IBM® Tivoli® Netcool/OMNIbus SNMP Probe 22.0

Reference Guide
October 30, 2020

SC11-7728-14

Note

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 61.

Edition notice

This edition (SC11-7728-14) applies to version 22.0 of IBM Tivoli Netcool/OMNIbus SNMP Probe and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC11-7728-13.

Contents

About this guide ..... **v**
- Document control page v
- Conventions used in this guide xi

Chapter 1. SNMP Probe **1**
- Summary 1
- Installing probes 2
- Getting started 3
- Configuring the probe 4
- SNMP engine update automation processing 6
  - Enabling SNMP engine update automation 7
  - SNMP engine information ProbeWatch messages 8
  - NHttp Bidirectional commands on the SNMP engine 8
- Firewall considerations 9
- SNMP V3 support 10
  - Configuration files 11
  - Probe properties 11
  - Defining the startup behavior 11
  - Adding new users 13
  - Traps and informs 16
  - Probe initialization 16
  - Limiting processing to SNMP V3 only 17
  - Minimum security level for trap processing 17
  - Example configuration 18
- Data acquisition 20
  - ObjectServer information 20
  - Communications information 21
  - Paths used by the probe 23
  - Queue and buffer settings 23
  - Logging performance data 24
  - Monitoring flood conditions at IP addresses 25
  - Generic trap handling 25
  - Logging lost traps 26
  - Heartbeat 26
  - Support for Unicode and non-Unicode characters 26
  - Rules files 28
  - Peer-to-peer failover functionality 29
- Managing the SNMP agent over the probe's HTTP/HTTPS interface 30
  - Enabling remote management of the probe 30
  - Sending commands using **nco_http** 31
  - Sending commands using **nco_probeeventfactory** 35
  - Storing commands in `nco_http.props` properties file 37
  - Rules file 38
- Properties and command line options 39
- Elements 48
  - Static elements 48
  - Dynamic elements 50
- Error messages 51
- ProbeWatch messages 55
- Running the probe 56

Running the probe as suid root 57
Running the probe on AIX 57
Known Issues 57
Troubleshooting 59

Appendix A. Notices and Trademarks 61
Notices 61
Trademarks 62

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 SNMP Probe documentation is provided in softcopy format only. To obtain the most recent version, visit the IBM Tivoli Netcool Knowledge Center:

https://www.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/common/Probes.html

Table 1. Document modification history

Document version	Publication date	Comments
SC23-6003-00	October 10, 2007	First IBM publication.
SC23-6003-01	December 29, 2007	VRM number updated. Summary table updated. Note about the lack of support for V3 privacy on the Windows version of the probe removed. The Windows version of the probe now supports V3 privacy and authentication. List of supported traps and informs clarified. List of statistics logged by the probe updated. Tip about setting the **TrapQueueMax** property to avoid dropping events during a trap burst added.
SC23-6003-02	July 25, 2008	Support for Linux® for zSeries added. IPv6 support information added. Information about the compatibility of the probe with Federal Information Processing Standards (FIPS) added.
SC23-6003-03	April 30, 2009	Support for the Configuration Analyser added. Summary section updated. Installing the probe topic added. Installing the Configuration Analyser topic added. Configuring the probe environment topic updated. Running the Configuration Analyser topic added.
SC11-7728-00	November 6, 2009	Functionality added to enable the probe to perform additional processing on rules files.

Table 1. Document modification history (continued)		
Document version	**Publication date**	**Comments**
SC11-7728-01	July 1, 2011	FIPS compatability statement added to “Summary” on page 1. Installation section replaced by “Installing probes” on page 2. Prerequisites information added to “Running the probe as suid root” on page 57. SNMP GET and SET operations updated in “Rules file processing” on page 28.
SC11-7728-02	July 6, 2012	Changes made to enable you configure the probe to meet the Common Criteria assurance level of EAL4+. The Common Criteria version of this guide is available from the IBM Passport Advantage website. http://www.google.com
SC11-7728-03	November 30, 2012	Added note regarding Common Criteria version of this probe to Chapter 1, “SNMP Probe,” on page 1. Added “Firewall considerations” on page 9. Added information on properties that define the locations of the `mttrapd.conf` file and the hashed file to “Adding new users to the configuration file” on page 13. Enhanced “Example configuration” on page 18. SNMP GET and SET operations updated in “Rules file processing” on page 28. Added “Managing the SNMP agent over the probe's HTTP/HTTPS interface” on page 30. Added **DSALogs** and **DSAPeriod** properties to “Properties and command line options” on page 39. Updated “Error messages” on page 51. Added “Known Issues” on page 57.
SC11-7728-04	March 14, 2013	Add information on get and set functions for SNMP V2c and V3 to “Rules file processing” on page 28. Updated “Managing the SNMP agent over the probe's HTTP/HTTPS interface” on page 30 to include get and set commands for SNMP V2c and V3. Updated “Static elements” on page 48 to include the elements specific to remotely managing the SNMP agent over HTTP/HTTPS. Updated the “Summary” on page 1 and “Known Issues” on page 57 to include information on IPv6 support on the Microsoft Windows operating system. Updated “Error messages” on page 51.

Table 1. Document modification history (continued)

Document version	Publication date	Comments
SC11-7728-05	July 5, 2013	Added "Configuration files" on page 11, "Probe properties" on page 11, "Defining the startup behavior" on page 11, and "Probe initialization " on page 16 to "SNMP V3 support" on page 10. Updated "Adding new users to the configuration file" on page 13 in "SNMP V3 support" on page 10. Added **ReuseEngineBoots**, **SnmpConfigChangeDetectionInterval** and **UsmUserBase** properties to "Properties and command line options" on page 39. Updated "Running the probe as suid root" on page 57. **Fixes:** Version 16 of the SNMP Probe includes fixes for the following APARs: • **IV49788**: Rebuilt probe to incorporate changes made in **IV40476**.
SC11-7728-06	November 8, 2013	Updated "Configuration files" on page 11, "Probe properties" on page 11, "Adding new users to the configuration file" on page 13, and "Probe initialization " on page 16 in "SNMP V3 support" on page 10. Updated "Paths used by the probe" on page 23 and added "Logging lost traps" on page 26 in "Data acquisition" on page 20. Updated "Error messages" on page 51. Updated "Running the probe as suid root" on page 57 in "Running the probe" on page 56. **Fixes:** Version 17 of the SNMP Probe includes fixes for the following APARs: • **IV50344**: Fixed virtual memory growth issue when realtime configuration update feature is enabled.

Table 1. Document modification history (continued)

Document version	Publication date	Comments
SC11-7728-07	January 24, 2014	Topics added: "Getting started" on page 3 "Configuring the probe" on page 4 "Limiting processing to SNMP V3 only" on page 17 "Minimum security level for trap processing" on page 17 "ObjectServer information" on page 20 "Communications information" on page 21 "Logging performance data" on page 24 "Heartbeat" on page 26 "Support for Unicode and non-Unicode characters" on page 26 Topics updated: "SNMP V3 support" on page 10 "Queue and buffer settings" on page 23 **NoNameResolution**, **snmpv3ONLY**, **SocketSize**, and **TrapQueueMax** in "Properties and command line options" on page 39 "Running the probe" on page 56 **Fixes:** Version 18 of the SNMP Probe includes fixes for the following APARs: • **IV63794**: Added support for the DSALog properties. • **IV66556**: Hostname resolution failed on 64-bit Solaris and Linux.
SC11-7728-08	December 11, 2014	"Summary" on page 1 updated. "Monitoring flood conditions at IP addresses" on page 25 added. Description for the **TrapStat** property added to "Properties and command line options" on page 39. "Error messages" on page 51 and "ProbeWatch messages" on page 55 updated. **Fixes:** Version 19 of the SNMP Probe includes fixes for the following APARs: • **IV65309**: Collect incoming trap statistics and identify event flood in probe.

Table 1. Document modification history (continued)

Document version	Publication date	Comments
SC11-7728-09	March 12, 2015	"Summary" on page 1 updated. "Writing resolved host names and discarded host names to flat files" on page 21 added. Description for the following properties were added to "Properties and command line options" on page 39: • **ActiveHostnameDuration** • **RefreshHostnameInterval** • **HostnameTableSize** "Error messages" on page 51 updated. **Fixes:** Version 20 of the SNMP Probe includes fixes for the following APARs: • **IV68119**: Fix on memory freeing sequence for trap stat feature. • **IV68690**: Provide a file identifying probe-specific rules functions for use by the Syntax Probe.
SC11-7728-10	June 30, 2016	"Troubleshooting" on page 59 added.
SC11-7728-11	November 24, 2016	Known Issues updated on page 60. Troubleshooting updated on page 62.
SC11-7728-12	June 28, 2019	The following topics updated to clarify the usage of the `mttrapd.conf` file: • "Configuration files" on page 11 • "Defining the startup behavior" on page 11 • "Adding new users to the configuration file" on page 13

Table 1. Document modification history (continued)

Document version	Publication date	Comments
SC11-7728-13	October 31, 2019	Guide updated for version 21 of the SNMP Probe. "Summary" on page 1 updated. Support for the Configuration Analyser deprecated. References to the Configuration Analyser and `mttrapd_check.jar` removed. The following topics added: • "SNMP engine update automation processing" on page 6 • "Enabling SNMP engine update automation" on page 7 • "SNMP engine information ProbeWatch messages" on page 8 • "NHttp Bidirectional commands on the SNMP engine" on page 8 Description for the following properties were added to "Properties and command line options" on page 39: • **`ConfigCryptoAlgo`** • **`ConfigCryptoKeyFile`** • **`EnableCryptoConfig`** • **`EngineProbeWatch`** **Fixes and enhancements:** Version 21 of the SNMP Probe includes fixes for the following APARs: • **`IJ01162`**: Fixed issue on probe crashing when a trap fails IP format validation. • **`IV94048`**: Fixed issue on probe not using the specified port number for `snmpset_p` and `snmpget_p` function. • **`IV70329`**: Probe now will not check availability of the connection when Heartbeat property is set to 0. • **`IJ16506`**: Fixed issue which made probe not able to receive traps with SHA-256 authentication type. • **`IV76246`**: Fixed vulnerability in Net-SNMP due to known defect documented in Bug #2615: http://sourceforge.net/p/net-snmp/code/ci/f23bcd3ac6ddee5d0a48f9703007ccc738914791. • **`IJ16506`**: Fixed issue which made probe not able to receive traps with SHA-256 authentication type. **`IJ16407`**: Segmentation fault caused by the attempt to process non-existent buffer when mttrapd.conf is unavailable, or fails to be opened by the probe. This version of the probe also includes the following enhancements: • **`RFE 126930`**: Probe enhanced so that `mttrapd` needs no reboot when remote devices reboot with old boot. Also the following commands added: `update_engine` HTTP command to allow users to update Engine Boots and Engine Time of the existing SNMPv3 engines in `ConfPath` and `print_engine_info` HTTP command to print live engine information to a specified text file. • **`RFE 119237`**: Crypto processing for SNMP configuration file (`mttrapd.conf`).

Table 1. Document modification history (continued)		
Document version	**Publication date**	**Comments**
SC11-7728-14	October 30, 2020	Guide updated for version 22 of the SNMP Probe. "Summary" on page 1 updated. **Fixes:** Version 22 of the SNMP Probe includes fixes for the following APARs: • **IJ24466**: Fixed issue whereby the probe responded with incorrect acknowledgment for SNMP Informs with AES, AES-192 and AES-256 privacy types, hence causing timeout error to the sender. • **IJ26463**: Fixed issue of ProbeWatch events for updating `engine_boot` or `engine_time` may get deduplicated, causing only either one of `engine_boot` or `engine_time` being corrected but not both. • **IJ27412**: Fixed file leak issue with the persistent configuration file when **`ConfPath`** property is unset. To avoid this issue, the probe will now disable the **`SnmpConfigChangeDetectionInterval`** feature when the **`ConfPath`** property value is unset or empty. • **IJ22889**: Fixed issue in which the probe could not start due to a required file (`mttrapd.conf`) not being created by users, especially users who are not using SNMP V3. The file `mttrapd.conf` with example in remarked form is now bundled together and will be installed alongside.

Conventions used in this guide

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

Operating system-dependent variables and paths

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

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

```
$OMNIHOME/probes
```

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

```
%OMNIHOME%\probes
```

For probes supported on UNIX, Linux, and Windows operating systems, probe guides use the standard UNIX conventions for specifying environment variables and describing directory paths. When using the Windows command line with these probes, 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. Where such variables are described in the guide, both the UNIX and Windows conventions will be used.

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/probes/arch`

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

Note : This probe 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

Probes 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. SNMP Probe

The IBM Tivoli Netcool/OMNIbus SNMP Probe monitors SNMP traps and informs on both UDP and TCP sockets concurrently.

This probe has the following features that allow it to handle generic traps:

- It can handle a high volume and high rate of traps.
- It receives traps independently of trap processing, using an internal queue mechanism.
- It handles high trap rates and high burst rates using two buffers: one buffer is for all of the sockets that the probe monitors, and the other buffer is an internal queue between the reader and writer sides of the probe.
- It supports SNMP V1, V2c, and V3 traps.
- It supports SNMP V2c and V3 traps and informs.
- It uses a User-based Security Model (USM) for SNMP V3.

The following topics describe the probe and how it works:

- "Summary" on page 1
- "Getting started" on page 3
- "Installing probes" on page 2
- "Configuring the probe" on page 4
- "Firewall considerations" on page 9
- "SNMP V3 support" on page 10
- "Data acquisition" on page 20
- "Managing the SNMP agent over the probe's HTTP/HTTPS interface" on page 30
- "Properties and command line options" on page 39
- "Elements" on page 48
- "Error messages" on page 51
- "ProbeWatch messages" on page 55
- "Running the probe" on page 56
- "Known Issues" on page 57
- "Troubleshooting" on page 59

Summary

Each probe works in a different way to acquire event data from its source, and therefore has specific features, default values, and changeable properties. Use this summary information to learn about this probe.

The following table summarizes the probe.

Table 3. Summary

Probe target	SNMP traps and informs
Probe executable file name	`nco_p_mttrapd`
Probe installation package	`omnibus-`*`arch`*`-probe-nco-p-mttrapd-`*`version`*

Table 3. Summary (continued)	
Package version	22.0
Probe 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=swg21660373
Properties file	`$OMNIHOME/probes/arch/mttrapd.props`
Rules file	`$OMNIHOME/probes/arch/mttrapd.rules`
Rules file for trap flood monitoring	`$OMNIHOME/probes/arch/mttrapd_flood_control.rules`
Rules file for remote management over HTTP	`$OMNIHOME/probes/arch/mttrapd.bidir.rules`
Requirements	For details of any additional software that this probe requires, refer to the `description.txt` file that is supplied in its download package.
Connection method	Listens for SNMP traps using UDP, TCP, UDPV6, and TCPV6
Peer-to-peer failover functionality	Available
Multicultural support	Available
IP environment	IPv4 and IPv6 For communication between the probe and Netcool/OMNIbus V7.3.0 and above, IPv6 is supported on all operating systems. For communication between the probe and the target device, IPv6 is supported on all supported operating systems except Microsoft Windows.
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*.

Installing probes

All probes are installed in a similar way. The process involves downloading the appropriate installation package for your operating system, installing the appropriate files for the version of Netcool/OMNIbus that you are running, and configuring the probe to suit your environment.

The installation process consists of the following steps:

1. Downloading the installation package for the probe from the Passport Advantage Online website.

   Each probe has a single installation package for each operating system supported. For details about how to locate and download the installation package for your operating system, visit the following page on the IBM Tivoli Knowledge Center:

   http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/all_probes/wip/reference/install_download_intro.html

2. Installing the probe using the installation package.

   The installation package contains the appropriate files for all supported versions of Netcool/OMNIbus. For details about how to install the probe to run with your version of Netcool/OMNIbus, visit the following page on the IBM Tivoli Knowledge Center:

   http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/all_probes/wip/reference/install_install_intro.html

3. Configuring the probe.

   This guide contains details of the essential configuration required to run this probe. It combines topics that are common to all probes and topics that are peculiar to this probe. For details about additional configuration that is common to all probes, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Getting started

This section shows how to start the probe with the minimum required configuration. The procedure assumes that you have a version of Netcool/OMNIbus installed and running.

Use the following procedure to start the probe with a minimal configuration:

Note : The commands shown in this example are for a Linux operating system. Adapt these commands as necessary for the operating system that runs on your probe server.

1. Download the probe's installation package following the instructions in http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/all_probes/wip/reference/install_download_intro.html.
2. Install the probe following the instructions in http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/all_probes/wip/reference/install_install_intro.html.
3. Edit the probe's properties file and set values for the following items:

   `Server`: Set this property to the name of the ObjectServer that the probe sends events to. The default value of this property is `NCOMS`.

   `Port`: Set this property to the port that the probe listens to for SNMP traffic. Make sure that no other SNMP agent is using the port you choose. Note that if you use the default value of 162, you cannot run the port from a non-root user account.

   `Protocol`: Set this property to `ALL`. This setting ensures that the probe listens for V1 and V2c traps on both the UDP and TCP protocols.

   `MessageLevel`: Set this property to Debug. This setting provides the maximum amount of information when the probe is running.

   For example, if the ObjectServer is named MY_OBJ and the probe uses port 1162, the properties file would contain the following entries:

```
##############################################################################
#
# Add your settings here
#
##############################################################################
Server : 'MY_OBJ'
Port : 1162
Protocol : 'ALL'
MessageLevel : 'debug'
```

4. Ensure that `$NCHOME/etc/omni.dat` includes information on the ObjectServer.
5. If you made changes to `omni.dat`, regenerate the definition file by running `$NCHOME/bin/nco_igen`.
6. Check that the server where you have installed the probe can contact the ObjectServer using the `ping` utility. For example:

   ```
   ping -c 10 server-name
   ```

   Replace *server-name* with the name of the server where the ObjectServer is running.

   Alternatively, if you have installed the Netcool/OMNIbus desktop feature, you can use the `nco_ping` utility to check connectivity to the ObjectServer. For example:

   ```
   nco_ping object-server
   ```

   Replace *object-server* with the name of the ObjectServer.
7. Obtain a listing of the probe's command line options to check that the probe is correctly installed. For example:

   ```
   $NCHOME/omnibus/probes/linux2x86/nco_p_mttrapd -help
   ```

   Check that the output from this command begins as follows:

   ```
   Usage: opt/netcool/omnibus/probes/linux2x86/nco_p_mttrapdd [options]

   where options can be:

   -all           Listen for traps via UDP and TCP, using IPv4
   -allip6        Listen for UDP or TCP traps, using IPv6
   -any           Listen for traps via UDP and TCP, using IPv4
   -autosaf       Enable automatic Store and Forward on startup
   -beatinterval  Probe failover heartbeat interval.
   ```
8. Run the probe from the command line:

   ```
   $NCHOME/omnibus/probes/linux2x86/nco_p_mttrapd \
   -propsfile $NCHOME/omnibus/probes/linux2x86/mttrapd.props -messagelevel debug
   ```
9. Check the probe's log file to ensure the probe started correctly and is ready to receive, process, and dispatch events.

   The probe's **MessageLog** property provides the name and location of the probe's log file.
10. Where possible, use a test tool to send events to the probe and so check they are processed correctly.

    For example, use the snmptrap utility to send a number of test traps to the probe.

The probe is now successfully installed and operational. You can now configure the probe to suit your operating environment.

For example, you can change the values of the properties used in this example to their production values:

MessageLevel: Set this property to the level you require, such as `Warning`.
Protocol: Set this property to the protocol you require, such as TCP.

In addition, you can set up other facilities of the probe, such as SNMP V3, peer-to-peer failover, and remote management using the probes HTTP/HTTPS interface.

Configuring the probe

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

The following table outlines how to use the probe's properties to configure the product's features. Configuration of some features is mandatory for all installations. For those features set the properties to

the correct values or verify that their default values are suitable for your environment. Further configuration is optional depending on which features of the probe you want to use.

Table 4. Configuring the probe

Feature	Properties	See
Mandatory features:		
ObjectServer name	**Server**	"ObjectServer information" on page 20
Communications method	**Port** **Protocol** **BindAddress** **NoNameResolution** **NoNetbiosLookups**	"Communications information" on page 21
Optional features:		
SNMP V3 Enables the probe to process traps and informs in SNMP V3 format.	**ConfPath** **PersistentDir** **ReUseEngineBoots** **UsmUserBase** **SnmpConfigChangeDetection Interval** **snmpv3ONLY** **snmpv3MinSecurity**	"SNMP V3 support" on page 10
Queue and buffer settings Defines the properties of the queue of traps waiting to be processed and of the buffer of events to send to the ObjectServer.	**BufferSize** **Buffering** **FlushBufferInterval** **TrapQueueMax**	"Queue and buffer settings" on page 23
Logging lost traps Specifies whether the probe creates log messages for events that are lost because the trap queue is full.	**DSALog** **DSAPeriod**	"Logging lost traps" on page 26
Heartbeat policy Specifies whether the probe periodically checks that the connection to the SNMP endpoint is still operational.	**Heartbeat**	"Heartbeat" on page 26
Logging performance statistics Specifies whether the probe creates log messages that contain performance data.	**LogStatisticsInterval**	"Logging performance data" on page 24

Table 4. Configuring the probe (continued)			
	Feature	**Properties**	**See**
	Peer-to-peer failover pair Allows you to set up two probes to act as a failover pair to improve availability. If the master probe should stop working, the slave probe takes over until the master is available once more.	**MessageLog** **Mode** **PeerHost** **PeerPort** **PidFile** **PropsFile** **RulesFile**	"Peer-to-peer failover functionality" on page 29
	Backup ObjectServer Specifies a backup ObjectServer that the probe is to use should the connection to the primary ObjectServer fail.	**NetworkTimeout** **PollServer** **ServerBackup**	"ObjectServer information" on page 20
	HTTP/HTTPS command interface Enables the HTTP/HTTPS command interface and defines the port that it uses.	**MessageLevel** **MessageLog** **NHttpd.AccessLog** **NHttpd.EnableHTTP** **NHttpd.ListeningPort** **NHttpd.ExpireTimeout** **RulesFile**	"Managing the SNMP agent over the probe's HTTP/HTTPS interface" on page 30

SNMP engine update automation processing

To enable SNMP engine update automation, you must turn on NHttppd and assign an available port using the following properties in the `snmp.props` file:

```
NHttpd.EnableHTTP        : TRUE
NHttpd.ListeningPort     :  <any available port>
```

When enabled, SNMP engine update automation performs the following actions:

1. When inconsistencies in engine boot or engine time are detected, the probe rejects the trap and sends an SNMPUSM ProbeWatch message containing the values of `engineboot` and `enginetime` from the cache and the remote device.
2. The SNMPUSM ProbeWatch message gets processed by the `mttrapd.snmpwatch.rules` file and forwarded to the ObjectServer.
3. The ProbeWatch insertion triggers `mttrapd_set_engine_correction` to issue the command to the PA which in turn executes `mttrapd_Nhttp_SnmpActions.pl`.
4. `mttrapd_Nhttp_SnmpActions.pl` sends the **`set_engine_correction`** command with engine information using HTTP bidirection to the probe rules engine.
5. `mttrapd.bidir.rules` processes the **`set_engine_correction`** command mapping the error type to its corresponding remedy action. Typically, the remedy calls the `update_engine()` probe rules function to update `engineboot` or `enginetime` in the probe cache.

Note: Two traps may be required to update both the engine boot and time issue.

The probe will accept any engine boot and engine time which is more than the cached engine boot and engine time. That is, the engine boot and time correction only gets triggered when either the remote engine boot is lesser than the cached engine boot, or the remote engine time is less than the cached engine time.

If the remote engine boot is less than the cached engine boot, the error reported will be: `err:Remote boot count invalid`

If the remote engine time is less than the cached engine time of more than 150s, the error reported will be: `err:Message too old`

So whether it takes one trap or two traps to correct the engine boot and time issue depends on what cached values the probe has and the current values the device has.

The following table shows whether one or two traps are required to correct the engine boot and time issue.

Table 5. Examples

Cached boot	Cached time	Remote boot	Remote time	# of traps to correct issue
100	0	0	1	1 (For `err:Remote boot count invalid`)
100	20800	0	20	2 (1st time for `err:Remote boot count invalid.", 2nd time for "err:Message too old`)
100	80090	100	0	1 (For `err:Message too old`)
100	1000	200	1001	1 (For `err:Remote boot count invalid`)
100	30000	101	30001	0 (No error here, as engine boot count is meant to be incremented when engine time reaches it maximum possible value.)

Enabling SNMP engine update automation

To create the probe tools for engine update automation, perform the following steps:

1. After installation, run the following command:

   ```
   $OMNIHOME/bin/nco_sql -server server_name -user user_name -password password
   < mttrapd_create_SnmpActionTools.sql
   ```

   This command performs the following tasks:

   - Adds the two columns `SNMPAction` and `SNMPError` to `alerts.status`.
   - Creates the trigger group `mttrapd_triggers`.
   - Creates the trigger `mttrapd_set_engine_correction`.
   - Creates the procedure `mttrapd_engine_command`.
2. Set the probe property **`EngineInfoProbeWatch`** to 1.
3. Activate the Process Agent (PA).

   Now when the PA receives commands issued from the `mttrapd_set_engine_correction` ObjectServer trigger, the PA log will contain messages of the following type:

   ```
   2019-05-09T17:02:09: Information: Command '[SETGID=0]$OMNIHOME/probes/linux2x86/
   mttrapd_Nhttp_SnmpActions.pl 'http:// your_omnihost:your_port/probe/c
   ommon' set_engine_correction 'err:Remote boot count invalid.' 'engine_id=8011ab22cd33ef
   cached_boot=1 cached_time= remote_boot=1 remote_time=3
   232' '8011ab22cd33ef' ' passed to dispatch thread and RPC acknowledged.
   2019-05-09T17:02:09: Debug: Route Lookup on 'localhost' which is a ROUTE_LOOPBACK route type.
   2019-05-09T17:02:09: Information: Starting an unmanaged process. (OS External effect).
   ```

```
2019-05-09T17:02:09: Debug: Launch process, argv[0] = /opt/IBM/tivoli/netcool81/omnibus/
probes/linux2x86/mttrapd_Nhttp_SnmpActions.pl
2019-05-09T17:02:09: Debug: Launch process, argv[1] = http://your_omnihost:your_port/probe/
common
2019-05-09T17:02:09: Debug: Launch process, argv[2] = set_engine_correction
2019-05-09T17:02:09: Debug: Launch process, argv[3] = err:Remote boot count invalid.
2019-05-09T17:02:09: Debug: Launch process, argv[4] = engine_id=8011ab22cd33ef cached_boot=1
cached_time= remote_boot=1 remote_time=3232
2019-05-09T17:02:09: Debug: Launch process, argv[5] = 8011ab22cd33ef
2019-05-09T17:02:09: Debug: Launch process, uid = 0
2019-05-09T17:02:09: Debug: Launch process, gid = 0
2019-05-09T17:02:09: Debug: Successful Fork.
```

SNMP engine information ProbeWatch messages

To enable the generation of SNMP ProbeWatch message, set the **EngineInfoProbeWatch** property to 1.

When SNMP engine update automation is enabled, the probe generates following types of SNMP ProbeWatch messages:

- ProbeWatch messages that report trap processing failure due to engine boot or engine time inconsistencies.

  These messages are in the following format:

  `SNMP_WATCH<time=%d module=SNMPUSM>err:%s|engineinfo:type=%s id=%s cached_boot=%d cached_time=%d remote_boot=%d remote_time=%d`

  Note : `cached_boot` and `cached_time` are the values in the probe's cache; `remote_boot` and `remote_time` are the values extracted from the SNMP trap.
- ProbeWatch messages that indicate the successful operation of the **update_engine** command.

  These messages are in the following format:

  `SNMP_WATCH<time=%d module=LCDTIME>engine_update_successful|engine_id=%s boot_updated=%s time_updated=%s host_updated=%s`

  Note : Fields not updated will appear as `<field_name>=none`. For example, if engine boot was not updated, the ProbeWatch message will contain `boot_updated=none`; otherwise the updated value will be shown in the ProbeWatch summary. When the probe hits engine boot or engine time consistency issues during trap processing, `SNMPUSM probe watch` will appear in the event list as a problem. If the engine in question is successfully updated, `LCDTIME probe watch` will be sent to the event list as the resolution.

See the `mttrapd.snmpwatch.rules` file for the processing of these ProbeWatch messages.

NHttp Bidirectional commands on the SNMP engine

The following bidirectional commands operate on the SNMP engine:

- **print_engines**
- **get_engine_info**
- **update_engine**
- **set_engine_correction**

print_engines command:

```
$OMNIHOME/bin/nco_http -uri http://:/probe/common -datatype application/json -
data '{"eventfactory": [{"snmp_action":"print_engines", "file_path":""}]}' -
method post
```

`snmp_action`: **print_engines**

Mandatory parameter:

`$file_path`

get_engine_info command:

```
$OMNIHOME/bin/nco_http -uri http://:/probe/common -datatype application/json -
data '{"eventfactory": [{"snmp_action":"get_engine_info",
"engine_id":"<engine_id>"}]}' -method post
```

snmp_action: **get_engine_info**

Mandatory parameter:

$engine_id

update_engine command:

```
$OMNIHOME/bin/nco_http -uri http://:/probe/common -datatype application/json -
data '{"eventfactory": [{"snmp_action":"update_engine",
"engine_id":"<engine_id>", "engine_boot":"","engine_time":"",
"$engine_host":"", "send_probewatch":""}]}' -method post
```

snmp_action: **update_engine**

Mandatory parameter:

$engine_id

Optional parameters (at least one must be avaialble):

$engine_boot

$engine_time

$engine_host

$send_probewatch: 0 - disable; 1 - enable ProbeWatch on successful update_engine

set_engine_correction command:

```
$OMNIHOME/bin/nco_http -uri http://:/probe/common -datatype application/json -
data '{"eventfactory": [{"snmp_action":"set_engine_correction",
"engine_err":"<err_string>", "engine_info":"<info_string>",
"alert_id":"<aler_id>"}]}' -method post
```

snmp_action: **set_engine_correction**

Mandatory parameters:

$engine_err

$engine_info

$alert_id

Note : set_engine_correction is for the engine update automation, it is issued from the ObjectServer trigger.

Firewall considerations

When using the probe in conjunction with a firewall, configure the firewall so that the probe can connect to the target system.

Two probe properties define the communication port it uses and the communications protocols it uses:

Port
Protocol

Configure the firewall to allow communication through the port defined in the **Port** property for all the protocols specified in the **Protocol** property.

Linux example

The following example configures a Linux firewall to enable communication using the TCP and UDP protocols through port 162 (the default value of the **Port** property):

1. Make a local copy of the file containing the firewall rules:

```
cp /etc/sysconfig/iptables /root/firewall_rules
```

2. Edit the local copy of the rules and add the following:

```
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 162 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m udp -p udp --dport 162 -j ACCEPT
```

3. Load the firewall with the new rules:

```
iptables-restore < root/firewall_rules
```

4. List the rules to check they are correct:

```
iptables -L

ACCEPT tcp -- anywhere anywhere state  NEW tcp dpt:162
ACCEPT udp -- anywhere anywhere state NEW udp dpt:snmptrap
```

5. Test that the probe can receive traps with the new rules in force:
   a. Start the probe allowing all protocols for the required port:

```
$OMNIHOME/probes/nco_p_mttrapd -protocol "ALL" -port 162 -messagelevel debug
-messagelog stdout
```

   b. From a remote host, enter the following command:

      `telnet` *`probe_ip_address port`*

      Where *`probe_ip_address`* is the IP address of the probe server and *`port`* is the number of the port specified in the **Port** property.

      If the firewall is correctly configured, the probe displays the following message:

```
Error: SNMP Message (priority=3): Received broken packet. Closing session.
```

      If no message appears, check the edits to the rules file, adjust as necessary, and repeat this step.

6. Implement the changes permanently:
   a. Edit the file `/etc/sysconfig/iptables`.
   b. Add the rules you implemented in the `firewall_rules` file.
   c. Load the firewall with these new rules:

```
iptables-restore < /etc/sysconfig/iptables
```

SNMP V3 support

The probe supports SNMP V3 traps and informs using User-based Security Model (USM) for authentication and privacy. USM requires that you specify a user that can log on to the probe for each security name from which the probe receives traps. You also specify a unique user or engine ID for each trap source from which the probe receives traps.

The probe stores information on users in configuration files. It uses the information in these files when it starts up to load the necessary data, which is determined by two probe properties.

This section contains information on the following topics:

- Configuration files
- Probe properties

- Defining the startup behavior
- Adding new users to the configuration file
- Traps and informs
- Probe initialization
- Limiting processing to SNMP V3 only
- Minimum security level for trap processing

In addition, there is an example of configuring a probe for SNMP V3.

Note : When the probe is running SNMP V3, it is compatible only with FIPS 140-2 if it is using AES and SHA.

Configuration files

There are two configuration files that hold user data.

The `mttrapd.conf` file resides in the directory specified by the **ConfPath** property. You use this file to add the identities of new users (endpoints) that are authorized to send traps and informs to the probe. This is known as the non-persistent configuration file.

The probe also uses a persistent version of the `mttrapd.conf` file. It contains hashed values of the contents of the non-persistent versions of the `mttrapd.conf` file. This file is also named `mttrapd.conf`, but resides in the directory specified by the **PersistentDir** property, and is known as the persistent configuration file. You do not edit this file, and make changes only to the non-persistent configuration file.

The way the probe uses these files depends on the settings of various properties. By default, the probe detects whether there are changes made to the non-persistent configuration file, uses the results to replace the user entries in the persistent file. For more information on how the probe uses these configuration files, see "Defining the startup behavior" on page 11, and in "Using the automatic detection facility" on page 13.

Note :

- To create the initial, persistent configuration file, edit the non-persistent configuration file and then start the probe.
- Ensure that the **ConfPath** and **PersistentDir** properties refer to separate directories.
- Use the host operating system's file protection facilities to ensure that the **ConfPath** and **PersistentDir** directories are adequately secured.

Probe properties

Two properties (**ReuseEngineBoots** and **UsmUserBase**) define how the probe processes the configuration files.

- **ReuseEngineBoots**: This property controls whether the probe reuses its local engine identifier (oldEngineID) or creates a new identifier. It also determines whether the probe increments the engine boots counter or resets it to one when the probe starts.
- **UsmUserBase**: This property defines how the probe processes the configuration files on startup.

"Properties and command line options" on page 39 contains details of the values that these properties can have.

Defining the startup behavior

You can use the **UsmUserBase** and **ReuseEngineBoots** properties to define how the probe processes the configuration files on startup.

The following table shows how the values of the properties define the way that the probe processes the configuration files.

Table 6. Defining the probe's startup behavior

Value of **UsmUserBase**	Value of **ReuseEngineBoots**	Probe action
0	0	1. Generate a replacement for the local engine identifier and store that in the persistent configuration file. 2. Set the value of engine reboots to one, and store that in the persistent configuration file. 3. Read the user entries from the non-persistent configuration file, and replace the user entries in the persistent configuration file. For any user entries that do not include an engine identifier, use the local engine identifier generated in step 1. 4. Use the user entries read from the non-persistent configuration file.
	1	1. Increment the value of engine boots and store it in the persistent configuration file. 2. Read the user entries from the non-persistent configuration file, and replace the user entries in the persistent configuration file. For any user entries that do not include an engine identifier, use the local engine identifier read from the persistent configuration file. 3. Use the user entries read from the non-persistent configuration file.
1	0	1. Generate a replacement for the local engine identifier and store it in the persistent configuration file. 2. Set the value of engine reboots to one, and store it in the persistent configuration file. 3. Use the user entries in the persistent configuration file.
	1	1. Increment the value of engine boots and store it in the persistent configuration file. 2. Use the local engine identifier in the persistent configuration file. 3. Use the user entries in the persistent configuration file.

Table 6. Defining the probe's startup behavior (continued)		
Value of UsmUserBase	**Value of ReuseEngineBoots**	**Probe action**
2	0	1. Generate a replacement for the local engine identifier and store it in the persistent configuration file. 2. Set the value of engine reboots to one, and store it in the persistent configuration file. 3. Read the user entries from the non-persistent configuration file and store them in the persistent configuration file in addition to the entries that file already contains. For any user entries that do not include an engine identifier, use the local engine identifier generated in step 1. 4. Use the user entries in the persistent configuration file and the user entries read from the non-persistent configuration file.
	1	1. Increment the value of engine boots and store it in the persistent configuration file. 2. Read the user entries from the non-persistent configuration file, hash them, and store them in the persistent configuration file in addition to the entries that file already contains. For any user entries that do not include an engine identifier, use the local engine identifer from the persistent configuration file. 3. Use the user entries in the persistent configuration file and the user entries read from the non-persistent configuration file. **Note :** This is the procedure that the probe uses when the properties have their default values, and maintains compatibility with previous versions of the probe.

Adding new users to the configuration file

To create a new user, add a line to the non-persistent configuration file.

Using the automatic detection facility

The probe can detect changes to the non-persistent configuration file and automatically load them without any interruption to the processing of traps and informs. The non-persistent configuration file is named `mttrapd.conf` and is in the directory specified by the value of the **ConfPath** property.

To add users to `mttrapd.conf`, use the following procedure:

1. If you have not already done so, edit the probe's property file and set the value of the **SnmpConfigChangeDetectionInterval** property to the interval (in minutes) that the probe checks for changes to the configuration file.
2. Edit `mttrapd.conf` in the directory specified by the **ConfPath** property and add the following line for each new user:

```
createUser -e engineId username authtype password privtype privpassword
```

Replace the arguments and option values as follows:

Table 7. Arguments and options for the createUser command	
engineId	Provide the engine ID of the trap source associated with the user. The engine ID is required for traps but optional for informs.
username	Provide the security name of the user.
authtype	Provide an authentication type (MD5, SHA, or SHA256). When running in FIPS 140-2 mode, use the value SHA for this option.
password	Provide the password (must be at least eight characters).
privtype	Optional: Provide the type of privacy (either DES, AES, AES192 or AES256). When running the probe in FIPS 140-2 mode, use the value AES for this option. **Note :** DES uses a 16 byte key. The probe truncates the encrypted 20 byte key to 16 bytes to use it as the DES key.
privpassword	Optional: Provide the privacy password (if different from *password*).

Two properties determine the locations of the `mttrapd.conf` file and the automatically generated non-persistent configuration file:

ConfPath defines the location of `mttrapd.conf` file.
PersistentDir defines the location of the hashed file.

It is best practice to use separate directories to hold these files. For an example of this, see "Example configuration" on page 18.

Implementation notes:

- You cannot change the value of **SnmpConfigChangeDetectionInterval** while the probe is running. To change the interval, stop the probe, change the value and restart the probe.
- The probe refreshes the entire list of users each time it detects a change in the non-persistent configuration file.
- The probe uses the non-persistent configuration file's last modified date to detect changes. So if the only change has been the addition or removal of whitespace the probe still loads the entire list of users.
- If the non-persistent configuration file is missing from either directory or has incorrect syntax, updates to the list of users held by the probe do not occur.
- If the non-persistent configuration file is present, but has no content, the probe deletes all user/engine credentials that it holds.

Manually updating the non-persistent configuration file

You can manually load a new `mttrapd.conf` file into the probe, in either a stand-alone or failover pair configuration.

Note : To be able to manually load the configuration file, ensure that the **UsmUserBase** property has the value `0` or 2.

Stand-alone probe

To add users to `mttrapd.conf` use the following procedure:

1. Stop the probe.
2. Add the new users to the configuration file as shown in step 2 of "Using the automatic detection facility" on page 13.
3. Start the probe again.

Failover pair

A failover pair consists of a master and a slave probe. The master probe is the one that is set up to process traps and informs from the SNMP endpoint while the slave lies idle. If the master probe fails for any reason, the slave probe takes over processing of traps and informs. When the master probe becomes available once again, it resumes processing of traps and informs, taking control back from the slave probe. This configuration ensures continuity of processing.

Unlike a stand-alone probe you can update the list of users for a failover pair without interrupting the processing of traps and informs from the SNMP endpoint. You update each probe in turn, leaving the other probe in the pair running to process traps and informs. The procedure has two parts:

1. Set up the failover pair to use separate configuration file directories that contain identical copies of the `mttrapd.conf` file.
2. Add users to the configuration by updating the slave probe and then the master probe.

Set up the failover pair

Set up the failover the pair as follows:

1. Create separate properties files for the master and slave probes. Set up each file with the same values for all properties except for the following:

   `ConfPath`
   `PersistentDir`

   Ensure that for each probe these properties reference separate directories. For example:

   Master probe:

   ```
   ConfPath: $NCHOME/omnibus/var/mttrapd/master/snmpv3
   PersistentDir: $NCHOME/omnibus/var/mttrapd/master
   ```

   Slave probe:

   ```
   ConfPath: $NCHOME/omnibus/var/mttrapd/slave/snmpv3
   PersistentDir: $NCHOME/omnibus/var/mttrapd/slave
   ```

   Note : In the remainder of this procedure, the directories that these properties map to are referred to as `ConfPath` and `PersistentDir`.
2. Create the four directories specified in the two pairs of **`ConfPath`** and **`PersistentDir`** properties.
3. Create a copy of the `mttrapd.conf` file in each of the `ConfPath` directories.
4. Start the master and slave probes.

Add users to the configuration

Add users to the configuration as follows:

Table 8. Adding users to the configuration file and updating the failover pair of probes

Step		**Action**
1.	Update the slave probe.	1. Ensure that the master probe is running and then stop the slave probe. 2. Delete `mttarpd.conf` from the slave probe's `PersistentDir` directory. 3. Add the new users to the slave probe's configuration file in the `ConfPath` directory as shown in step 2 of “Using the automatic detection facility” on page 13. 4. Start the slave probe.

Table 8. Adding users to the configuration file and updating the failover pair of probes (continued)		
Step		**Action**
2.	Update the master probe.	1. Ensure that the slave probe is running and then stop the master probe. 2. Delete `mttrapd.conf` from the master probe's `ConfPath` and `PersistentDir` directories. 3. Copy `mttrapd.conf` from the slave probe's `ConfPath` directory to the master probe's `ConfPath` directory. 4. Copy `mttrapd.conf` from the slave probe's `PersistentDir` directory to the master probe's `PersistentDir` directory. 5. Start the master probe.

Traps and informs

In SNMP V3 USM, the probe is the non-authoritative security engine for traps. The `engineId` argument is required for each SNMP trap source so that the trap can be authenticated.

The sender of the SNMP informs is the non-authoritative security engine for informs. If informs are used, there is no requirement for the user to specify the `engineId` of the inform sender.

Probe initialization

The probe has facilities for handling traps and SIGINT interrupts it receives while it is parsing the configuration file for SNMP V3 (`mttrapd.conf`).

"Defining the startup behavior" on page 11 shows how the probe processes user entries in the configuration files and loads them. While the probe is doing this, there is a possibility of it receiving traps or a SIGINT interrupt. This is especially likely to occur when the configuration file contains a large number of user entries and engine identifiers. So the probe has facilities for dealing with these circumstances.

SNMP traps

The probe has facilities for handling all versions of SNMP traps.

The probe can receive and handle SNMP V1 and V2 traps while it is parsing the configuration file. Since these do not require any user validation, there is no dependency on the non-persistent configuration file.

The probe can receive V3 traps and process them, as long as the related user credentials have been read from the configuration file at the time the trap arrives. If the probe has not read that part of the configuration file, the probe cannot process the trap. Hence it rejects the trap.

SIGINT interrupt

The probe can receive a SIGINT interrupt at any time. If it receives one while parsing the configuration file, the probe stops processing the file and exits after writing out the set of users it has so far processed to the persistent configuration file. This means that the list of users may be incomplete or inconsistent depending on when the interrupt was received during the parsing process.

SNMP configuration validation

During startup, the `mttrapd.conf` files residing in **`ConfPath`** and **`PersistentDir`** are subject to the configuration validation process, which triggers the probe to exit upon any of the following scenarios:

1. The probe cannot find an `mttrapd.conf` file in **`ConfPath`**.

   Note : An `mttrapd.conf` file need not be present in **`PersistentDir`** during startup.
2. The probe cannot read the `mttrapd.conf` file.

3. The probe detects a syntax error in known directives, for example: **createUser**.
4. The probe reads an unknown directive.

   Note : The first word read from a line is taken to be a directive.

Mismatch between the probe configuration and deployment

Because the decryption test is not part of the configuration validation process, during configuration validation the probe's take on `mttrapd.conf` as a plaintext or an encrypted artifact is as per the assumption established by the probe's properties. This means that a mismatch between the probe configuration and deployment can occur, for example:

With **`EnableCryptoConfig`** set to `0` while `mttrapd.conf` is encrypted, the probe would read in the encrypted content without performing decryption, thus causing the raw content to not be processable by the configuration validation process.This would result in an example of **Scenario 4**, the probe reading an unknown directive.

Limiting processing to SNMP V3 only

You can configure the probe to process SNMP v3 traps and informs only.

Setting the value of the **`snmpv3ONLY`** to 1 limits the probe to processing only SNMP V3 traps and informs.

Minimum security level for trap processing

You can define which traps and informs the probe processes based on their security credentials.

In SNMP V3 each trap and inform is associated with a user (endpoint) and users are defined in the configuration files (see "Configuration files" on page 11). Each user has one of three security levels determined by their definition in the configuration files:

Table 9. Security levels for SNMP V3 users for traps and informs

Security level	Description
NoAuth	There is no authentication type and no privacy type defined for the user.
AuthNoPriv	The user has an authentication type but no privacy type.
AuthPriv	The user has both an authentication type and a privacy type.

By default, the probe processes all traps and informs regardless of the security level of their users. However, you can use the **`snmpv3MinSecurityLevel`** property to define which traps and informs the probe processes based on the security level of their users. The property can take the following values:

Table 10. Values of snmpv3MinSecurityLevel

Value of snmpv3Min Security Level	Traps and informs that the probe processes
1	The probe processes traps and informs associated with users with the following security levels: NoAuth AuthNoPriv AuthPriv

Table 10. Values of snmpv3MinSecurityLevel (continued)	
Value of snmpv3Min Security Level	**Traps and informs that the probe processes**
2	The probe processes traps and informs associated with users with the following security levels: AuthNoPriv AuthPriv
3	The probe processes traps and informs associated with users with the AuthPriv security level only.

Example configuration

The following examples show how to configure the SNMP Probe to receive traps and informs. In addition to the probe configuration, the examples show how to use the Net-SNMP command-line tool to send the trap or inform to the probe.

The examples use the default protocol of UDP.

Configuring traps

This example shows how to configure the probe to receive the `coldStart` trap. This trap occurs when the monitored device starts.

The following table shows the information in the `mttrapd.conf` file required to configure the SNMP Probe for a trap.

Table 11. Information required to configure traps	
Data item	**Example value**
SNMP user	
User name	`netcoolTrap`
Authorization encryption	
Method	`MD5`
Password	`tr4psMD5`
Privacy encryption	
Method	`DES`
Password	`trp4psDES`
Authorization engine	
Engine identifier	`0x0102030405`
SNMP Probe information	

Table 11. Information required to configure traps (continued)	
Data item	**Example value**
Server name	snapper
Port	1162

To configure the probe, edit the `mttrapd.conf` file and add the following `createUser` entry:

```
createUser -e 0x0102030405 netcoolTrap MD5 tr4psMD5 DES tr4psDES
```

To send the `coldStart` trap using the Net-SNMP command-line tool, add the following entry to `mttrapd.conf`:

```
snmptrap -e 0x0102030405 -v3 -u netcoolTrap -a MD5 -A tr4psMD5 -x DES -X tr4psDES
-l authPriv snapper:1162 "" coldStart.0
```

Configuring informs

Table 12. Information required to configure informs	
Data item	**Example value**
SNMP user	
User name	netcoolInform
Authorization encryption	
Method	MD5
Password	1nformsMD5
Privacy encryption	
Method	DES
Password	1nformsDES
SNMP Probe information	
Server name	snapper
Port	1162

To configure the probe edit the `mttrapd.conf` file and add the following `createUser` entry:

```
createUser netcoolInform MD5 1nformsMD5 DES 1nformsDES
```

To send an inform using the Net-SNMP command-line tool, add the following entry to `mttrapd.conf`:

```
snmptrap -Ci -v3 -u netcoolInform -a MD5 -A 1nformsMD5 -x DES -X 1nformsDES
-l authPriv snapper:1162 "" coldStart.0
```

Values of the ConfPath and PersistentDir properties

The **ConfPath** and **PersistentDir** properties define the locations of the `mttrapd.conf` file and the hashed file. For example:

```
ConfPath            : '$NCHOME/omnibus/var/master/conf'
PersistentDir    : '$NCHOME/omnibus/var/master'
```

Data acquisition

Each probe uses a different method to acquire data. Which method the probe uses depends on the target system from which it receives data.

The SNMP Probe is a direct SNMP monitoring probe. The probe acquires event data by acting as a trap daemon and monitoring SNMP traps and events on both UDP and TCP sockets.

Data acquisition is described in the following topics:

- "ObjectServer information" on page 20
- "Communications information" on page 21
- "Paths used by the probe" on page 23
- "Queue and buffer settings" on page 23
- "Generic trap handling" on page 25
- "Logging lost traps" on page 26
- "Monitoring flood conditions at IP addresses" on page 25
- "Support for Unicode and non-Unicode characters" on page 26
- "Rules files" on page 28
- "Peer-to-peer failover functionality" on page 29

ObjectServer information

Define the ObjectServer or pair of ObjectServers that the probe communicates with.

The probe sends processed events to an ObjectServer resource. This can be a single server or a failover pair of ObjectServers.

Single ObjectServer

Always configure the probe to communicate with an ObjectServer by setting the **Server** property to the name of the ObjectServer.

Failover pair

Optionally, you can define a failover pair of ObjectServers. One acts as the primary Objectserver that the probe communicates with initially. Should that ObjectServer become unreachable, the probe switches to using the other, backup ObjectServer. Use the following procedure to define a failover pair:

1. Set the **Server** property to the name of the primary ObjectServer.
2. Set the **ServerBackup** property to the name of the backup ObjectServer.
3. Set the **NetworkTimeout** property to the timeout period, in seconds, for the primary ObjectServer. If the probe does not receive a response from the ObjectServer within that time, it connects to the backup ObjectServer.
4. Set the **PollServer** property to the time period, in seconds, when the probe tries to reconnect with the primary ObjectServer.

   Ensure that the value of **NetworkTimeout** is less than the value of **PollServer**.

Communications information

Use this information to configure the probe to communicate with the SNMP device. Define the communications port the probe uses to listen for SNMP traps and informs, the protocol it uses to communicate with the SNMP device, and whether IP address resolution is used.

Communications port

Define the port that the probe uses to listen for SNMP traps and informs.

Use the **Port** property to define the communications port that the probe is to use when listening for SNMP traps and informs. The default port number is 162. Use the following guidelines to determine the value of this property:

- If the probe is to run from a non-root account, do not use the default port number of 162.
- Ensure that no other SNMP agent is listening on the port you want to use.

IP environment

The probe can run in either IPv4 or IPv6 environments. You specify the environment using the **Protocol** property. When the probe is running in an IPv4 environment, set the **Protocol** property to TCP, UDP, ALL or ANY. When running in an IPv6 environment, set the **Protocol** property to TCP6, UDP6, or ALLIPV6.

Ensure that the IP address you specify in the **BindAddress** property is in a format that matches the protocol specified by the **Protocol** property.

Table 13. Setting the BindAddress Property

Value of Protocol property	Value of BindAddress property
TCP, UDP, ALL, or ANY	An address in IPv4 format. That is, an address in 32-bit decimal notation with four decimal numbers, separated by periods. For example: `121.2.6.81`
TCP6, UPD6, or ALLIPV6	An address in IPv6 format. That is an address in 128-bit hexadecimal notation with eight hexadecimal fields, separated by colons. For example: `3ffe:ff9f:101::230:6eff:fe04:d9ff`

IP address resolution

If the Domain Name Server (DNS) is not resolving IP addresses quickly enough, you can improve performance by ensuring that the value of the **NoNameResolution** property is 1 (this is the default value). That value prevents the probe from performing name resolution.

To resolve IP addresses on Windows operating systems, set the **NoNetbiosLookups** property to 0.

Writing resolved host names and discarded host names to flat files

If you set the **NoNameResolution** property to 0, the probe attempts to resolve the IP address for every trap that the probe receives.

Holding in memory details of all host names resolved for IP sources

The probe can hold in memory details of all the host names resolved for the IP source of every trap it receives. This reduces the network overhead that is otherwise incurred due to reported DNS queries. To specify the maximum size of the table in which the probe stores IP-host name pairs, use the

HostnameTableSize property. If the table exceeds this size, the probe writes an error message to the log.

You can specify how long an ip-host name value pair stays in the table using the **ActiveHostnameDuration** property. When this length of time has elapsed since the probe last visited the element, it will be removed from the table held in memory and will be written instead to the flat file for discarded IP-host name pairs.

Processing alarms when the IP-host name table has reached its maximum size

The probe will stop parsing the active list at the list entry where it detects that the table is full.

The probe will not perform an instantaneous host name resolution for any traps received from new IP hosts and the IP will not be stored in the table. The relevant event tokens will carry numeric IP addresses.

Reading from the IP host flat file

When the probe starts, it reads the following flat file:

`$OMNIHOME/var/`*`Instance`*`_active_iphost.list`

Where *Instance* identifies the current running instance of the probe.

The *Instance*`_active_iphost.list` file contains a list of IP addresses and their corresponding host names. The probe uploads the IP address-host name pairs from the list into an internal table that it holds in memory. The probe ignores an IP address-host name pair if any of the following conditions apply:

- The IP address is incorrectly formatted.
- The IP address is not consistent with the setting of the **Protocol** property; for example, if **Protocol** is set to IPv4 the probe will ignore IPv6 addresses.

If a resolved host name exceeds 255 characters, the probe will truncate it, writing just the first 255 characters to the internal table.

Note : The probe does not query the DNS server to check whether the host name actually corresponds to the IP address. When you start the probe for the first time, you must ensure that the *Instance*`_active_iphost.list` file contains valid IP address-host name pairs and that all host names that you include are consistent with the DNS server, or are valid host names if they have not yet been updated in the DNS server. If no host name can yet be determined for a given IP, you can specify just the IP address for that host. The advantage of supplying IP addresses without host names is that the probe will make DNS queries for just those host names.

Specifying how frequently the probe performs a DNS query

To confirm the validity of the host name values associated with each IP address, the probe periodically queries the DNS server.

You can use the **RefreshHostnameInterval** property to specify the frequency (in minutes) with which the probe queries the DNS server and updates the host names for the IP addresses stored in memory.

If an IP node held in memory does not have a corresponding host name, and if during the next DNS query the probe manages to retrieve a host name from the DNS server, it will update the node with the resolved host name. If the DNS query fails for an IP node (whether or not it currently has a corresponding host name held in memory), the host name field for the node will remain as it is until the next DNS query attempt.

Hostname resolution policy during trap processing

If the trap's IP address exists in the probe's internal table and it has a host name set for it, the probe will assign that host name to the event token.

If a trap's IP is new to the internal table, the probe will send a DNS query for the IP's hostname. It will then add the IP-host name pair into the internal table as long the table has not reached the limit specified by the **HostnameTableSize** property. When the internal table is full, the probe makes no DNS query for the IP.

If the IP's host name is empty (either because of an unsuccessful DNS query, or because the original host name value was not set), the probe will assign the IP address to both the Node token and the PeerAddress token.

Note : The IP address values that are assigned to Node and PeerAddress come from IPaddress event token and PeerIPaddress event token respectively. This value may or may not be the same IP address.

Processing idle IP nodes

When an IP node in the probe's internal table has been idle for a period longer than that specified by the **ActiveHostnameDuration**, the probe writes the value of the IP address-host name pair to the following flat file:

$OMNIHOME/var/*Instance*_discarded_iphost.list

For example, if **ActiveHostnameDuration** is set to 20 minutes and an IP node was created at 1:08 PM, and if the node has not been referenced by the probe to get a resolved host name value to assign to a trap event token by 1:28 PM, then the probe deletes the node from the table and writes the IP address-host name value to the discarded list file. However, if the node was referenced by probe at 1:18 PM, its expiry time is reset to 1:38 pm.

If the probe receives a trap from an IP node that appears in the discarded list, the probe recreates the node in its internal table.

Writing details of the active IP nodes to the flat file on exit

When the probe exits, it writes the details held in its internal table to the following flat file:

$OMNIHOME/var/*Instance*_active_iphost.list

The probe writes one IP address-host name pair per line using the following format for each line:

ip_address hostname

Where *ip_address* is an IP address that the probe has attempted to resolve and *hostname* is the name of the host resolved for that IP address.

Note : If the host name for any of the IP addresses was unresolved, *hostname* will be absent.

Paths used by the probe

The probe uses two directories: the **ConfPath** property points to one and the **PersistentDir** property points to the other. The probe reads the **ConfPath** property to locate the mttrapd.conf file and writes the processed file to the directory specified by the **PersistentDir** directory. In SNMPv3, the resultant file contains the hashed key. "SNMP V3 support" on page 10 contains more information on the mttrapd.conf configuration files and how the probe processes them.

Queue and buffer settings

The probe uses a queue to store traps before processing them. Once processed, the probe puts the resulting alerts in a buffer ready to send to the ObjectServer. Use the information in this topic to set appropriate values for the properties that define the characteristics of the trap queue and the output buffer.

Trap queue

The trap queue holds SNMP traps that are waiting for processing by the probe. When an event storm occurs, this queue can grow quickly and consume excessive amounts of memory. To prevent this, you can

use the **TrapQueueMax** property to specify a maximum size to which the queue can grow before the probe starts to discard traps.

It is important that you set **TrapQueueMax** to a value appropriate for your environment, Using a value that is too low causes traps to be lost. Using a value that is too large, consumes memory and that can effect the efficiency of the probe.

Determining the appropriate value for your environment may be an iterative process. You can use the probe's facilities for logging performance data to monitor the size of the queue over a period of time. From that you can determine a value for the size of the queue in normal operation and add in extra capacity to allow for extraordinary circumstances, without adversely affecting the efficiency of the probe. See "Logging performance data" on page 24 for information on logging performance data.

Note : The **TrapQueueMax** property is set to `20000` by default. If the value is set to `0`, the probe generates the following warning message: `Memory growth of the probe is unbounded.`

Buffer settings

Use the following properties to help improve the efficiency of sending alerts to the ObjectServer:

- **Buffering** - When set to 1, this property instructs the probe to send alerts when the internal alert buffer has reached the size specified by the **BufferSize** property.
- **BufferSize** - This property specifies the size of the buffer that the probe uses to store alerts before sending them to the ObjectServer.
- **FlushBufferInterval** - This property specifies an interval in seconds that the probe waits before flushing the alerts to the ObjectServer. This property limits the time that alerts wait in the buffer when the buffer has yet to reach the size specified by the **BufferSize** property.

Example

The following example shows performance settings from the properties file of an SNMP Probe:

```
TrapQueueMax        : 1000
BufferSize          : 100
Buffering           : 1
FlushBufferInterval : 10
```

These settings instruct the probe to store a maximum of 1000 raw traps prior to converting them to ObjectServer alerts. When the internal alert buffer has 100 alerts waiting to be sent to the ObjectServer, or after 10 seconds have elapsed since the last flush, the probe flushes the alerts in the buffer to the ObjectServer.

Logging performance data

You can configure the probe to write performance data to the log file. You can use this data to help fine tune the probe.

Use the **LogStatisticsInterval** property to define how often the probe writes messages that contain performance data to the log file. When the property has a value of `0` (the default value), the probe does not log performance data. Any positive value defines the interval in seconds that occurs between each set of messages containing performance data.

The statistics that the probe records in the log file include the following:

- The size of the trap queue
- The size of the inform queue
- The number of traps read
- The number of traps processed

You can obtain additional performance data by setting the **MessageLevel** property to Debug. However, revert to the original setting of this property as soon as the additional data is no longer needed since using the Debug setting greatly increases the size of the log files.

Monitoring flood conditions at IP addresses

The probe supports trap flood monitoring functionality which allows the probe to identify potential event flooding conditions at individual IP addresses.

If enabled, the following flood monitoring processing occurs:

1. When the probe starts, it collects the trap count, drop count, and size of the trap queue for each IP address.
2. The probe uses these trap statistics to prevent trap floods filling up the queue, (which could otherwise cause traps from all IPs to be dropped).
3. When an IP sends an excessive number of traps to the queue, the probe detects that the IP's trap flow exceeds the pre-configured threshold value, and adds the IP address to its drop list. Traps from this IP will then be blocked from being added to the probe's internal queue, or will be discarded after retrieval from the queue.
4. When an IP address has been blocked, the probe periodically checks the number of traps that it is receiving. If the number of traps received has not slowed, the probe continues to block the IP address. If the number of traps received has slowed, the probe unblocks the IP address.

To enable the probe to use trap flood monitoring functionality, you must set the **`TrapStat`** property to 1.

Generic trap handling

Certain devices generate traps of various generic types. How the probe handles each trap depends on its type.

The following table describes the handling of each generic trap type.

Table 14. Generic trap handling

Generic trap	**Handling**

Generic trap-type 0- Cold Start	`Summary` field set to `Cold Start`
	`AlertGroup` field set to `Generic`
	`Severity` field set to 4

Generic trap-type 1- Warm Start	`Summary` field set to `Warm Start`
	`AlertGroup` field set to `Generic`
	`Severity` field set to 4

Generic trap-type 2 - Link Down	`Summary` field set to `Link Down`
	`Alert Key` set to the $1 varbind (`ifIndex`)
	`AlertGroup` field set to `Generic`
	`Severity` field set to 5
	`Identifier` field set to `Node name` plus `Agent` plus `generic trap` plus `specific trap` plus `ifIndex.`

Table 14. Generic trap handling (continued)	
Generic trap	**Handling**

Generic trap-type 3 - Link Up	`Summary` field set to `Link Up`
	`Alert Key` set to the `$1` varbind (`ifIndex`)
	`AlertGroup` field set to `Generic`
	`Severity` field set to 2
	`Identifier` field set to `Node name` plus `Agent` plus `generic trap` plus `specific trap` plus `ifIndex.`

Generic trap-type 4 - Authentication	By default, Authentication traps are not discarded.

Generic trap-type 5 - EGP Neighbor Loss	`Summary` field set to `EGP Neighbor Loss`
	`AlertGroup` field set to `Generic`
	`Severity` field set to 3.

Logging lost traps

The probe can create log messages for events that are lost because the trap queue is full.

If the traps queue becomes full, the probe discards any further traps that it receives until there is space in the queue. You can use the **DSALog** and **DSAPeriod** properties to direct the probe to create log messages that record which traps are lost. "Properties and command line options" on page 39 contains descriptions of these properties and their values.

Heartbeat

The probe can disconnect from the target system if the connection between them becomes unavailable.

You can use the **Heartbeat** property to specify whether the probe periodically checks that the connection to the target system is available and how often it performs that check. The probe shuts down if it detects that the connection to the target system is unavailable.

When the **Heartbeat** property has a value of `0` the probe does not check the availability of the connection. Any other positive value defines the number of seconds between each check of the connection's availability.

Support for Unicode and non-Unicode characters

The probe can process multibyte characters and so can display both Unicode and non-Unicode characters.

Note : If you do not configure the probe to process multi-byte characters, it discards any traps it receives that contain such characters.

Unicode characters

Use the following procedure to set up the probe to process characters in Unicode UTF-8 format:

1. Ensure that the SNMP endpoint is configured to send traps in UTF-8 format.
2. Set the required locale on the system that runs the probe:

Table 15. Setting the locale for Unicode UTF-8 characters	
Operating system	**Procedure to set the locale**

Linux and Unix	Set the locale by changing the values of the **LANG** and **LC_ALL** environment variables. For example, to set the locale to simplified Chinese, use the following commands:
	`export LANG=zh_CN.UTF-8`
	`export LC_ALL=zh_CN.UTF-8`

Windows	a. Open the Control Panel and double click on **Regional and Language**. b. On the **Formats** tab, select the language from the list in **Format**. c. On the **Administrative** tab, click **Change system locale**. d. Select the language from the list in **Current System Locale**. e. Click **OK**. f. Click **OK**.

3. Configure the ObjectServer to enable the insertion of data that uses UTF-8 format. The *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide* shows how to create, configure, and run an ObjectServer in UTF-8 mode.
4. Start or, if it is already running, restart the probe.

   When running the probe on a Windows system, always specify the `-utf8enabled` command line option.

Other multi-byte character sets

In addition to characters encoded in Unicode UTF-8 format, the probe can process other multi-byte character sets such as GB, Big5 or Shift-JIS. Use the following procedure to process characters in such character sets:

1. Ensure that the SNMP endpoint is configured to send traps in the required format.
2. Set the locale on system that runs the probe:

Table 16. Setting the locale for other multi-byte character sets	
Operating system	**Procedure to set the locale**
Linux and Unix	Set the locale by setting the **LANG** environment variable to C environment variables. For example: `export LANG=C`
Windows	Follow the instructions in Table 15 on page 27

3. Configure the ObjectServer to process the required character set. The *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide* shows how to create, configure, and run an ObjectServer.
4. Start or, if it is already running, restart the probe.

   When running the probe on a Windows system, **do not** use the `-utf8enabled` command line option.

Rules files

The Netcool/Knowledge Library contains a collection of fully tested rules files for SNMP and Syslog enabled devices. It can be downloaded from the Passport Advantage Web Site.

This library can be used as a basis for creating rules files for use with the SNMP EMS Probe. For details, contact IBM Software Support.

Note : Avoid using construct details ($*) in the rules file.

Rules file processing

In addition to the rules file functions provided by the Common Probe Library (`libOpl`), the IBM Tivoli Netcool/OMNIbus SNMP Probe also supports SNMP SET and GET operations.

For details of standard rules file functions provided by the `libOpl`, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

This section describes the following operations:

- "SNMP GET" on page 28
- "SNMP SET" on page 28

SNMP GET

The SNMP GET functions allow the rules file to retrieve SNMP variables from an SNMP agent.

There are two functions for each version of SNMP; one without a _p suffix and one with the suffix. The form without the suffix (for example, `snmpget`) uses the default port number `161`. The form with the suffix (for example, `snmpget_p`) uses the port number on the SNMP agent specified by the **`Port`** property.

When using the form without the suffix in the rules file, you must include the port number with the IP address of the host, preceded by a colon.

The syntax of the functions is different for each version of SNMP.

SNMP V1

```
$value = snmpget($hostname, $OID, $community)
$value = snmpget_p($hostname, $OID, $community, $port)
```

SNMP V2c

```
$value = snmpget_v2c($hostname, $OID, $community)
$value = snmpget_v2c_p($hostname, $OID, $community, $port)
```

SNMP V3

```
$value = snmpget_v3($hostname, $userid, $OID)
$value = snmpget_v3_p($hostname, $userid, $OID, $port)
```

SNMP SET

The SNMP SET functions enable the rules file to set SNMP variables in an SNMP agent.

There are two functions for each version of SNMP; one without a _p suffix and one with the suffix. The form without the suffix (for example, `snmpset`) uses the default port number `161`. The form with the suffix (for example, `snmpset_p`) uses the port number on the SNMP agent specified by the **`Port`** property.

Both functions return a value indicating whether the variable was set: `0` means the variable was not set and `1` means the variable was set.

When using the form without the suffix in the rules file, you must include the port number with the IP address of the host, preceded by a colon. Due to a limitation in the rules file processor, the value returned by the form that uses the default port must be used even if it is subsequently ignored.

The syntax of the functions is different for each version of SNMP.

SNMP V1

```
$success = snmpset($hostname, $OID, $type, $value, $community)
$success = snmpset_p($hostname, $OID, $type, $value, $community, $port)
```

SNMP V2

```
$success = snmpset_v2c($hostname, $OID, $type, $value, $community)
$success = snmpset_v2c_p($hostname, $OID, $type, $value, $community, $port)
```

SNMP V3

```
$success = snmpset_v3($hostname, $userid, $OID, $type, $value)
$success = snmpset_v3_p($hostname, $userid, $OID, $type, $value, $port)
```

Object types

In the functions, *type* has one of the following values:

Type	Description
a	IP address
b	Bit
d	Decimal string
i	Integer
n	Null object
o	Object identifier
s	String
t	Timetick
u	Unsigned
x	Hex string

Peer-to-peer failover functionality

The probe supports failover configurations where two probes run simultaneously. One probe acts as the `master` probe, sending events to the ObjectServer; the other acts as the `slave` probe on standby. If the master probe fails, the slave probe activates.

While the slave probe receives heartbeats from the master probe, it does not forward events to the ObjectServer. If the master probe shuts down, the slave probe stops receiving heartbeats from the master and any events it receives thereafter are forwarded to the ObjectServer on behalf of the master probe.

When the master probe is running again, the slave probe continues to receive events, but no longer sends them to the ObjectServer.

Example property file settings for peer-to-peer failover

You set the peer-to-peer failover mode in the properties files of the master and slave probes. The settings differ for a master probe and slave probe.

Note : In the examples, make sure to use the full path for the property value. In other words replace `$OMNIHOME` with the full path. For example: `/opt/IBM/tivoli/netcool`.

The following example shows the peer-to-peer settings from the properties file of a master probe:

```
Server      :    "NCOMS"
RulesFile   :    "master_rules_file"
MessageLog  :    "master_log_file"
PeerHost    :    "slave_hostname"
PeerPort    :    6789 # [communication port between master and slave probe]
Mode        :    "master"
PidFile     : "master_pid_file"
```

The following example shows the peer-to-peer settings from the properties file of the corresponding slave probe:

```
Server      :    "NCOMS"
RulesFile   :    "slave_rules_file"
MessageLog  :    "slave_log_file"
PeerHost    :    "master_hostname"
PeerPort    :    6789 # [communication port between master and slave probe]
Mode        :    "slave"
PidFile     : "slave_pid_file"
```

Managing the SNMP agent over the probe's HTTP/HTTPS interface

IBM Tivoli Netcool/OMINbus V7.4.0 includes capabilities for managing the SNMP agent remotely over the probe's HTTP/HTTPS interface using the **nco_http** and **nco_probeventfactory** utilities.

For example, you can use this facility to monitor the connection between the probe and the SNMP agent by periodically issuing an SNMP GET command over the HTTP/HTTPS interface.

The following sections:

- Summarize how to enable remote management of the agent.
- Show the format of the commands that you send to the probe using **nco_http**, with examples.
- Show the format of commands that you send the probe using **nco_probeventfactory** with examples.
- Show how you can use a properties file to hold frequently-used commands.
- Set out the changes required to the probe's rules file for remote management.

See the chapter on remotely administering probes in the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* for detailed information on managing probes remotely, including the **nco_http** and **nco_probeeventfactory** utilities.

Enabling remote management of the probe

To enable the probe to receive commands over HTTP/HTTPS, set the following properties in the property file `mttrapd.props`:

MessageLevel: Set this property to the required level for message logging.
MessageLog: Set this property to the location of the message log.
NHttpd.AccessLog: Set this property to the path of the access log that the probe creates.
NHttpd.EnableHTTP: Set this property to TRUE.

NHttp.ExpireTimeout: Set this property to the timeout period, in seconds.
NHttp.ListeningPort: Set this property to the port that the probe uses to listen for HTTP commands.
RulesFile: Set this property to the full path name of the probe's rules file.

Always provide values for **NHttp.EnableHTTP** and **NHttp.ListeningPort**. Provide values for the remaining properties as required. On Windows systems, use a quoted backslash (\\) to separate the elements of any path name.

For a full description of each of these properties see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

In addition, modify the rules file as shown in "Rules file" on page 38

The following shows an example set of properties for HTTP remote management for a Red Hat Linux system.

```
MessageLevel: 'debug'
MessageLog: '$OMNIHOME/log/mttrapd.log'
NHttpd.AccessLog: '$OMNIHOME/log/bidir_access.log'
NHttpd.EnableHTTP: TRUE
NHttpd.ExpireTimeout: 15
NHttpd.ListeningPort: 6789
RulesFile: '$OMNIHOME/probes/linux2x86/mttrapd.rules
```

The following shows an example set of properties for a Windows system:

```
MessageLevel: "debug"
MessageLog: "%OMNIHOME%\\log\\mttrapd.log"
NHttp.AccessLog: "%OMNIHOME%\\log\\bidir_access.log'
NHttpd.EnableHTTP: TRUE
NHttpd.ExpireTimeout: 15
NHttpd.ListeningPort: 6789
RulesFile: "C:\\IBM\\Tivoli\\omnibus\\probes\\win32\\mttrapd.rules"
```

Sending commands using nco_http

When using **nco_http**, commands are sent to the probe, in JSON format, using the HTTP POST method.

There are two groups of command:

- Get commands for retrieving data from the agent
- Set commands for setting data in the agent

The following sections show the format of each command in **nco_http** and include an example of each. The examples operate on the System group of objects in the standard SNMP MIB. This MIB resides in the internet management subtree of the SNMP object tree. In addition, there is information on messages that can appear in the log file to indicate success of the command or indicate a problem with the syntax of the command line.

The sections use the following conventions for the parameter values:

Table 17. Syntax conventions for nco_http parameters

Parameter	Meaning
probeuri	The URI of the probe.
probeport	The port that the probe listens on for HTTP connections.
destinationhost	The host name or IP address of the destination address for the command. That is, the address of the SNMP agent.
userid	The user account required to log in to the probe when using SNMP V3.

Table 17. Syntax conventions for nco_http parameters (continued)	
Parameter	**Meaning**
oid	The identifier of the object.
type	The type of the object. See "Object types" on page 29 for a list of values that this parameter can take.
value	The value to associate with the object.
community	The SNMP community string.
port	The target port on the SNMP agent that the probe uses for the command.

Get commands

Get commands obtain information from the probe.

There are two get commands for each version of SNMP. The **snmpget** form of the command uses the default port number to communicate with the probe, and the **snmpget_p** form of the command uses the port number specified in the command.

SNMP V1 and V2c

The get command for SNMP V1 and V2c that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "getcommand",
"hostname":"destinationhost", "oid":"oid", "community":"community"}]}' -method
post
```

Where *getcommand* takes one of the following values, depending on your version of SNMP:

SNMPGET
SNMPGET_V2C

The command that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "getcommand",
"hostname":"destinationhost", "oid":"oid", "community":"community",
"port":"port"}]}' -method post
```

Where *getcommand* takes one of the following values, depending on your version of SNMP:

SNMPGET_P
SNMPGET_V2C_P

SNMP V3

The get command for SNMP V3 that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "SNMPGET_V3",
"hostname":"destinationhost", "userid": "userid", "oid":"oid"}]}' -method post
```

The command that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "SNMPGET_V3_P",
```

```
"hostname":"destinationhost", "userid": "userid", "oid":"oid", "port":port}]}'
-method post
```

Examples

The following example retrieves the value of the **sysDescr** object in the SNMP agent. The command uses the default port and SNMP V1:

```
$OMNIHOME/bin/nco_http -uri http://test28.example.com:6789/probe/common -
datatype application/json -data '{"eventfactory": [{"snmpreq": "SNMPGET",
"hostname":"9.127.128.299", "oid":".1.3.6.1.2.1.1.0", "community":"private"}]}'
-method post
```

The following example also retrieves the value of the **sysDescr** object, but uses SNMP V3. The command specifies the user as **snmpadmin** and uses port 4990 on the agent:

```
$OMNIHOME/bin/nco_http -uri http://test28.example.com:6789/probe/common -
datatype application/json -data '{"eventfactory": [{"snmpreq": "SNMPGET_V3_P",
"hostname":"9.127.128.299", "userid": "snmpadmin", "oid":".1.3.6.1.2.1.1.0",
"port":"4990"}]}' -method post
```

Set commands

Set commands set the value of data items in the SNMP agent.

There are two set commands for each version of SNMP. The **snmpset** form of the command uses the default port number to communicate with the probe, and the **snmpset_p** form of the command uses the port number specified in the command.

SNMP V1 and V2c

The set command for SNMP V1 and V2c that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "setcommand",
"hostname":"destinationhost", "oid":"oid", "type":"type", "value": "value",
"community":"community"}]}' -method post
```

Where *setcommand* takes one of the following values, depending on your version of SNMP:

SNMPSET
SNMPSET_V2C

The command that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "setcommand",
"hostname":"destinationhost", "oid":"oid", "type":"type", "value": "value",
"community":"community", "port":"port"}]}' -method post
```

Where *setcommand* takes one of the following values, depending on your version of SNMP:

SNMPSET_P
SNMPSET_V2C_P

SNMP V3

The set command for SNMP V3 that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "SNMPSET_V3",
"hostname":"destinationhost", "userid":"userid", "oid":"oid", "type":"type",
"value": "value"}]} -method post
```

The command that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_http -uri probeuri:probeport/probe/common -datatype
application/json -data '{"eventfactory": [{"snmpreq": "SNMPSET_V3_P",
"hostname":"destinationhost", "userid": "userid", "oid":"oid", "type":"type",
"value": "value", "port":"port"}]}' -method post
```

Examples

The following example sets the value of the **sysLocation** object in the SNMP agent to `First floor computer room`. The command uses port 4990 on the agent and SNMP V2c:

```
$OMNIHOME/bin/nco_http -uri http://test28.example.com:6789/probe/common -
datatype application/json -data '{"eventfactory": [{"snmpreq": "SNMPSET_V2C_P",
"hostname":"9.127.128.299", "oid":".1.3.6.1.2.1.1.6.0", "type":"s", "value":
"First floor computer room", "community":"private", "port":"4990"}]}' -method
post
```

The following example also sets the value of the **sysLocation** object using SNMP V3. The command uses the default port on the agent and a user name of **snmpadmin**:

```
$OMNIHOME/bin/nco_http -uri http://test28.example.com:6789/probe/common -
datatype application/json -data '{"eventfactory": [{"snmpreq": "SNMPSET_V3",
"hostname":"9.127.128.299", "userid":"snmpadmin", "oid":".1.3.6.1.2.1.1.6.0",
"type":"s", "value": "First floor computer room"}]}' -method post
```

Messages in the error log

Messages in the error log can help isolate syntax problems with get and set commands.

The **nco_http** utility makes extensive entries in the log file (`mttrapd.log`) indicating the progress of each operation.

For example, here are a sample set of messages for a get command:

```
2012-11-15T13:58:06: Debug: D-NHT-105-002: [HTTP Listener]: Connection from
'test28.example.com' (9.180.210.223) on socket '16'.
2012-11-15T13:58:06: Debug: D-NHT-105-008: [HTTP Listener]: Data available
 from 'test28.example.com' (9.180.210.223) on socket '16'.
2012-11-15T13:58:06: Debug: D-NHT-105-009: [WorkPoolThread]: Received HTTP
 request from 'test28.example.com' (9.180.210.223) on socket '16'.
2012-11-15T13:58:06: Information: I-UNK-000-000: Property NHttpd.BasicAuth
is empty, allowing connection with no authentication
2012-11-15T13:58:06: Information: I-UNK-000-000: HTTP: POST: /probe/common
request from auth=none host=[9.180.210.223]
2012-11-15T13:58:06: Debug: D-UNK-000-000: Received SNMPGET request
2012-11-15T13:58:06: Debug: D-UNK-000-000: Calling snmpget
2012-11-15T13:58:06: Debug: D-UNK-000-000: Return value : Router 1
2012-11-15T13:58:06: Debug: D-UNK-000-000: Rules file processing took
2323 usec.
2012-11-15T13:58:06: Debug: D-UNK-000-000: Flushing events to object servers
2012-11-15T13:58:07: Debug: D-NHT-105-001: [HTTP Listener]:
Disconnection from 'test28.example.com' (9.180.210.223) on socket '16'.
```

For SNMPGET and SNMPGET_P operations, some of these messages can indicate that there are problems with the syntax of the **nco_http** command. When a command completes successfully, the log file includes messages similar to the following example:

```
2012-11-15T13:58:06: Debug: D-UNK-000-000: Received SNMPGET request
2012-11-15T13:58:06: Debug: D-UNK-000-000: Calling snmpget
2012-11-15T13:58:06: Debug: D-UNK-000-000: Return value : Router 1
2012-11-15T13:58:06: Debug: D-UNK-000-000: Rules file processing took 2323 usec.
```

Note that the returned value is filled in with the data retrieved from the target.

If there is an error in the command syntax, however, that segment of messages is similar to the following example:

```
2012-11-15T15:28:52: Debug: D-UNK-000-000: Received SNMPGET_P request
2012-11-15T15:28:52: Debug: D-UNK-000-000: Calling snmpget_p
2012-11-15T15:28:52: Error: E-P_M-102-000: snmpget: (noSuchName) There is no such
variable name in this MIB.
2012-11-15T15:28:52: Debug: D-UNK-000-000: Return value :
2012-11-15T15:28:52: Debug: D-UNK-000-000: Rules file processing took 1677 usec.
```

In this case, the returned value is blank and the message immediately before that indicates the type of error. In this example, the value of the `-oid` option is incorrect.

Sending commands using nco_probeeventfactory

Commands are sent to the probe using the **nco_probeeventfactory** utility. In turn, this uses the **nco_http** utility to despatch the data to the probe.

There are two groups of command:

- Get commands that retrieve data from the agent
- Set commands that set data in the agent

The following sections show the format of each command in **nco_probeeventfactory** and include an example of each. The examples operate on the System group of objects in the standard SNMP MIB. This MIB resides in the internet management subtree of the SNMP object tree.

The sections use the following conventions for the parameter values:

Table 18. Syntax conventions for nco_probeeventfactory parameters

Parameter	Meaning
probehost	The host name or IP address of the system that runs the probe.
probeport	The port that the probe listens on for HTTP connections.
destinationhost	The host name or IP address of the destination address for the command. That is, the address of the SNMP agent.
userid	The user account required to log in to the probe when using SNMP V3.
oid	The identifier of the object.
type	The type of the object. See "Object types" on page 29 for a list of values that this parameter can take.
value	The value to associate with the object. Enclose the value in quotes when it includes spaces.
community	The SNMP community string.
port	The target port on the SNMP agent that the probe uses for the command.

Get operations

Get operations obtain information from the probe.

There are two get operations for each version of SNMP. The **snmpget** form of the command uses the default port number to communicate with the probe, and the **snmpget_p** form of the command uses the port number specified by the **Port** property.

SNMP V1 and V2c

The get operation for SNMP V1 and V2C that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=getcommand hostname=hostname oid=oid type=type value=value
community=community
```

Where *getcommand* takes one of the following values, depending on your version of SNMP:

SNMPGET
SNMPGET_V2C

The operation that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=getcommand hostname=hostname oid=oid type=type value=value
community=community port=port
```

Where *getcommand* takes one of the following values, depending on your version of SNMP:

SNMPGET_P
SNMPGET_V2C_P

SNMP V3

The get operation for SNMP V3 that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=SNMPGET_V3 hostname=hostname userid= userid oid=oid type=type
```

The operation that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=SNMPGET_V3_P hostname=hostname userid=userid oid=oid type=type
port=port
```

Examples

The following example retrieves the value of the **sysDescr** object in the SNMP agent. The command uses port 161 on the agent and SNMP V2c:

```
$OMNIHOME/bin/nco_probeeventfactory -host http://test28.example.com/probe/
common -port 6789 snmpreq=SNMPGET_V2C_P hostname=probehost.example.com
oid=1.3.6.1.2.1.1.1.0 community=private port=161
```

The following example also retrieves the value of the **sysDescr** object. However, it uses SNMP V3, the default port on the SNMP agent, and a user name of **snmpadmin**:

```
$OMNIHOME/bin/nco_probeeventfactory -host http://test28.example.com/probe/
common -port 6789 snmpreq=SNMPGET_V3 hostname=probehost.example.com
userid=snmpadmin oid=1.3.6.1.2.1.1.1.0
```

Set operations

Set operations set the value of data items in the SNMP agent.

There are two set operations for each version of SNMP.The **snmpset** form of the command uses the default port number to communicate with the probe, and the **snmpset_p** form of the command uses the port number specified in the command

SNMP V1 and V2c

The set operation for SNMP V1 and V2c that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=setcommand hostname=destinationhost oid=oid type=type value=value
community=community
```

Where *setcommand* takes one of the following values, depending on your version of SNMP:

SNMPSET
SNMPSET_V2C

The operation that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=setcommand hostname=destinationhost oid=oid type=type value=value
community=community port=port
```

Where *setcommand* takes one of the following values, depending on your version of SNMP:

SNMPSET_P
SNMPSET_P_V2C

SNMP V3

The set operation for SNMP V3 that uses the default port on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=SNMPSET_V3 hostname=destinationhost userid=userid oid=oid type=type
value=value
```

The operation that specifies the port to use on the SNMP agent has the following format:

```
$OMNIHOME/bin/nco_probeeventfactory -host probehost -port probeport
snmpreq=SNMPSET_V3_P hostname=destinationhost userid=userid oid=oid type=type
value=value port=port
```

Examples

The following example sets the value of the **sysLocation** object in the SNMP agent to `First floor computer room`. The command uses the default port on the agent and SNMP V1:

```
$OMNIHOME/bin/nco_probeeventfactory -host http://test28.example.com/probe/
common -port 6789 snmpreq=SNMPSET hostname=probehost.example.com
oid=1.3.6.1.2.1.1.6.0 type=s value="First floor computer room"
community=private
```

The following example also sets the value of **sysLocation**. However, it uses SNMP V3, port 4990 on the SNMP agent, and a user name of **snmpadmin**:

```
$OMNIHOME/bin/nco_probeeventfactory -host http://test28.example.com/probe/
common -port 6789 snmpreq=SNMPSET_V3_P hostname=probehost.example.com
userid=snmpadmin oid=1.3.6.1.2.1.1.6.0 type=s value="First floor computer room"
port=4990
```

Storing commands in `nco_http.props` properties file

You can use the `nco_http.props` file to hold frequently used command characteristics.

If you have a particular command that you send to the probe regularly, you can store characteristics of that command in the **nco_http** properties file (`$OMNIHOME/etc/nco_http.props`). Once you have done that, the format of the **nco_http** or **nco_probeeventfactory** command line is simplified.

You can use the one or more of the following **nco_http** properties to hold default values for the equivalent options on the **nco_http** command line:

Data
DataType
Method

URI

Specify the value of each property in the same way as you would on the command line. Once you have these values in place you do not need to specify the corresponding command line switch unless you want to override the value of the property. When using **nco_probeeventfactory**, the content of the **Data** property replaces the name-value pairs you specify on the command line.

The following is an example of the use of the properties file and the simplification of the **nco_http** command that results. In this example, the **nco_http** properties file contains the following values (note that line breaks appear for presentational purposes only; when editing the properties use one line for each property value):

```
Data : '{"eventfactory":[{"snmpreq:"SNMPSET", "hostname":"9.127.96.134",
"oid":"1.3.6.2.1.1.6.0",  "type":"s", "value":"First floor computer room",
"community:"private", "port"161}]}'
DataType : 'application/JSON'
Method : 'POST'
```

To use this set of values use the following **nco_http** command:

```
$OMNIHOME/bin/nco_http -uri http://test28.example.com:6789
```

Rules file

Customize the rules file so that the probe can process HTTP POST requests.

By default, the rules to process HTTP POST requests are not enabled. To enable the rules:

1. Open the rules file (`mttrapd.rules`) in a text editor and locate the following lines:

```
if (exists($snmpreq) && !match($snmpreq,""))
{
    # Uncomment the line below to enable rules file for bi-directional
    # functionality. Specify the path of the rules file if it is not
    # in the same directory with this rules file.

    # include "mttrapd.bidir.rules"
}
else if( match( @Manager, "ProbeWatch" ) )
```

2. Remove the comment marker from the line: `# include "mttrapd.bidir.rules"`.
3. Save the file and exit from the text editor.

The `mttrapd.bidir.rules` file supplied in the same directory as the `mttrapd.rules` file. If the file is moved to a separate directory, specify its full path in the `include` statement.

Changes to the mttrapd.rules file

With the introduction of the cached host name resolution feature in Release 20 of this probe, several updates have been made to the `mttrapd.rules` file.

In previous releases, `Node` (among other tokens) has been used to compose the `Identifier` event. This release of the `mttrapd.rules` file has the following changes:

1. `NodeAlias` supplants the role of `Node` in the composition of `Identifier`.

   When the host name resolution feature is enabled (that is when the **NoNameResolution** property is set to `0`), the `Node` token for the same trap alert can be amended in runtime either from IP address to host name, or from one host name value to a different host name value. Because of this, `Node` can no longer be used in the `Identifier` event, otherwise deduplication will fail. So `NodeAlias` is used instead in the `Identifier` event. For example:

   ```
   @Identifier = "" + @NodeAlias + "" + @Agent + "" + $generic-trap + "" +
   $specific-trap + "" + $1 + ""
   ```

2. The Node column in the event list is subject to mandatory update in the `mttrapd.rules` file when the **NoNameResolution** property is set to `0`:

```
if ( int(%NoNameResolution) == 0 )
{
      update(@Node)
}
```

If you currently deploy the SNMP Probe with a custom `mttrapd.rules` file, you must make similar changes to your custom rules file to enable the event deduplication and to update the Node column.

Properties and command line options

You use properties to specify how the probe interacts with the device. You can override the default values by using the properties file or the command line options.

The following table describes the properties and command line options specific to this probe. For more information about generic Netcool/OMNIbus properties and command line options, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table 19. Properties and command line options

Property name	Command line option	Description
ActiveHostnameDuration *integer*	`-activehostname duration` *integer*	Use this property to specify how long (in minutes) an ip-host name value pair stays in the table. When this length of time since the probe last visited the element has elapsed, the probe removes it from the table held in memory and writes the details to the flat file for discarded IP address-host name pairs. You must specify a value between 5 and 10080 minutes. The default is `30`.
BindAddress *string*	`-bindaddress` *string*	Use this property to specify the IP address to which the probe binds. The default is " ". **Note :** The IP address can be in either IPv4 or IPv6 format depending on the setting of the **Protocol** property.
ConfPath *string*	`-snmpconfpath` *string*	Use this property to specify the path of directories that contain configuration information for the SNMP probe engine. The default is `$OMNIHOME/probes/`*arch*`/ $OMNIHOME/var` or *`${LOGNAME} /`* `solaris/Omnibus36/var`. **Note :** The directories specified using this property must be separated by a colon (:) on UNIX and Linux or a semi-colon (;) on Windows. The configuration file is named `mttrapd.conf`.

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
ConfigCryptoAlgo *string*	`-configcryptoalgo` *string*	Use this property to specify the cipher algorithm used in crypto configuration processing. This property takes the following values: • AES_FIPS • AES The default is `""`.
ConfigCryptoKeyFile *string*	`-configcryptokeyfile` *string*	Use this property to specify the keyfile used in crypto configuration processing. Use the `nco_keygen` tool to generate the keyfile which will be used to encrypt the `mttrapd.conf` file. The default is `""`.
DSALog *integer*	`-dsalog` *integer*	Use this property to specify whether the probe logs traps that are lost because the trap queue has become full: `0`: The probe does not log traps that are lost. `1`: The probe logs traps that are lost. The default is `0`.
DSAPeriod *integer*	`-dsaperiod` *integer*	Use this property to specify the time, in seconds, that traps are logged when the **DSALog** property has a value of `1`. The default is `30`.

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
EnableCryptoConfig *integer*	`-enablecryptoconfig` *integer*	Use this property to enable crypto processing on the SNMP configuration file (`mttrapd.conf`). This property takes the following values: `0`: The probe does not perform crypto processing. `1`: The probe performs the following crypto processing: a) The probe validates the values specified by the **ConfigCryptoAlgo** and **ConfigCryptoKeyFile** properties during startup b) The probe processes the encrypted `mttrapd.conf` file in the `ConfPath` directory using the cipher algorithm specified by the **ConfigCryptoAlgo** property and the key specified by the **ConfigCryptoKeyFile** property to extract the SNMP configuration contents. c) The probe produces the `<PersistentDir>/mttrapd.conf` file in encrypted format. `2`: The probe performs the following crypto processing: a) The probe validates the values specified by the **ConfigCryptoAlgo** and **ConfigCryptoKeyFile** properties during startup b) The probe runs the decryption test on any configurations files found in the **ConfPath** and **PersistentDir** directories. The probe will exit if the test fails. c) The probe processes the encrypted `mttrapd.conf` file in the `ConfPath` directory using the cipher algorithm specified by the **ConfigCryptoAlgo** property and the key specified by the **ConfigCryptoKeyFile** property to extract the SNMP configuration contents. d) The probe produces the `<PersistentDir>/mttrapd.conf` file in encrypted format. The default is `0`.

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
EngineInfoProbeWatch *integer*	-engineinfoprobewatch *integer*	Use this property to enable the probe to generate ProbeWatch messages related to engine information. This property takes the following values: 0: The probe does not send ProbeWatch messages related to engine information. 1: The probe sends a ProbeWatch message when engine inconsistencies are detected. The default is 0. **Note :** For details about ProbeWatch messages related to SNMP engine information, see "SNMP engine update automation processing" on page 6.
FlushBufferInterval *integer*	-flushbufferinterval *integer*	Use this property to specify the interval (in seconds) that the probe waits before flushing the buffer contents to the ObjectServer. The default is 0.
Heartbeat *integer*	-heartbeat *integer*	Use this property to specify the length (in seconds) of the heartbeat period. If the probe receives no traps for this length of time, it sends a heartbeat ProbeWatch message to the ObjectServer. The default is 60.
HostnameTableSize *integer*	-hostnametablesize *integer*	Use this property to specify the maximum number of IP address-host name pairs that the probe writes to a table in memory. If the table exceeds this size, the probe creates no new IP node entries in the table and writes an error message to its log. You must specify a value between 50 and 50000 entries. The default is 20000.
LogStatisticsInterval *integer*	-logstatisticsinterval *integer*	Use this property to specify the interval (in seconds) at which the probe logs internal probe statistics. The statistics reported include the following: • Size of the trap queue and the inform queue • Number of traps read since the last rollover • Number of traps processed since the last rollover The default is 0 (does not log the internal statistics).

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
MIBDirs *string*	`-mibdirs` *string*	Use this property to specify where the probe searches for MIB modules. This is in the form of a colon-separated list of directories. The default is `$OMNIHOME/common/mibs`. **Note :** The directories specified using this property must be separated by a colon (:) on UNIX or a semicolon (;) on Windows.
MIBFile *string*	`-mibfile` *string*	Use this property to specify the name of the MIB file. The default is `$OMNIHOME/probes/`*arch*`/mib.txt`. **Note :** If you are using a rules file generated by the trapd converter, you must set this property to point to an empty file; for example, `/dev/null`.
MIBs *string*	`-mibs` *string*	Use this property to specify which MIB modules the probe loads. Your entry should be in the form of a colon-separated list of modules. The default is `ALL` (this instructs the probe to load all modules available in the list of directories specified by the **MIBDirs** property).
NoNameResolution *integer*	`-nonameresolution` (This is equivalent to **NoNameResolution** with a value of `1` `-nameresolution` (This is equivalent to **NoNameResolution** with a value of `0`	Use this property to specify whether the probe performs name resolution on IP addresses: `0`: The probe performs name resolution. `1`: The probe does not perform name resolution. The default is `1`.
NoNetbiosLookups *integer*	`-nonetbioslookups` *integer* (This is equivalent to **NoNetbiosLookups** with a value of `1`.) `-usenetbioslookups` *integer* (This is equivalent to **NoNetbiosLookups** with a value of `0`.)	Use this property to specify whether the probe performs netbios lookups during DNS queries: `0`: The probe performs lookups `1`: The probe does not perform netbios lookups The default is `0`. **Note :** This property is only available on Windows operating systems.

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
NonPrintableAsHex *integer*	`-nonprintableashex` *integer*	Use this property to specify whether the probe sets non-printable characters to their hexadecimal values: `0`: The probe does not set non-printable characters to their hexadecimal values `1`: The probe sets non-printable characters to their hexadecimal values The default is `0`.
PersistentDir *string*	`-persistentdir` *string*	Use this property to specify where the persistent configuration information is stored. The default is `$OMNIHOME/var`.
Port *integer*	`-port` *integer*	Use this property to specify the port to which the probe listens for SNMP traffic. The default is `162`.
Protocol *string*	`-protocol string` `-udp` (This is equivalent to **Protocol** with a value of `UDP` or `-protocol UDP`.) `-tcp` (This is equivalent to **Protocol** with a value of `TCP` or `-protocol TCP`.) `-all` (This is equivalent to **Protocol** with a value of `ALL` or `-protocol ALL`.) `-any` (This is equivalent to **Protocol** with a value of `ANY` or `-protocol ANY`.) `-udp6` (This is equivalent to **Protocol** with a value of `UDPIPV6` or `-protocol UDPIPV6`.) `-tcp6` (This is equivalent to **Protocol** with a value of `TCPIPV6` or `-protocol TCPIPV6`.) `-allipv6` (This is equivalent to **Protocol** with a value of `TCPIPV6` and `UDPIPV6` or `-protocol ALLIPV6`.)	Use this property to specify the network protocol that the probe uses. The default is `UDP`. If the probe is running in an IPv4 environment, specify one of the following values: • `UDP` • `TCP` • `ALL` • `ANY` **Note :** The values `ANY` and `ALL` are interchangeable. If the probe is running in an IPv6 environment, specify one of the following values: • `UDPV6` • `TCPV6` • `ALLIPV6`

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
QuietOutput *integer*	`-quietoutput` (This is equivalent to **QuietOutput** with a value of 1.) `-noquietoutput` (This is equivalent to **QuietOutput** with a value of 0.)	Use this property to specify whether the probe outputs tokens that correspond to an OID with symbolic OID expansion: 0: The probe outputs tokens with symbolic OID expansion 1: The probe outputs tokens without symbolic OID expansion The default is 1.
RefreshHostname Interval *integer*	`-refreshhostname interval` *integer*	Use this property to specify the interval (in minutes) that the probe leaves between successive DNS queries to the resolve the host name of an IP. You must specify a value between 15 and 10080 minutes. The default is 60.
ReuseEngineBoots *integer*	`-reuseengineboots` (This is equivalent to **ReuseEngineBoots** with a value of 1.) `-noreuseengineboots` (This is equivalent to **ReuseEngineBoots** with a value of 0.)	Use this property to specify whether the probe reuses the engine ID and the number of SNMP engine boots specified in the `mttrapd.conf` file. The property takes the following values: 0: The probe does not reuse the engine ID and number of SNMP boots. 1: The probe reuses the engine ID and the number of SNMP boots. The default is 1.
SleepTime *integer*	`-sleeptime` *integer*	Use this property to specify the poll time (in seconds) of the trap list. If there are no traps to be processed, the probe sleeps for this amount of time before polling the trap queue again. The default is 1.

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
SnmpConfigChange DetectionInterval *integer*	`-snmpconfigchange detectioninterval` *integer*	Use this property to specify the frequency (in minutes) that the probe checks for changes to the `mttrapd.conf` configuration file. If the file has changed, the probe loads its contents. The value of this property can be between `0` and `10080` (equivalent to `1` week). A value of `0` turns off the automatic detection and loading of a changed configuration file. Any other value specifies the frequency that the probe checks for changes to the file. The default is: `1` **Note :** The `mttrapd.conf` file is specified by the **ConfPath** property. If **ConfPath** is not set, the automatic detection facility will be disabled.
snmpv3MinSecurity Level *string*	`-snmpv3minsecurity level` *string*	Use this property to specify which SNMPv3 traps the SNMP Probe processes. By default the probe processes SNMPv3 traps of all security levels. Traps and informs have the following levels of security: • NoAuth - No authorization and no privacy • AuthNoPriv - Authorization, but no privacy • AuthPriv - Authorization and privacy This property takes the following values: 1: The probe processes SNMP V3 Traps/ Inform PDUs of security level NoAuth, AuthNoPriv, or AuthPriv. 2: The probe processes SNMP V3 Traps/ Inform PDUs of security level AuthNoPriv or AuthPriv. 3: The probe processes SNMP V3 Traps/ Inform PDUs of security level AuthPriv. The default is `1`.
snmpv3ONLY *integer*	`-snmpv3only` (This is equivalent to **snmpv3ONLY** with a value of `1`.) `-nosnmpv3only` (This is equivalent to **snmpv3ONLY** with a value of `0`.)	Use this property to specify that the probe only processes SNMPv3 traps and informs. This allows you to limit event processing. This property takes the following values: • 0: The probe processes all types of SNMP traps. • 1: The probe only processes SNMPv3 traps. The default is `0`.

Table 19. Properties and command line options (continued)

Property name	Command line option	Description
SocketSize *integer*	`-socketsize` *integer*	Use this property to specify the size (in bytes) of the kernel buffer on the socket being used. This is set on a per-socket basis. A higher value increases the number of traps that the probe can handle. For UDP traps, this may improve performance. The default is 8192. **Note :** The minimum value for the **SocketSize** property is 128 bytes; the default is 8192 bytes. In the majority of cases, the default size is recommended.
TrapQueueMax *integer*	`-trapqueuemax` *integer*	Use this property to specify the maximum number of traps that can be queued for processing at any one time. The probe discards any traps received while the buffer is full. The default is `20000`.
TrapStat *integer*	`-trapstat` *integer*	Use this property to enable trap statistics collection and the following custom rule functions used in the `mttrapd_flood_control.rules` file: • `drop_list_contains()` • `drop_list_remove()` • `drop_list_add()` • `read_trap_count()` • `read_drop_count()` • `get_queue_size()` `0`: The probe does not collect trap statistics. `1`: The probe collects trap statistics. The default is `0`.

Table 19. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
UsmUserBase *integer*	`-usmuserbase` *integer*	Use this property to specify whether the probe reads the `mttrapd.conf` file in the directory specified by the **PeristentDir** property or the **ConfPath** property or both of those directories. The property takes the following values: `0`: The probe uses the file in **ConfPath** directory. 1: The probe uses the file in the **PersistentDir** directory. 2: The probe uses the files in both the **PeristentDir** and **ConfPath** directories. The default is 2.

Elements

The probe breaks event data down into tokens and parses them into elements. Elements are used to assign values to ObjectServer fields; the field values contain the event details in a form that the ObjectServer understands.

Static and dynamic elements are described in the following topics:

- "Static elements" on page 48
- "Dynamic elements" on page 50

Static elements

The following table describes the static elements that the probe generates:

Table 20. Static elements		
Element name	**SNMP version**	**Element description**
`$community`	V1 and V2c	This element contains the SNMP community string.
`$contextEngineID`	V3	This element identifies the engine associated with the data.
`$enterprise`	V1	This element contains the SNMP enterprise string.
`$EventCount`	V1, V2c, and V3	This element displays the number of traps processed during the current execution of the probe.
`$generic-trap`	V1	This element contains the SNMP generic trap integer value.

Table 20. Static elements (continued)

Element name	SNMP version	Element description
$hostname	V1, V2c, and V3	This element contains the host name or IP address of the SNMP agent that is the subject of a GET or SET command.
$IPaddress	V1, V2c, and V3	This element contains the IP address (origin of the SNMP trap).
$NodeAlias	V1, V2c, and V3	This element contains the node name (origin of the SNMP trap). This will be the IP address if node name cannot be resolved.
$notify	V2c and V3	This element displays the notify V2c specific field.
$oid	V1, V2c, and V3	This element contains the identifier of the object in the SNMP agent that is the subject of a GET or SET command.
$PeerAddress	V1, V2c, and V3	This element contains the host name or IP address where the SNMP trap was received from.
$PeerIPaddress	V1, V2c, and V3	This element contains the IP address where the SNMP trap was received from.
$port	V1, V2c, and V3	This element identifies the port on the SNMP agent to use for a GET or SET command.
$Protocol	V1, V2c, and V3	This element contains the protocol of the trap received. This can be either UDP or TCP.
$ReceivedPort	V1, V2c, and V3	This element contains the port number where the SNMP trap was received from. This is determined by the Port property.
$ReceivedTime	V1, V2c, and V3	The time that the SNMP packet was received from the network interface.
$ReqId	V1	This element contains the SNMP request ID.
$securityEngineID	V3	This element contains the engine ID of the authoritative SNMP entity. For informs, this is the engine ID of the probe. For traps, this is the engine ID of the source of the trap.

Table 20. Static elements (continued)

Element name	SNMP version	Element description
$securityLevel	V3	Security level of the trap or inform: noAuth: The trap or inform had no authentication and no privacy authNoPriv: The trap or inform had authentication, but no privacy authPriv: The trap or inform had authentication and privacy
$securityName	V3	This element contains the security name used for trap authentication.
$SNMP_Version	V1, V2c, and V3	This element contains the the value 1 for SNMP V1 traps and the value 2 for SNMP V2c traps.
$specific-trap	V1	This element contains the SNMP specific trap integer value.
$type	V1, V2c, and V3	This element identifies the type of object of a GET or SET command.
$Uptime	V1 and V2c	This element contains the SNMP uptime for traps expressed in the format 0:00:00.
$UpTime	V1 and V2c	This element contains the SNMP uptime for traps expressed as an integer.
$value	V1, V2c, and V3	This element contains the value to assign to the object identified by the $oid element in a SET command.

Dynamic elements

The other elements that the probe generates are created dynamically and are entirely dependent on the network devices. The varbind variables that are generated by the SNMP trap are mapped to elements called $1,$2,$3, and so on. For each varbind, the object ID is placed in a corresponding element called $OID1, $OID2, $OID3, and so on up to the number of varbind elements.

Note : Previous versions of the SNMP Probe (pre version 3.5) had no leading dot (.) in the $OIDn elements, whereas the latest probe does include the leading dot; if you are upgrading from an old version of the probe, your rules files may need updating.

The probe can also generate the following elements from various representations of the varbind variables:

- $*n*_raw - raw string representation of the varbind variables (containing all control characters)
- $*n*_text - printable text representation of the varbind variables (with non-printable characters replaced with periods)
- $*n*_hex - hexadecimal representation of the varbind variables

  Note : The $*n*_raw, $*n*_text, and $*n*_hex elements are only available for SNMP variables of type OCTET-STRING.

Error messages

Error messages provide information about problems that occur while running the probe. You can use the information that they contain to resolve such problems.

The following table describes the error messages specific to this probe. For information about generic Netcool/OMNIbus error messages, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table 21. Error messages

Error	Description	Action
`Error: ipv6_address is not a valid address for Protocol protocol`	The IP address specified for the **BindAddress** property was not in IPv4 format and not valid for the specified protocol.	Check the value specified for the **BindAddress** property; if the **Protocol** property is set to TCP, UDP, ALL, or ANY, this IP address must be specified in IPv4 format.
`Error: ipv4_address is not a valid address for Protocol protocol`	The IP address specified for the **BindAddress** property was not in IPv6 format and not valid for the specified protocol.	Check the value specified for the **BindAddress** property; if the **Protocol** property is set to TCPV6, UDPV6, or ALLIPV6, this IP address must be specified in IPv6 format.
`Error: Unknown authentication protocol`	A user entry in the configuration file contains an unrecognized value for the authentication protocol	The preface to the message includes the line number in the configuration file that caused the error. Check that user entry to ensure that the authentication protocol is one of the supported types and that the protocol is specified in upper case.
`Failed to parse SNMP PDU, Version unrecognised!!!PDU command was invalid`	The probe failed to process the traps.	Check that the device is running correctly.
`get_enginetime() got less-than-zero engineID_len`	An incoming inform did not contain an engine ID. The user entry in the configuration file does not provide a default one to use.	Ensure that the SNMP agent includes an engine ID in its informs. Or add a default engine ID to the user record in the configuration file.
`IP token[ip_value] not usable in the Protocol setting [protocol], no insertion to the table.`	The probe has received a trap from a host using an IP protocol other than that specified by the **Protocol** property.	Change the setting of the **Protocol** property to match the traps that the probe is receiving. For further details refer to the "Writing resolved host names and discarded host names to flat files" on page 21.

Table 21. Error messages (continued)

Error	Description	Action
`Line line_number: Probe's IP-Hostname pairs has reached the limit as configured in HostnameTableSize property. The current and all subsequent entries in the file will not be parsed.`	The internal table that the probe uses to hold IP-host name details has reached the maximum size specified by the **HostnameTableSize** property. So the current entry and all subsequent entries will not be parsed.	This message is for your information only. No action is required, but you may need to increase the value set for the **HostnameTableSize** property.
`Line line_number: Hostname token exceeds max length hostname_lentgh, the string is truncated.`	A host name retrieved following a DNS query exceeds 255 characters. The probe will truncate the host name, and only write the first 255 characters to the internal table.	This message is for your information only. No action is required.
`Cached IP-hostname list is full. No instantaneous and periodic hostname resolution to be done on IP[%s]`	The internal table that the probe uses to hold IP-host name details has reached the maximum size specified by the **HostnameTableSize** property. So the new IP address of the trap received by the probe cannot be written to the table.	This message is for your information only. No action is required.
`Line line_number IP tokenip_address not usable in the Protocol setting protocl_id.`	The probe received a trap from a node whose IP address does not match the format expected by the probe.	Check the value specified the **Protocol** property.
`Missing input to SNMPSET or SNMPGET request`	One or more arguments are missing from the **nco_http** command line when managing the SNMP agent through the probe's HTTP/ HTTPS interface.	Check the command and make sure it contains all of the required arguments as defined in "Sending commands using nco_http" on page 31.
`plaintextscopedPDU parsing returned NULL - Possible rootcause: decrypted text is incorrect in format.`	The privacy password in a trap or inform is incorrect.	Check that the password is correctly specified in the configuration file and that the SNMP agent is specifying the password correctly.
`Property ActiveHostnameDuration [%d] is not within valid range (%d - %d) mins. Default to value [%d]`	The **ActiveHostnameDuration** property has been set to an invalid value in the properties file. The probe is using the default value of 30 minutes instead.	Set **ActiveHostnameDuration** to a value between 5 and 10080 minutes.

Table 21. Error messages (continued)

Error	Description	Action
`Property RefreshHostnameInterval [%d] is not within valid range (%d - %d) mins. Default to value [%d]`	The **RefreshHostnameInterval** property has been set to an invalid value in the properties file. The probe is using the default value of 60 minutes instead.	Set **RefreshHostnameInterval** to a value between 15 and 10080 minutes.
`Property HostnameTableSize [%d] is not within valid range (%d - %d). Default to value [%d`	The **HostnameTableSize** property has been set to an invalid value in the properties file. The probe is using the default value of 20000 entries instead.	Set **HostnameTableSize** to a value between 50 and 50000 entries.
`protocol not known`	An invalid protocol has been specified.	Check the value specified for the **Protocol** property.
`scopedPDU desperate parsing still returned NULL`	A trap or inform received from the SNMP agent contains an error in the specified authentication protocol, privacy type, authentication password, privacy password, or engine ID.	A subsequent message contains further indication of the reason for the error. Use that to determine the cause and to take corrective action.
`search_enginetime_list( ) error.`	An inform contained an unrecognized engine ID.	Ensure that the SNMP agent specifies the correct engine ID.
`UDP snmp_open: Unknown host (Address already in use) Failed to open UDP sessionUnable to get hold of session link pointer`	As another process is running on the port specified, it is not available for this session.	Specify a different port either using the command line, or using the **Port** property in the properties file.
`TRAP_FLOOD: IP=[ip_address] has [number_of_traps] entries in the queue. Dropping and discarding.`	The probe has dropped the host IP addressed identified in the message and marked it as blocked.	The number of traps queued for the IP address exceeded the maximum allowed.
`TRAP_FLOOD: IP=[ip_address] has sent [number_of_traps] traps in the last [number_of_seconds] seconds, rate = [traps_per_second] so remains banned.`	The probe is continuing to block the host identified in the message.	The probe checked the blocked host to determine whether the number of traps received by the host has slowed, but found that it has not.

Table 21. Error messages (continued)

Error	Description	Action
TRAP_FLOOD: IP=[ip_address] has only sent [number_of_traps] traps in the last [number_of_seconds] seconds, rate = [traps_per_second] so is being allowed again.)	The probe is unblocking the host identified in the message.	The probe checked the blocked host to determine whether the number of traps received by the host has slowed. As the number has slowed, the probe is no longer blocking the host.
TRAP_FLOOD: IP=[ip_address] ban ending, enabling.	The probe has now marked the host identified in the message as no longer blocked.	The probe unblocked the host.
TRAP_FLOOD_REPORT_START at [time] with [number_of_traps] traps in the queue.	Indicates the start of the periodic Trap Flood Report.	This messages is for information only. No action is required.
TRAP_FLOOD_REPORT_HOST: ip=[ip_address] status=[DROP] since [number_of_seconds] seconds. inqueue=[number_of_traps] nosdrop=[number_of_traps _dropped]	Indicates that the specified host is blocked.	This messages is for information only. No action is required.
TRAP_FLOOD_REPORT_HOST: ip=[ip_address] status=[ACCEPT] inqueue=[number_of_traps] nosdrop=[number_of_traps _dropped]	Indicates that the previously blocked host is now unblocked.	This messages is for information only. No action is required.
TRAP_FLOOD_REPORT_END at [time]	Indicates the end of the periodic Trap Flood Report.	This messages is for information only. No action is required.
TrapStat is disabled.	The **TrapStat** property is set to 0.	If you want to use the trap flood monitoring functionality, set the **TrapStat** property to 1. Otherwise, leave the **TrapStat** property set to 0.

ProbeWatch messages

During normal operations, the probe generates ProbeWatch messages and sends them to the ObjectServer. These messages tell the ObjectServer how the probe is running.

The following table describes the ProbeWatch messages that the probe generates. For information about generic Netcool/OMNIbus ProbeWatch messages, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table 22. ProbeWatch messages

ProbeWatch message	Description	Triggers or causes
`Dropping the traps`	The probe is dropping events.	The trap limit specified by the **`TrapQueueMax`** property has been exceeded.
`Going Down`	The probe is shutting down.	The probe is shutting down after performing the shutdown routine.
`Heartbeat Message`	The heartbeat message that the probe sends to the ObjectServer.	The probe has not received any events for the time specified by the **`Heartbeat`** property. This may be useful for debugging purposes.
`Running ...`	The probe is running normally.	The probe has just been started up.
`Unable to get events`	A problem occurred while trying to listen for traps.	Either there was a problem initializing the connection due to insufficient memory or (if this message was sent after some events had been parsed) there was a connection failure.
`mttrapd on [ip_address] is blocked`	The host at the IP address specified is currently blocked.	The host at the IP address specified is currently sending too many traps.
`mttrapd on [ip_address] is unblocked`	The previously blocked host at the IP address specified is now unblocked.	The rate of traps sent by a previously blocked IP address has reduced. While the probe is exiting, each IP in the drop list will be removed as part of the clean up process.

Table 22. ProbeWatch messages (continued)		
ProbeWatch message	**Description**	**Triggers or causes**
TRAP_FLOOD_REPORT_START at [time] TRAP_FLOOD_REPORT_HOST: *message* TRAP_FLOOD_REPORT_STOP at [time]	These three ProbeWatch messages will appear together and form the periodic flood report. The first and the last message indicate the start and the end of the reporting period. The middle part of the message contains the trap statistics for the host at the IP address specified. This can be a combination of the following: TRAP_FLOOD_REPORT_HOST:ip = [ip_addr] status=[DROP] since "timestamp" seconds. inqueue=[ip_trap_count] nosdrop=[ip_drop_count] and TRAP_FLOOD_REPORT_HOST:ip = [ip_addr]status=[ACCEPT] inqueue=[number_of_traps] If the message exceeds the Summary length limit, it will be truncated.	This is the periodic flood report that the probe generates. The periodic flood report shows the trap statistics of the blocked, or a newly unblocked IP addresses (ip_addr) specified in the ProbeWatch messages.

Running the probe

Before running the probe for the first time, you must specify a minimum set of properties.

Before you run the probe, you must specify the following properties:

- **Port** - this property specifies the port on which the probe listens for SNMP traffic.
- **Protocol** - this property specifies the network protocol that the probe uses.

Note : To run the probe on a machine where another trapd process is running (for example, HP NNM or SunNet Manager), you must specify an SNMP port that is not being used by the other trapd process.

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

```
$NCHOME/omnibus/probes/arch/nco_p_mttrapd
```

To start the probe from the Windows command prompt, use the following command:

```
%NCHOME%\omnibus\probes\win32\nco_p_mttrapd
```

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

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

   ```
   %NCHOME%\omnibus\probes\win32\nco_p_mttrapd -install -depend NCOObjectServer
   ```
2. To run the probe on a different host to the ObjectServer, use the following command to register it as a service:

```
%NCHOME%\omnibus\probes\win32\nco_p_mttrapd -install
```

3. Start the probe service using the Microsoft Services Management Console.

Note : On the Windows operating system, always specify the `-utf8enabled` command line option when starting the probe if you have configured the probe to process UTF-8 (multibyte) characters.

Running the probe as suid root

The probe can be run as `suid` root without compromising system security. In this mode, the probe drops its root privileges after it has opened the SNMP session and before the IBM Tivoli Netcool/OMNIbus probe library starts. This mode grants privileged port use.

Note : Running the probe as `suid` root causes environment variables to be ignored. This procedure only works if the IBM Tivoli Netcool/OMNIbus installation is on a local file system and is installed in the default location.

Information on how to run the probe as `suid`, visit the IBM Tivoli Netcool Information Center.

Running the probe on AIX

Problems with library paths and events not being read properly by the probe have been reported by some users of the AIX operating system.

If you experience these issues, one possible workaround is to run the probe from the `$NCHOME/omnibus/platform/`*`arch`*`/lib` directory.

To do this, use the following steps:

1. Change to the following directory:

   ```
   /usr/Omnibus/platform/aix4/lib
   ```

2. Run the following command:

   ```
   $NCHOME/omnibus/probes/nco_p_mttrapd options
   ```

If running the probe from this directory does not work, contact IBM Software Support.

Known Issues

At the time of release, a number of known issues were reported that you should be aware of when running the probe.

Error messages from remote management commands on Solaris

When remotely managing the SNMP agent over the probe's HTTP/HTTPS interface on a Solaris system, the following messages appear in the error log:

```
Error: E-UNK-000-000: SNMP Message (priority=3):
/netcool/omnibus/var/snmpv3/hosts/targethost.conf: Not owner
Error: E-UNK-000-000: SNMP Message (priority=3):
/netcool/omnibus/var/snmpv3/hosts/targethost.local.conf: Not owner
Error: E-UNK-000-000: SNMP Message (priority=3):
/netcool/omnibus/var/hosts/target.conf: Not owner
Error: E-UNK-000-000: SNMP Message (priority=3):
/netcool/omnibus/var/hosts/targethost.local.conf: Not owner
```

In each case, *targethost* is replaced by the identity of the target for the command.

Despite these errors, the SNMP command is processed correctly. The messages result from a Solaris system call, because the files are not available. You can disregard these messages or choose to put the appropriate files in the named directories.

Unable to use nco_http on a Windows system

When using **nco_http** to send JSON data to the probe from a Windows systems, the value of the `-data` option is not processed correctly. Use either of the following workarounds:

- Store the content of the `-data` option in the `nco_http.props` file to avoid entering it on the **nco_http** command line.

  See "Storing commands in nco_http.props properties file" on page 37 for information on using the `nco_http.props` file to hold data for the **nco_http** utility.
- Use the **nco_probeeventfactory** utility instead of **nco_http**.

  See "Sending commands using nco_probeeventfactory" on page 35 for information on using the **nco_probeeventfactory** utility.

snmpget() function error messages from the Probe Rules Syntax Checker

The Probe Rules Syntax Checker can only access rules functions provided by the OMNIbus library.

The following functions are supported by the SNMP Probe but not by the OMNIbus library:

- `snmp_get()`
- `snmpget_p()`
- `snmpget_v2c()`
- `snmpget_v2c_p()`
- `snmpget_v3()`
- `snmpget_v3_p()`
- `snmpset()`
- `snmpset_p()`
- `snmpset_v2c()`
- `snmpset_v2c_p()`
- `snmpset_v3()`
- `snmpset_v3_p()`
- `drop_list_contains()`
- `drop_list_remove()`
- `drop_list_add()`
- `read_trap_count()`
- `read_drop_count()`
- `get_queue_size()`
- `print_engines()`
- `get_engine_info()`
- `update_engine()`
- `set_engine_correction()`

The Probe Rules Syntax Checker generates errors when checking the syntax of any rules files that contain any of these functions, or any other functions not in the OMNIbus library. You can disregard these messages.

Probe cannot run without `libncrypt`

On AIX and Windows operating systems, running OMNIbus version 7.3.0, the probe requires the **libncrypt** library. Make sure that you have installed the correct OMNIbus fix pack as set out in the patch dependencies section of the `description.txt` file supplied with the download package.

IPv6 not supported on Microsoft Windows

When running on a Microsoft Windows system, the probe does not support using IPv6 to communicate with the SNMP target device. That is, do not use any of the following values for the **Protocol** property on Microsoft Windows:

```
UPDV6
TCPV6
ALLIPV6
```

Traps dropped due to malformed timestamps

The SNMP probe only processes traps that have a timestamp less than or equal to four bytes. The probe drops any trap that has a malformed timestamp (one that is greater than four bytes in size).

Troubleshooting

Various issues arise as users work with the SNMP Probe (nco_p_mttrapd probe).

Using buffering to improve the performance of the probe

If the buffering facility is not used, the SNMP Probe queue increases in size until it is eventually exceeded.

The following example shows the log file messages that the probe writes to the log file as traps are received:

```
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51451
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51454
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51453
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51452
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51451
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51450
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51449
2015-07-09T17:57:26: Information: I-P_M-104-000: Number of items in the trap queue is 51449
```

If there are a high number of traps in the probe's queue (as there are in the example above), the probe will run more slowly. To reduce the size of the queue and thus improve the performance of the probe, you should use the buffering facility.

See “Queue and buffer settings” on page 23.

Other issues resolved by IBM Software Support

To help you diagnose and resolve other issues, refer to the link below to the SMC blog page that consolidates all the frequently asked questions and troubleshooting tips for the probe.

https://ibm.biz/BdHyF7.

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.

SC11-7728-14