryWait    : 5
Gate.ServiceNow.RetryLimit   : 0
.
.
.
```

Given the above settings, if the gateway operation to forward an event to the ServiceNow instance fails, the gateway waits five seconds and then retries the event forwarding operation. The gateway waits five seconds each time it performs the event forwarding operation until it succeeds.

### Understanding how the gateway queries the ServiceNow instance database

The gateway communicates with the ServiceNow instance database by adhering to the policy specified in the **Gate.ServiceNow.QueryPolicy** property. Specifically, the gateway performs all queries on the ServiceNow instance database using a scoping clause that includes the following:

```
sys_create_by=sngw_user
```

Where *sngw_user* is the value specified in the **Gate.ServiceNow.Username** property.

Thus, the only tickets in the ServiceNow instance database that are of interest to the gateway are those tickets that the gateway itself creates.

The default query policy value is `Account`.

**Note :** Currently, this property supports only a query policy with the value `Account`.

## Controlling how the gateway handles the alternate index policy

It is very important that you configure the gateway with an alternate index field (in the **Gate.ServiceNow.AlternateIndexField** property) whenever possible. An alternate index field allows the gateway to enforce end-to-end correlation integrity. The gateway, by default, is configured to use an alternate index field of `correlation_id`, which is a column in the ServiceNow incident table.

As discussed in "Controlling which ServiceNow target table the gateway uses" on page 35 the gateway uses the ServiceNow target table specified in the **Gate.ServiceNow.TableName** property to create tickets and to retrieve data about these tickets. You control how the gateway handles the alternate index policy behavior through the **Gate.ServiceNow.AlternateIndexField** and **Gate.ServiceNow.AlternateIndexPolicy** properties. Specifically, you use the **Gate.ServiceNow.AlternateIndexField** to specify the name of the column in the ServiceNow target table that the gateway uses as an alternate index. You use the **Gate.ServiceNow.AlternateIndexPolicy** property to instruct the gateway on how to set the alternate index field.

The default column name in the ServiceNow target table is `correlation_id`. This column name implies that the values in the ServiceNow instance database are unique within the scope of records that the gateway queries. More specifically, the `correlation_id` must be unique across all tickets with the same value for `sys_created_by` equal to that of the user name of the dedicated ServiceNow gateway account. Note that the **Gate.ServiceNow.QueryPolicy** property (discussed in "Understanding how the gateway queries the ServiceNow instance database" on page 36) controls how the gateway queries the ServiceNow instance database by using a scoping clause.

The default behavior for the way the gateway sets the alternate index field value is specified by the value `internal`.

Specifically, the operational behavior for how the gateway sets the alternate index field value in the **Gate.ServiceNow.AlternateIndexField** property is specified by the following values:

- `internal`: Instructs the gateway to internally set the alternate index for the ServiceNow target table to the name of the originating ObjectServer and the serial number of the alert on the originating ObjectServer. Specifying `internal` causes the gateway to generate a value that can be used for the column specified in the **Gate.ServiceNow.AlternateIndexField** property. An example of an internally set alternate index value might be `Correlation ID: NCOMS/2421`

  **Note :** If the **Gate.ServiceNow.AlternateIndexField** property is an empty string and **Gate.ServiceNow.AlternateIndexPolicy** is set to `internal`, the value that is generated cannot be included in the outbound data because there is no column specified to hold the value.

- `custom`:

  Indicates that the value of the **Gate.ServiceNow.AlternateIndexPolicy** property should be set in the `StatusMap` (defined in the `servicenow.map` map definition file). You must ensure a unique value for each ticket in the **Gate.ServiceNow.AlternateIndexField**.

  Specifying `custom` means that the user takes responsibility for the value used within the **Gate.ServiceNow.AlternateIndexField** property by specifying the alternate index field in the `servicenow.map` definition file as described above.

  **Note :** If the **Gate.ServiceNow.AlternateIndexField** property is an empty string and **Gate.ServiceNow.AlternateIndexPolicy** is set to `custom`, the gateway reports an error and shuts down.

The combination that must be used to effectively disable alternate index functionality is to specify `internal` for the **Gate.ServiceNow.AlternateIndexPolicy** property and `""` for the **Gate.ServiceNow.AlternateIndexField** property.

## Controlling how the gateway handles outbound data management

Outbound data management refers to the data flow sent from the gateway to the ServiceNow instance database. Conversely, inbound data management refers to the data flow received by the gateway from the ServiceNow instance database. You control how the gateway handles outbound data management through the **Gate.ServiceNow.OutboundManagedTypes** and **Gate.ServiceNow.OutboundManagement** properties. Specifically, you use the **Gate.ServiceNow.OutboundManagedTypes** property to specify which types of data the gateway should use as part of its outbound data management operations. For example, `type1;type2;type3`. Note that the semicolon (`;`) is used to separate the data types. The default value for the types of data is `""`. Note that the **Gate.ServiceNow.OutboundManagedTypes** property is active only if you specify the value `custom` for the **Gate.ServiceNow.Outbound Management** property.

You use the **Gate.ServiceNow.OutboundManagement** property to instruct the gateway on how to perform outbound data management operations. The default operational behavior for outbound data management operations is specified by the value `default`.

Specifically, you control how the gateway performs outbound data management by specifying one of the following values:

- `default`: The gateway performs the default outbound data management behavior. Currently, this behavior is limited to field width checking for specified types.
- `disable`: Switches off all outbound data management.
- `custom`: The gateway performs the custom outbound data management behavior. Currently, this is limited to field width checking for the types specified by the **Gate.ServiceNow.OutboundManagedTypes** property.

  The following are the built in types for which the gateway performs field width checking operations:

  – `integer`

  – `string`

- journal
- journal_input
- journal_list

If a field width is breached for a managed outbound type, the gateway logs a warning message and the data will be truncated in the ServiceNow instance database. There is one exception to this rule that occurs if you enable pre-create checks (using the **Gate.ServiceNow.PreCreateChecks** property) and outbound management (using the **Gate.ServiceNow.OutboundManagement** property) and the outbound field is the alternate index field (specified by the **Gate.ServiceNow.AlternateIndexField** property) whose type is managed. In this case, if the field width is breached then the gateway aborts the create ticket operation because truncation of data on the ServiceNow instance database could cause duplicate alternate index field values.

Another property related to how the gateway handles outbound data is the **Gate.ServiceNow.DateAndTimeFormat** property. This property specifies the format for the outbound date and time data so that it is compatible with the format specified for the associated ServiceNow system properties. The default value is `yyyy-MM-dd HH:mm:ss`.

**Note :** If this date and time format is not compatible with how ServiceNow is configured, the outbound date and time data will be ignored by ServiceNow.

## Controlling how the gateway handles first notifications of created tickets

The gateway periodically sends HTTP polling requests to the ServiceNow instance. As a result of these HTTP polling requests, the gateway retrieves recent changes, for example, that the ServiceNow instance has created a new ticket. You can control whether the gateway should process the first notification of a created ticket through the **Gate.ServiceNow.DropFirstNotification** property.

The property takes one of the following values:

- `true`: The gateway should not process the first notification that it receives following the creation of a ticket by the ServiceNow instance.
- `false`: The gateway should process the first notification that it receives following the creation of a ticket by the ServiceNow instance.

The default value is `true`.

The gateway identifies the first update to a ticket by checking the value of the **Gate.ServiceNow.ModifCounterField** property (default value of `sys_mod_count`). If `sys_mod_count` is equal to 0 (zero), then the gateway knows that the ticket has not yet been updated on ServiceNow. Thus, the gateway determines that the ticket is newly created. If `sys_mod_count` is equal to a value greater than 0 (zero) in the values received from the HTTP polling request, the gateway evaluates the notification as not resulting from a ticket creation operation.

## Controlling where the gateway stores error codes

When the gateway encounters an error during some operation, it writes a gateway specific error code to a column in an ObjectServer alerts.status table. You can specify the name of the column in the ObjectServer alerts.status table where the gateway stores these error codes using the **Gate.ServiceNow.ErrorCodeColumn** property.

The default name of the column is `ServiceNowErrorCode`.

**Note :** The column in the ObjectServer alerts.status table is created when you run the default `servicenow.sql` that is provided with the gateway.

Table 8 on page 39 identifies some of the possible gateway specific error codes.

| *Table 8. Gateway specific error codes* | | |
|---|---|---|
| **Gateway error code** | **Description** | **Action** |
| 1 | The gateway reported a transient error when performing the specified operation. The following list identifies some of these operations:<br>• The gateway was unable to perform a check on the pre-create operation.<br>• A CREATE operation fails and an HTTP error gets generated (that is, there is a communication problem). In this case, the CREATE operation should be attempted again when the communication is restored. | No action required. The next time the gateway resynchronizes with the ObjectServer the gateway attempts the action again. |
| 2 | The gateway reported an unrecoverable error when performing the specified operation. The following list identifies some of these operations:<br>• The gateway was unable to update the specified ticket because it could not locate the ticket in the ServiceNow instance database. | For this particular operation, if you want to discard further updates on the original alert, update the `FILTER BY` clause in the `servicenow.rdrwtr .tblrep.def` table replication definition file to filter out events on alerts when the `ServiceNowErrorCode` is set to the value 2. |

## Controlling how the gateway performs checks before it creates a ticket

The gateway can perform a variety of checks before it creates a record in the ServiceNow target table.

**Note :** You specify the name of the ServiceNow target table using the **`Gate.ServiceNow.TableName`** property, as described in "Controlling which ServiceNow target table the gateway uses" on page 35.

You control the pre-create record checking behavior by specifying one of the following values for the **`Gate.ServiceNow.PreCreateChecks`** property:

- `default`: The gateway performs, where possible, default pre-create record check operations. Currently, the default behavior centers around checking the integrity of the **`Gate.ServiceNow.AlternateIndexField`** property and any associated value presented as part of a create record request.

  **Note :** If the **`Gate.ServiceNow.AlternateIndexField`** property is an empty string, then it effectively disables any pre-create record check operations associated with that property. The gateway supplies a warning message associated with this condition during gateway start up.
- `disable`: Disables any pre-create check. In other words, the gateway is prevented from performing any pre-create check operations.

If the **`Gate.ServiceNow.AlternateIndexField`** is active, the gateway performs the following pre-create record check operations as part of the default pre-create record check functionality:

1. Checks the ServiceNow target table for any records that have the same alternate index value. If duplicate alternate index values are found, the gateway reports an error condition and aborts the create record operation.
2. Checks if the alternate index value is empty. This is assumed to be an error condition and the gateway reports an appropriate error and aborts the create record operation.
3. Checks if the **Gate.ServiceNow.OutboundManagement** property is set to default and that the **Gate.ServiceNow.AlternateIndexField** property has a type that is being managed, then the gateway checks the alternate index value to determine that the data will fit (that is, not too wide). If it is too wide, the gateway reports an error condition and aborts the create record operation.

**Note :** The default value for the **Gate.ServiceNow.AlternateIndexField** property is `correlation_id`, which is of type string and is one of the default managed types. Therefore, by default, the gateway checks the width of the alternate index values and aborts the create record operation if they are too wide.

See "Controlling how the gateway handles the alternate index policy" on page 36 for information about the **Gate.ServiceNow.AlternateIndexField** property and "Controlling how the gateway handles outbound data management" on page 37 for information about the **Gate.ServiceNow.OutboundManagement** property.

## Controlling how the gateway handles processing of ServiceNow tickets

You can control the following characteristics of how the gateway handles the processing of ServiceNow records by using the specified gateway specific property:

- The gateway periodically queries the ServiceNow instance to check for updates to records. You use the **Gate.ServiceNow.PollingInterval** property to specify how often (in seconds) the gateway should query the ServiceNow instance to check for updates to records. The default query interval is 17 seconds.

## Controlling how the gateway handles operations related to ServiceNow ticket fields

You can control the following characteristics of how the gateway handles the processing of ServiceNow record fields by using the specified gateway specific property:

- The gateway core framework needs to know which column in the ServiceNow target table serves as the internal ticket identifier. You use the **Gate.ServiceNow.TicketIdField** property to specify the column in the ServiceNow target table that serves as the internal ticket identifier. This column name must be a unique record identifier. The default column name is `sys_id`.

  **Note :** You specify the name of the ServiceNow target table using the **Gate.ServiceNow.TableName** property, as described in "Controlling which ServiceNow target table the gateway uses" on page 35.
- When polling for updates, the gateway retrieves record field information from the ServiceNow target table by accessing specific columns from that table. You use the **Gate.ServiceNow.UpdateFields** property to specify which columns to use to retrieve record field information. The default columns are `state;number`. Note that you separate column names using the semicolon (`;`).

  In addition to the columns specified in this property there are six other default columns that the gateway retrieves when polling for updates. The following list identifies these six other default columns:

  ```
  sys_created_by
  sys_created_on
  sys_updated_on
  sys_id
  sys_updated_by
  sys_mod_count
  ```

  **Note :** The queried fields will be a combination of the six previously listed default columns plus update fields and, if configured, the alternate index field. The queried fields are merely returned to the gateway

as part of the polling REST GET request. In order to persist fields back to the ObjectServer, you will need to configure the functions defined in the `servicenow.notification.js` file accordingly.

- When the gateway begins to check for updates on the ServiceNow instance can be controlled using the **`Gate.ServiceNow.UpdateQueryStartTime`** property. You specify the initial date and time (in minutes) that the gateway uses to check for these updates. If the default value of `0` (zero) is used, the gateway will poll for changes that have occurred since the time the gateway was started.

  To specify a value other than 0 (zero), use the command line option for this property. For example:

  ```
  -servicenowupdatequerystarttime 15
  ```

  In the previous example, if the gateway, following its startup sequence, is ready to perform its first poll at 13:00 GMT, the query would look for updates after 12:45 GMT.
- The gateway uses the globally unique identifier (GUID) for each ServiceNow record to form the URLs that it uses to communicate with the ServiceNow instance. You use the **`Gate.ServiceNow.SysIdField`** property to specify the name for the ServiceNow target table column that holds the globally unique identifier (GUID) for each ServiceNow record. The default value for the GUID column name is `sys_id`.

  This specifies the field in the response body from a create ticket request (HTTP POST) that will be used in future request URLs to update the respective ticket.

## Controlling how the gateway handles resynchronization

The gateway can perform unidirectional resynchronization. To enable unidirectional resynchronization, the **`Gate.Cache.ResyncMode`** property is set to the default value of `UNI` in the `G_SERVICENOW.props` properties file. The gateway performs a resynchronize operation from Tivoli Netcool/OMNIbus to ServiceNow.

# Running the gateway

This topic describes how to run the gateway on UNIX and Windows operating systems. On Windows operating systems, you can run Process Control as a service and configure it to run the gateway.

Before running the gateway on UNIX and Windows operating systems, do the following:

1. Load the SQL (Structured Query Language) commands contained in the `servicenow.sql` file as follows:

   On UNIX:

   ```
   $OMNIHOME/bin/nco_sql -server server_name -user user_name
                         -password 'password' < $OMNIHOME/gates/servicenow/servicenow.sql
   ```

   On Windows:

   ```
   %OMNIHOME%\bin\isql.bat -U user_name -P 'password' -S server_name -i servicenow.sql
   ```

   **Note :** The `servicenow.sql` file adds four new columns to the ObjectServer alerts.status table and inserts values into alerts.conversions to map the `ServiceNowState` from an integer to a String. This is so that the state is displayed as a String in the EventList.

   Where:

   *server_name*: Specifies the name of the ObjectServer instance that the gateway is running against.
   *user_name*: Specifies the username for the user account to run the `nco_sql` command on the ObjectServer instance that the gateway is running against.
   '*password*': Specifies the password for the user account to run the `nco_sql` command on the ObjectServer instance that the gateway is running against. Single or double quotes for the password are acceptable.

<: Specifies the UNIX redirection operator that executes the `nco_sql` command using `$OMNIHOME/gates/servicenow/servicenow.sql` as the source of input.

The previous configuration step results in the following actions:

- Adding these four columns to the ObjectServer alerts.status table: `ServiceNowErrorCode`, `ServiceNowState`, `ServiceNowSysId`, `TTNumber`, `CloseCode`, and `CloseNotes`.
- Creating and populating a conversions table to convert `ServiceNowState` integer values to their string equivalents.

2. Edit the `G_SERVICENOW.props` properties file and then copy it under the `$OMNIHOME/etc` directory. The following properties must be updated:
   - **`Gate.ServiceNow.Host`** - Specify the IP address or hostname of the host server on which the ServiceNow instance is running.
   - **`Gate.ServiceNow.Username`** - Specify the username for the user account that the gateway uses to access the ServiceNow instance.
   - **`Gate.ServiceNow.Password`** - Specify the password for the user account that the gateway uses to access the ServiceNow instance.
   - **`Gate.RdrWtr.Server`** - Specify the name of the ObjectServer the gateway is running against.

   See "Properties and command line options" on page 43 for more information on these properties.
3. If you are upgrading from a previous release of the ServiceNow gateway, remove the following JAR files from the `$OMNIHOME/gates/java/` directory:

   ```
   ```
   commons-httpclient-3.1.jar
   commons-logging-1.1.1.jar
   commons-codec-1.6.jar
   ```
   ```

   Newer versions of these Apache libraries are now provided by the Java Gateway Framework (`nco-g-java`) that is bundled with the gateway package.

To run the gateway on UNIX operating systems, use the following command:

```
$OMNIHOME/bin/nco_g_servicenow
```

The command can take command line options; for example: `-propsfile <path to gateway properties file>`

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

**Note :** The following example shows how to run the gateway on a Windows 32 bit operating system with a 32 bit Java Runtime Environment (JRE). Occasionally, you might see an error message similar to this: `This app can't run on your PC. To find a version for your PC, check with the software publisher.` The solution to this issue is to run the gateway on a Windows 32 bit operating system with a 64 bit JRE (which means you would be executing a 64 bit `jave.exe`). You may have a 64 bit JRE already installed. Otherwise, you will have to install a 64 bit JRE.

When running the gateway on a Windows operating system, change directory to `%OMNIHOME%\gates\win32\` and use the following command:

```
nco_g_servicenow.bat
```

The command may take command line options; for example. `-propsfile <path to gateway properties file>`

**Note :** With no command line arguments, the gateway expects to find its properties file, `G_SERVICENOW.props`, located in the `$OMNIHOME/etc` directory.

**Note :** For additional information on running gateways and for running a second instance of a gateway on the same host, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

# Properties and command line options

The following tables describe the properties and command line options that are applicable to the Java Gateway for ServiceNow. They cover the properties specific to the gateway and those provided by the gateway's operational environment. Specifically, the tables provide these categories of properties: properties specific to the Java Gateway for ServiceNow, gateway framework properties, and generic Tivoli Netcool/OMNIbus properties.

Table 9 on page 43 lists and describes the properties specific to the Java Gateway for ServiceNow that are defined in the `G_SERVICENOW.props` properties file. The default values in Table 9 on page 43 are the default values as specified in the `G_SERVICENOW.props` properties file.

See "Gateway operation" on page 33 for more details on the properties specific to the Java Gateway for ServiceNow properties. The details include how the properties specific to the Java Gateway for ServiceNow relate to each other, depending on the values specified.

The descriptions that follow make use of the following terms:

- Outbound -- Specifies the data flow sent from the gateway to the ServiceNow instance database.
- Inbound -- Specifies the data flow received by the gateway from the ServiceNow instance database.

*Table 9. Gateway specific properties*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.AlternateIndexField** *string* | `-servicenowalternateindexfield` *string* | Use this property to specify the name of a column in the ServiceNow target table that can be used as an alternate index. This column name implies that the values in the ServiceNow instance database are unique within the scope of records that the gateway queries. Note that the **Gate.ServiceNow.QueryPolicy** property controls how the gateway queries the ServiceNow instance database by using a scoping clause.<br><br>The name of the ServiceNow target table is specified in the **Gate.ServiceNow.TableName** property.<br><br>The default column name in the ServiceNow target table is `correlation_id`. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.AlternateIndexPolicy** *string* | `-servicenowalternateindexpolicy` *string* | Use this property to specify how the gateway sets the alternate index field.<br><br>The property takes one of the following values:<br><br>• `internal`: Instructs the gateway to internally set the alternate index for the ServiceNow target table to the name of the originating ObjectServer and the serial number of the alert on the originating ObjectServer. Specifying `internal` causes the gateway to generate a value that can be used for the column specified in the **Gate.ServiceNow.AlternateIndexField** property.<br>• `custom`:<br><br>Indicates that the value of the **Gate.ServiceNow.AlternateIndexPolicy** property should be set in the `StatusMap` (defined in the `servicenow.map` map definition file). You must ensure a unique value for each ticket in the **Gate.ServiceNow.AlternateIndexField**.<br><br>Specifying `custom` means that the user takes responsibility for ensuring that the value used within the **Gate.ServiceNow.AlternateIndexField** property is unique for each ticket, by specifying the alternate index field in the `servicenow.map` definition file as described above.<br><br>The default operational behavior for the way the gateway sets the alternate index field value is specified by the value `internal`. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.Connections** *integer* | `-servicenowconnections` *integer* | Use this property to set the number of simultaneous connections the gateway makes with ServiceNow.<br>The default value is 3. |
| **Gateway.ServiceNow.CreationErrorCode** *string* | `-servicenowcreationerrorcode` *string* | Use this property to mark any alert subjected to ticket creation that receives a positive reply from the ServiceNow gateway but does not complete the final state of receiving or updating `targetID` to the alert.<br>The default is `'CREATION-FAILURE'`. |
| **Gate.ServiceNow.DateAndTimeFormat** *string* | `-servicenowdateformat` *string* | Use this property to specify the format for the outbound date and time data so that it is compatible with the format specified for the associated ServiceNow system properties.<br>The default value is `yyyy-MM-dd HH:mm:ss`.<br>See “Controlling how the gateway handles outbound data management” on page 37 for additional information about this property and its relationship to ServiceNow system properties. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.Drop FirstNotification** *boolean* | -servicenowdrop firstnotification *boolean* | Use this property to specify whether the gateway should process the first notification it receives following the creation of a ticket by the ServiceNow instance.<br>The property takes one of the following values:<br>• `true`: The gateway should not process the first notification that it receives following the creation of a ticket by the ServiceNow instance.<br>• `false`: The gateway should process the first notification that it receives following the creation of a ticket by the ServiceNow instance.<br>The default value is `true`. |
| **Gate.ServiceNow.Error CodeColumn** *string* | -servicenowerrorcodecolumn *string* | Use this property to specify the name of the column in an ObjectServer table to be used for storing gateway error codes.<br>The default name of the column is `ServiceNowErrorCode`. |
| **Gate.ServiceNow.Host** *string* | -servicenowhost *string* | Use this property to specify the name (or IP address) of the host server on which the ServiceNow instance is running. You must configure this property in order for the gateway to operate. The gateway uses the host server name (or IP address) specified in this property to form the URLs used to communicate with the ServiceNow instance.<br>The default value is `""`. |
| **Gate.ServiceNow.Modif CounterField** *string* | -servicenowmodifcounterfiel d *string* | Use this property to specify the name of the field in the ServiceNow ticket that holds the counter of modifications associated with the ticket.<br>The default field name is `sys_mod_count`. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.Outbound ManagedTypes** *string* | `-servicenowoutbound managedtypes` *string* | Use this property to specify which types of data the gateway should use as part of its outbound data management operations. For example, `type1;type2;type3`.<br>**Note :** This property is active only if you specify the value `custom` for the **Gate.ServiceNow.Outbound Management** property.<br>The default value is `""`. |
| **Gate.ServiceNow.Outbound Management** *string* | `-servicenowoutbound management` *string* | Use this property to specify how the gateway manages outbound data.<br>The property takes one of the following values:<br>• `default`: The gateway performs the default outbound data management behavior. Currently, this behavior is limited to field width checking for specified types.<br>When this property is set to `default`, the following are the built in types for which field width checking is performed:<br>– `integer`<br>– `string`<br>– `journal`<br>– `journal_input`<br>– `journal_list`<br>• `disable`: Switches off all outbound data management.<br>• `custom`: The gateway performs the custom outbound data management behavior. Currently, this is limited to field width checking for the types specified by the **Gate.ServiceNow. OutboundManagedTypes** property.<br>The default value is `default`. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.NoRetryCodes** *string* | `-servicenownoretrycodes` *string* | Use this property to configure the gateway to not retry REST queries against ServiceNow after receiving specific HTTP response codes returned from queries. The response codes indicate that specific errors have occurred and by not retrying the gateway may save time and resource to process remaining records. These errors should be addressed and resolve accordingly.<br>The default value is `403,404,429`.<br>To completely disable this functionality, uncomment the property and specify `''` (two single quotes). |
| **Gate.ServiceNow.Password** *string* | `-servicenowpassword` *string* | Use this property to specify the password for the service account that the gateway uses to access the ServiceNow instance. This service account must be created in ServiceNow and consist of a username and password. Use this property in conjunction with the **Gate.ServiceNow.Username** property.<br>The default value is `""`.<br>See “Authenticating the gateway with ServiceNow” on page 7 for additional information about this property and its relationship to defining suitable ServiceNow user roles. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.PollingAdjustment** *integer* | `-pollingadjustment` *integer* | Use this property to specify the adjustment (in seconds) that the gateway makes to allow for a difference in the time kept by the gateway clock and the ServiceNow clock.<br><br>When **Gate.ServiceNow.PollingAdjustment** is less then zero, and **Gate.ServiceNow.UpdateQueryStartTime** is greater than zero, the magnitude (the absolute value) of **Gate.ServiceNow.PollingAdjustment** must be less than or equal to **Gate.ServiceNow.UpdateQueryStartTime** in order to prevent miscalculation that places upper bound of query window behind lower bound.<br><br>If the condition is not complied, then gateway will throw exception and exit.<br><br>**Note :** For details about configuring this property, see "Configuring the PollingAdjustment property" on page 29. |
| **Gate.ServiceNow.PollingInterval** *integer* | `-servicenowpollinginterval` *integer* | Use this property to specify how often (in seconds) the gateway should query the ServiceNow instance to check for updates to tickets.<br><br>The default query interval is 17 seconds. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.PreCreateChecks** *string* | `-servicenowprecreatechecks` *string* | Use this property to specify how the gateway should perform checks before creating a new ticket in the ServiceNow target table.<br><br>The property takes one of the following values:<br><br>• `default`: The gateway performs, where possible, default pre-create record check operations. Currently, the default behavior centers around checking the integrity of the **Gate.ServiceNow.AlternateIndexField** property and any associated value presented as part of a create record request.<br><br>**Note :** If the **Gate.ServiceNow.AlternateIndexField** property is an empty string, then it effectively disables any pre-create record check operations associated with that property. The gateway supplies a warning message associated with this condition during gateway start up.<br><br>• `disable`: Disables any pre-create check. In other words, the gateway is prevented from performing any pre-create check operations.<br><br>The default checking behavior value is `default`. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.Proxy** *string* | `-servicenowproxy` *string* | Use this property to connect the gateway to the ServiceNow instance by an HTTP proxy server.<br>There are four ways to configure the property:<br>**Gate.ServiceNow.Proxy** : 'hostname' # Hostname only, port will be 80<br>**Gate.ServiceNow.Proxy** : 'hostname:port' # Hostname and port specification<br>**Gate.ServiceNow.Proxy** : 'user:password@hostname' # Hostname and authentication specification, port will be 80.<br>**Gate.ServiceNow.Proxy** : 'user:password@hostname:port' # Hostname, authentication and port specification.<br>Where `hostname` refers to the proxy hostname, `user` is the proxy username, and `password` is the proxy password. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.QueryPolicy** *string* | `-servicenowquerypolicy` *string* | Use this property to control how the gateway must query the ServiceNow instance database. Specifically, the gateway performs all queries on the ServiceNow instance database using a scoping clause that includes the following:<br>`sys_create_by=sngw_user`<br>Where<br>• `sys_create_by` is a ServiceNow instance database field.<br>• *sngw_user* is the value specified in the **Gate.ServiceNow.Username** property.<br>Thus, the only tickets in the ServiceNow instance database that are of interest to the gateway are those tickets that the gateway itself creates.<br>The default query policy value is `Account`.<br>**Note :** Currently, this property supports only a query policy with the value `Account`. |
| **Gateway.ServiceNow.ReclaimIncident** *integer* | `-servicenowreclaimincident` *integer* | Use this property to control the ticket reclaim process for outbound processing.<br>Set the **Gate.ServiceNow.ReclaimIncident** property to `0` to disable.<br>Set to the **Gate.ServiceNow.ReclaimIncident** property to 1 to conditionally enable if the ticket is marked with **CreationErrorCode**.<br>Set to the **Gate.ServiceNow.ReclaimIncident** property to 2 to always enable.<br>The default is `0`. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.RetryLimit** *integer* | `-servicenowretrylimit` *integer* | Use this property to specify the maximum number of retries the gateway should make on an operation (for example, forwarding an event to the ServiceNow instance) that failed.<br>The default value of `0` (zero) means there is no limit to the number of retries that the gateway makes on a failed operation. |
| **Gate.ServiceNow.RetryWait** *integer* | `-servicenowretrywait` *integer* | Use this property to specify the number of seconds the gateway should wait before retrying an operation (for example, forwarding an event to the ServiceNow instance) that failed.<br>The default value is 7 seconds. |
| **Gate.ServiceNow.SysIdField** *string* | `-servicenowsysidfield` *string* | Use this property to specify the name for the ServiceNow target table column that holds the globally unique identifier (GUID) for each ServiceNow ticket. The values for the GUID column are crucial to the operation of the gateway as they are often used to form the URLs which the gateway uses to communicate with the ServiceNow instance.<br>The default value for the GUID column name is `sys_id`. |
| **Gate.ServiceNow.TableName** *string* | `-servicenowtablename` *string* | Use this property to specify the name of the ServiceNow target table. The gateway uses this target database table to create tickets and to retrieve data about these tickets.<br>The default name for the ServiceNow target table is `incident`. |

*Table 9. Gateway specific properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ServiceNow.TicketIdField** *string* | `-servicenowticketidfield` *string* | Use this property to instruct the gateway to use the specified column (which must be a unique ticket identifier) in the ServiceNow target table as the internal ticket identifier.<br>The default column name is `sys_id`. |
| **Gate.ServiceNow.UpdateFields** *string* | `-servicenowupdatefields` *string* | In addition to the six automatically included columns, use this property to specify which columns to retrieve record field information from the ServiceNow target table when polling for updates. The following list identifies these six other columns:<br>`sys_created_by`<br>`sys_created_on`<br>`sys_updated_on`<br>`sys_id`<br>`sys_updated_by`<br>`sys_mod_count`<br>The default value for this property specifies two columns: `state;number`.<br>**Note :** If you define the name of a column in the ServiceNow target table that can be used as an alternate index (the **Gate.ServiceNow.AlternateIndexField** property), that column is also automatically included as a retrieval column. |
| **Gate.ServiceNow.UpdateQueryStartTime** *integer* | `-servicenow updatequerystarttime` *integer* | The range of **Gate.ServiceNow.UpdateQueryStartTime** is 0 to 10,000 minutes. |

| *Table 9. Gateway specific properties (continued)* | | |
|---|---|---|
| **Property name** | **Command Line option** | **Description** |
| **Gate.ServiceNow.Username** *string* | -servicenowusername *string* | Use this property to specify the username for the user account that the gateway uses to access the ServiceNow instance. This user account must be created in ServiceNow and consist of a username and password. Use this property in conjunction with the **Gate.ServiceNow.Password** property.<br>The default value is " ".<br>See "Authenticating the gateway with ServiceNow" on page 7 for additional information about this property and its relationship to defining suitable ServiceNow user roles. |

Table 10 on page 55 lists and describes the properties provided by the gateway framework.

| *Table 10. Gateway framework properties* | | |
|---|---|---|
| **Property name** | **Command Line option** | **Description** |
| **Gate.Cache.NotificationMap** *string* | -cachenotificationmap *string* | Use this property to specify the JavaScript file to use for alert notifications.<br>The default value is $OMNIHOME/gates/servicenow/servicenow.notification.js. |
| **Gate.Cache.GenericScript** *string* | -cachegenericscript *string* | Use this property to specify the JavaScript file to use for alert modifications.<br>The default value is $OMNIHOME/gates/servicenow/servicenow.generic.js |

*Table 10. Gateway framework properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.Cache.ResyncMode** *string* | -cacheresyncmode *string* | Use this property to specify a resynchronization mode for the gateway. This property takes the following value:<br>UNI: The gateway performs a resynchronize operation from Tivoli Netcool/OMNIbus to ServiceNow.<br>The default value is UNI.<br>The gateway does not currently support the values BI and AUTO. |
| **Gate.Cache.ResyncSuppressDeletes** *string* | -cacheresyncsuppress deletes *string* | Use this property to suppress all resynchronization deletes. This property takes the following values:<br>• TRUE: Suppresses all resynchronization deletes.<br>• FALSE: Does not suppress all resynchronization deletes.<br>The default value is FALSE. |
| **Gate.Cache.Resync SuppressInitialDeletes** *string* | -cacheresyncsuppress initialdeletes *string* | Use this property to suppress initial (gateway startup) resynchronization deletes. This property takes the following values:<br>• TRUE: Suppresses initial (gateway startup) resynchronization deletes.<br>• FALSE: Does not suppress initial (gateway startup) resynchronization deletes.<br>The default value is FALSE. |

*Table 10. Gateway framework properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.ConversionsDirection** *string* | -conversionsdirection *string* | Use this property to specify how the gateway converts field values.<br>This property takes the following values:<br>BOTH: The gateway converts in both forward and reverse directions. Conversions generally convert integer ObjectServer values to and from target specific string values, typically converting Severity ("Clear", "Indeterminate", "Warning", "Minor", "Major", "Critical") from its integer representation to a string.<br>FORWARD: The gateway only converts from OMNIbus integers to target string.<br>REVERSE: The gateway only converts from target strings to OMNIbus integers.<br>The default value is REVERSE.<br>**Note :** The gateway does not supply a conversion table. However, you can create a conversion table to make use of this property. |
| **Gate.HistoricResync** *boolean* | -historicresync *boolean* | Use this property to configure how the gateway handles historic journal entries for an ObjectServer alert. The property takes one of the following values:<br>• `true`: Instructs the gateway to transfer existing journal entries when creating a ticket in ServiceNow.<br>• `false`: Instructs the gateway to not transfer existing journal entries when creating a ticket in ServiceNow.<br>The default value is `true`. |
| **Gate.MapFile** *string* | -mapfile *string* | Use this property to specify the map definition file for the gateway to use.<br>The default value is `$OMNIHOME/gates/servicenow/servicenow.map`. |

*Table 10. Gateway framework properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.PackageBase** *string* | -packagebase *string* | Use this property to specify the Java gateway package.<br>The default value is<br>com.ibm.tivoli.netcool.integrations.gateway.<br>**Note :** This is an internal property and its value should not be changed. |
| **Gate.RdrWtr.DetailsTableName** *string* | -detailstablename *string* | The gateway does not currently support this property. |
| **Gate.RdrWtr.IducFlushRate** *integer* | -iducflushrate *integer* | Use this property to specify the rate (in seconds) of the granularity of the gateway's reader. If you set this property to 0 (zero), the reader gets its updates at the same granular rate as that of the ObjectServer to which it is connected.<br>The default value is 0.<br>**Note :** If you set this property to a value greater than 0, the reader issues automatic IDUC flush requests to the ObjectServer with this period. 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* | -journaltablename *string* | Use this property to specify the name of the journal table that the gateway reads.<br>The default value is alerts.journal.<br>**Note :** The value for this property must be alerts.journal and should not be changed. |

*Table 10. Gateway framework properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **Gate.RdrWtr.Password** *string* | -password *string* | Use this property to specify the login password for the ObjectServer. The **Gate.RdrWtr.Password** is associated with the user specified by the **Gate.RdrWtr.Username** property.<br>The default value is "".<br>**Note :** The gateway uses this property only when the ObjectServer runs in secure mode. |
| **Gate.RdrWtr.ReconnectTimeout** *integer* | -reconnecttimeout *value*<br>-noreconnecttimeout | 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.<br>The default value is 30 seconds. You can use -noreconnecttimeout to set the value to 0 (zero) to indicate that the gateway does not try to reconnect. |
| **Gate.RdrWtr.Server** *string* | -server *string* | Use this property to specify the name of the ObjectServer from which the gateway reads alerts. The name can either be an interface name (for example, NCOMS) or the <host>:<port> details of the ObjectServer.<br>The default value is localhost:4100. |
| **Gate.RdrWtr.StatusTableName** *string* | -statustablename *string* | Use this property to specify the name of the status table to which the gateway reads and writes.<br>The default value is alerts.status. |

| *Table 10. Gateway framework properties (continued)* | | |
|---|---|---|
| **Property name** | **Command Line option** | **Description** |
| **Gate.RdrWtr.TblReplicateDefFile** *string* | -tblrepfile *string* | Use this property to specify the path to the table replication definition file. The path includes the name of the table replication definition file.<br>The default value (including the name of the table replication definition file supplied with the gateway) is $OMNIHOME/gates/servicenow/servicenow.rdrwtr.tblrep.def. |
| **Gate.RdrWtr.Username** *string* | -username *string* | Use this property to specify the username used to authenticate the ObjectServer connection.<br>The default value is root.<br>**Note :** The gateway uses this property only when the ObjectServer runs in secure mode. |
| **Gate.TargetIdField** *string* | -targetidfield *string* | Use this property to specify the alerts.status column to use for target identifier tracking (that is, the identifier in ServiceNow).<br>The default value is ServiceNowSysId.<br>**Note :** The identifier is the globally unique identifier (GUID) for each ServiceNow ticket. |
| **Gate.TraceFile** *string* | -tracefile *string* | Use this property to specify the path of the file used to store trace information.<br>The default value is "".<br>**Note :** Typically, this property is used by IBM support to debug any issues with gateway operations. |

| Table 10. Gateway framework properties (continued) | | |
|---|---|---|
| **Property name** | **Command Line option** | **Description** |
| **Gate.TraceMode** *string* | -tracemode *string*<br>-traceread<br>-tracewrite<br>-tracenone | Use this property to specify the gateway trace mode.<br>This property takes the following values:<br>READ: Enables trace reading.<br>WRITE: Enables trace writing.<br>NONE: Does not enable trace reading or writing.<br>The default value is NONE.<br>**Note :** Typically, this property is used by IBM support to debug any issues with gateway operations.<br>You can use the command line options -traceread, -tracewrite, or -tracenone to set the value to READ, WRITE, or NONE, respectively. |
| **Gate.Type** *string* | -type *string* | Use this property to specify the gateway type.<br>The default value is servicenow.<br>**Note :** The value for this property must be servicenow and should not be changed. |

Table 11 on page 61 lists and describes the generic Tivoli Netcool/OMNIbus properties that the gateway uses. For details on these properties, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

| Table 11. Generic OMNIbus properties | | |
|---|---|---|
| **Property name** | **Command Line option** | **Description** |
| **ConfigKeyFile** | | See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for information on this property. |
| **HttpServer.Password** *string* | | This property is not currently used. |
| **HttpServer.Port** *integer* | | This property is not currently used. |
| **HttpServer.Realm** *string* | | This property is not currently used. |
| **HttpServer.Root** *string* | | This property is not currently used. |
| **HttpServer.Username** *string* | | This property is not currently used. |

| *Table 11. Generic OMNIbus properties (continued)* | | |
|---|---|---|
| **Property name** | **Command Line option** | **Description** |
| **MaxLogFileSize** | | See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for information on this property.<br>**Note :** The default maximum size for the message log file is `0` (meaning that the file size is limitless). |
| **MessageLevel** *string* | `-messagelevel` *string* | See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for information on this property.<br>**Note :** The default message level is `warn:warn`. |
| **MessageLog** *string* | `-messagelog` *string* | See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for information on this property.<br>**Note :** The default message log file is `$OMNIHOME/log/G_SERVICENOW.log`. |
| **MessageLogRollKeep** *integer* | `-messagelogrollkeep` *integer* | Use this property to specify the maximum number of old rolled logs to keep. To specify how older logs are processed, use the **MessageLogRollProcess** property.<br>The default is 5. |
| **MessageLogRollProcess** *string* | `-messagelogrollprocess` *string* | Use this property to specify how old logs are processed.<br>The default is ''.<br>**Note :** If no value is set for this property, old logs are simply deleted.<br>If a command is specified, the old log is passed as an argument to the command. The command might be `gzip` (to compress old files) or a custom user script to archive old log files in offline storage.<br>This functionality helps you to prevent the unrestricted build-up of old log files. |

*Table 11. Generic OMNIbus properties (continued)*

| Property name | Command Line option | Description |
|---|---|---|
| **MessageLogRolltime** *string* | -messagelogrolltime *string* | Use this property to specify at what time a daily log file should roll. Typically, this is done at midnight, to generate daily logs. However, you can specify any time. Specify the time in *HHMM* format.<br>Where: *HH* specifies the hour and *MM* specifies the minutes of the time you want to roll the daily log.<br>For example, the value `'0000'` specifies that the log file rolls at midnight. The value `'0430'` specifies that the log file rolls at 04:30.<br>The default daily log file roll time is `""`. |
| **Name** *string* | -name *string* | See the *IBM Tivoli Netcool/ OMNIbus Probe and Gateway Guide* for information on this property.<br>**Note :** The default name of the current gateway instance is `servicenow`. |
| **Props.AllowUnknown** *boolean* | -propsallowunknown *boolean* | Use this property to instruct the gateway on how to handle unknown properties in the properties file. Specify one of the following values:<br>`true`: Instructs the gateway to allow unknown properties in the properties file and to ignore these unknown properties.<br>`false`: Instructs the gateway to generate an error if there are unknown properties in the properties file. Use this setting to catch such errors as typos in property names and to avoid the diagnostics that follow typographical related errors.<br>The default value is `false`. |

| Table 11. Generic OMNIbus properties (continued) | | |
|---|---|---|
| **Property name** | **Command Line option** | **Description** |
| **PropsFile** *string* | `-propsfile` *string* | See the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for information on this property.<br>**Note :** The default location and name is `$OMNIHOME/etc/G_SERVICENOW.props`. |

## Error messages

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

The following table describes the error messages that the Java Gateway for ServiceNow generates. Many of these errors relate to the gateway's interaction with ServiceNow.

| Table 12. Error messages | | |
|---|---|---|
| **Error** | **Description** | **Action** |
| `Action forbidden - please double-check the value of 'Gate.ServiceNow.Username' and 'Gate.ServiceNow.Password'.` | The gateway request reached the ServiceNow instance, but something is wrong with the data the gateway provided. Specifically, the ServiceNow instance does not permit the requested operation for the specified user. | One way to correct this error is to check the values that you specified for the **Gate.ServiceNow.Username** and **Gate.ServiceNow.Password** properties to ensure that the user account that the gateway uses to access the ServiceNow instance is valid.<br>Also, verify that the user has created all the required roles.<br>For more information on these properties, see “Authentication” on page 7, “Connecting to ServiceNow” on page 9, and Table 9 on page 43. |
| `Gate.ServiceNow.PreCreateChecks value '`*`prop_value`*`' is invalid.`<br>`Permitted values are 'default' or 'disable'.` | You specified an invalid value for the **Gate.ServiceNow.PreCreateChecks** property. | To correct the issue, specify one of the values `default` or `disable` for the **Gate.ServiceNow.PreCreateChecks** property.<br>For more information on the **Gate.ServiceNow.PreCreateChecks** property, see “Controlling how the gateway performs checks before it creates a ticket” on page 39 and Table 9 on page 43. |

*Table 12. Error messages (continued)*

| Error | Description | Action |
|---|---|---|
| `Gate.ServiceNow.Outbound Management value 'prop_value' is invalid.`<br>`Permitted values are 'default', 'disable' or 'custom'.` | You specified an invalid value for the **Gate.ServiceNow.Outbound Management** property. | To correct the issue, specify one of the values `default`, `disable`, or `custom` for the **Gate.ServiceNow.Outbound Management** property.<br>For more information on the **Gate.ServiceNow.Outbound Management** property, see "Controlling how the gateway handles outbound data management" on page 37 and Table 9 on page 43. |
| `Ignoring unknown ticket: targetId` | The gateway has read an update in the ServiceNow instance and extracted *targetId* (the ID of the ticket). This ID is the column (which must be a unique ticket identifier) in the ServiceNow target table specified in the **Gate.ServiceNow.TicketIdField** property.<br>The gateway tries to retrieve the corresponding alert from the ObjectServer alerts.status table (by checking the *targetId* of each alert). In this case, the corresponding alert cannot be found. | As a result, the gateway does not apply the update it just read from the ServiceNow instance to the corresponding alert in the alerts.status table. It could be that in the meantime the alert was deleted from the ObjectServer alert.status table. |
| `Incorrect credentials - please double-check the value of 'Gate.ServiceNow.Username' and 'Gate.ServiceNow.Password'.` | The gateway request reached the ServiceNow instance, but something is wrong with the data the gateway provided. Specifically, the ServiceNow instance does not permit the specified user to make use of the ServiceNow API. | One way to correct this error is to check the values that you specified for the **Gate.ServiceNow.Username** and **Gate.ServiceNow.Password** properties to ensure that the user account that the gateway uses to access the ServiceNow instance is valid.<br>Also, verify that the user has created all the required roles.<br>For more information on these properties, see "Authentication" on page 7, "Connecting to ServiceNow" on page 9, and Table 9 on page 43. |

*Table 12. Error messages (continued)*

| Error | Description | Action |
|---|---|---|
| One of the following three key properties has no value:<br>Property 'Gate.ServiceNow.Host' : *host*<br>Property 'Gate.ServiceNow.Username' : *username*<br>Property 'Gate.ServiceNow.Password' : *password*<br>Therefore, the Gateway won't be able to connect to the Service Now server. | The gateway requires valid values for the **Gate.ServiceNow.Host**, **Gate.ServiceNow.Username**, and **Gate.ServiceNow.Password** properties to successfully connect to the ServiceNow server. The gateway has detected that one of these properties has no value specified. | Ensure that the **Gate.ServiceNow.Host**, **Gate.ServiceNow.Username**, and **Gate.ServiceNow.Password** properties contain valid values to ensure that the user account that the gateway uses to access the ServiceNow instance is valid.<br>For more information on these properties, see "Authentication" on page 7, "Connecting to ServiceNow" on page 9, and Table 9 on page 43. |
| Something went wrong while processing the notification | The gateway performed a polling update operation for the ServiceNow instance. One of the tasks associated with this operation is to retrieve a list of updates or events from the ServiceNow instance and, if valid, update this information on the ObjectServer.<br>The error message indicates that the gateway's attempt to update the ObjectServer failed. | There is most likely a transient error. There is no specific action to be taken. This is a symptom of an interrupted REST request. |
| Something went wrong while trying to extract *sysIdField* and *ticketIdField* from the reply | The gateway sent an HTTP CREATE request to the ServiceNow instance. The ServiceNow instance sent a copy of the ticket back to the gateway. However, an undetermined error occurred and the gateway was unable to extract the values from the *sysIdField* and *ticketIdField* contained in the copy of the ticket. | Turn the log to debug, and check the reply sent by the ServiceNow instance. If the attributes referred to by the *sysIdField* and *ticketIdField* are not present, check the values for the **Gate.ServiceNow.SysIdField** and **Gate.ServiceNow.TicketIdField** properties to ensure that they have not been altered from their default values.<br>For more information on these properties, see "Controlling how the gateway handles processing of ServiceNow tickets" on page 40, "Controlling how the gateway handles operations related to ServiceNow ticket fields" on page 40, and Table 9 on page 43. |

*Table 12. Error messages (continued)*

| Error | Description | Action |
|---|---|---|
| `Table 'table' was not found - please double-check the value of 'Gate.ServiceNow.TableName'.` | The gateway was attempting to validate the name of the ServiceNow target table specified in the **Gate.ServiceNow.TableName** property with the ServiceNow instance. However, the ServiceNow instance sent back an error indicating that the specified ServiceNow target table could not be found. | Ensure that the **Gate.ServiceNow.TableName** property contains a valid name for the ServiceNow target table and verify that the table is present in the ServiceNow instance. Also, verify that this ServiceNow target table is accessible to the user account for the gateway.<br>For more information on this property, see "Controlling which ServiceNow target table the gateway uses" on page 35 and Table 9 on page 43. |
| `There are multiple ServiceNow records correlated with server name srvName server serial srvSerial` | The gateway sent a request to the ServiceNow instance to retrieve the *sysId* of a ticket based on its *correlation_id* value. However, the ServiceNow instance returned several matching values. | Ensure that the *correlation_id* (or more accurately the AlternateIndex) is set to a unique value per alert.<br>**Note :** This error is most likely to occur when the alternate index policy (the **Gate.ServiceNow.AlternateIndexPolicy** property) is set to the value `custom`. |
| `Ticket could not be located in ServiceNow.` | The gateway attempted to update a ticket in the ServiceNow target table. However, the gateway could not locate the ticket in ServiceNow. | The user deleted the ticket in the ServiceNow instance. One suggestion is to check the history of the ticket in the ServiceNow instance. |
| `Unable to reach the ServiceNow server - please double-check the value of 'Gate.ServiceNow.Host'.` | The gateway request was unable to be delivered to the ServiceNow instance. It is likely that the host name for the ServiceNow instance was incorrectly entered in the **Gate.ServiceNow.Host** property. | To correct the error, check the value you specified for the **Gate.ServiceNow.Host** property in the `G_SERVICENOW.props` properties file.<br>For more information on this property, see "Connecting to ServiceNow" on page 9 and Table 9 on page 43. |
| `No retry for REST PUT as response status code is XXX` | The gateway received a response error code of XXX (where XXX is 403,404,429, etc.) for a REST PUT operation and will not retry that REST PUT operation. | This error message informs that the gateway will not retry REST queries returned with the error codes defined in **Gate.ServiceNow.NoRetryCodes**. If this is not correct, please update this property accordingly and restart the gateway. |

| *Table 12. Error messages (continued)* | | |
|---|---|---|
| **Error** | **Description** | **Action** |
| `Please verify that the proxy was correctly specified in 'Gate.ServiceNow.Proxy' and resolve any connectivity issues with this proxy.` | The gateway was unable to connect to ServiceNow using the proxy specified in **`Gate.ServiceNow.Proxy`**. | Confirm that the proxy credentials is correct, proxy is reachable and running and ensure that the proxy details is correctly specified in **`Gate.ServiceNow.Proxy`**. Refer to "Properties and command line options" on page 43. |
| `Please verify that the proxy was correctly specified with the authentication credentials in 'Gate.ServiceNow.Proxy'.` | Gateway received HTTP response code 407 from the proxy and is unable to connect to ServiceNow due to incorrect proxy credentials. | Ensure that the correct proxy credentials have been specified in **`Gate.ServiceNow.Proxy`**. Refer to "Properties and command line options" on page 43. |

## GatewayWatch messages

During normal operation, the gateway generates GatewayWatch messages and sends them to the ObjectServer. These messages inform the ObjectServer how the gateway is running.

The following table describes the GatewayWatch messages that the Java Gateway for ServiceNow generates. These GatewayWatch messages relate to the gateway's interaction with ServiceNow.

| *Table 13. GatewayWatch messages* | | |
|---|---|---|
| **GatewayWatch message** | **Description** | **Triggers/causes** |
| `Problem while querying ServiceNow for updates: exception` | While the gateway is running, it connects to and queries the ServiceNow server for alerts, in this case updates to ServiceNow target table records. This is an indication back to the ObjectServer that there is some issue with the gateway. | This GatewayWatch message gets sent to the ObjectServer when the gateway update query to the ServiceNow server fails.<br>The *exception* specifies a message that describes why this particular update query failed. For example:<br>`Problem while querying ServiceNow for updates: ERROR (90): 90; Connection refused connect 9.70.60.40..` |
| `Target connection restored` | A transient problem with communication to the ServiceNow instance has been resolved or no longer exists. | This GatewayWatch message gets sent to the ObjectServer when the transient problem with communication to the ServiceNow instance has been resolved or no longer exists. |
| `Target connection down` | A prolonged interruption with the ServiceNow REST service is occurring. | This GatewayWatch message gets sent to the ObjectServer when a ServiceNow REST service occurs. |

# 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:

SC27-8720-05