

CMD V2.00

Release history

Version	Author	Comments
1.0	Gijsbert Wiesenekker	Initial release.
1.1	Gijsbert Wiesenekker	Added alias support.
1.2	Gijsbert Wiesenekker	Removed qos_float option.
1.3	Gijsbert Wiesenekker	Added qos_float back in again as you cannot mix NimQoSSendValue and NimQoSSendValueStdev calls.
1.4	Gijsbert Wiesenekker	Fixed publishQoS bug and added support for execution of commands like iostat that have to run continuously in the background.
1.5	Gijsbert Wiesenekker	Added support for exit code monitoring.
1.6	Gijsbert Wiesenekker	It is now possible to use the members of a USM group as the source for resource names. It is now possible to send QoS's as if originating from another probe. The probe now supports auto-clear for all threshold alerts.
1.7	Gijsbert Wiesenekker	Fixed a bug that caused certain cmd sections to be skipped. It is now possible to use expressions for the samplevalue. The documentation explains one of the examples in the supplied configuration file.
1.8	Gijsbert Wiesenekker	Enabled track changes. Disabling/enabling commands using active now works. It is now possible to use sub-directories of a directory as the source for resource names.
1.82	Gijsbert Wiesenekker	Each command can be scheduled at it's own interval and possibly at an exact time.
2.00 1.9	Gijsbert Wiesenekker	ActiveState Perl is no longer required on Windows. conf_cmd.exe is a standalone executable that you can use to construct a regular expression. You can specify the CI type, the CI name and the metric ID for QoS's if you want the data to show in USM.
2.00	Gijsbert Wiesenekker	<u>The standalone Perl probe executable on Windows is now called cmd.bin instead of cmd.exe for obvious reasons.</u> <u>The username/password to login to Nimbus can be specified as command-line options when running the probe interactively.</u> <u>The alarm description can be any valid Perl eval() expression.</u> <u>You can specify a suppression key for the alarm.</u>

Description

This probe can execute commands and turn the output into QoS messages and/or alarms. The

cmd 2.00

logmon probe can do this too, but the logmon probe gives you no control over the source and target for the QoS messages and the source of alarms, you cannot publish floating point values and the alarm expression is limited to simple comparisons only. For example, if you want to query the read- and write-latency of multiple Netapp devices using the native Netapp rsh command-line interface you cannot use the name of the Netapp device as the source for the QoS message or the alarm. This probe allows you to do so.

Installation

Ensure SDK_Perl 5.04 or greater is deployed to the robot that will run cmd.
Deploy the probe.

Usage

You can run the probe from the command-line using
cmd.bin [-u <username>] -p <password>

on Windows or

./cmd.pl [-u <username>] -p <password>

on Linux. The username/passsword is required to login to Nimbus. The default username is administrator.

UsageConfiguration

Double click the probe in Infrastructure Manager to raw configure it or edit the configuration file with a text-editor (recommended):

Name	Optional/Required	Description
interval	Optional. The default is 300.	The scheduling interval of the probe.
dap_addr	Optional. The default is undefined.	The Nimbus address of the dap probe if resources have been specified as USM groups.

For each command in the <cmds> section you specify:

Name	Optional or required	Description
active	Optional. Default: yes	If no the command is disabled. This allows you to keep a command that you currently no longer need for future reference.
description	Optional. Not used by the probe	A description of the command for documentation purposes.

Name	Optional or required	Description
resources	Optional. The default is the hostname.	<p>A comma-separated list of resources (IIS resource pools, Netapp devices, hostnames etc.) the command will be executed against. For remote resources the resource-name is often 'the' host-name or device-name, but sometimes the command can only be executed through a management interface that is only known by a different host-name or an IP-address, and sometimes a user-name has to be specified as well like root@IP-address. As this would make the resource name unsuitable for the source or target of QoS messages and the source of alarms, you can specify an alias in the resource-name by specifying a ':' followed by the alias-name. The alias-name can be used in the source or target of QoS messages and in the source of alarms.</p> <p>You can specify a USM group by enclosing the group name by a pair of ?, for example: resources = ?usm group?</p> <p>The resources will consist of all group members in that case. The advantage is that if you add a member to the USM group cmd will be automatically executed against it.</p> <p>You can specify the sub-directories of a directory by enclosing the directory name by a pair of !, for example: resources = !/ppm/metrics/lvs!</p>
cmd	Required.	<p>The command to execute. It will be executed by the Perl system command with redirection of stdout and stderr. The command can refer to the Perl variable \$resource, in which case the command will be executed against each resource in the list of resources in turn, for example:</p> <pre>cmd = uptime cmd = ssh host-name uptime cmd = ssh root@1.2.3.4:logical-name uptime cmd = ssh \$resource uptime cmd = cat /ppm/metrics/lvs/\$resource/storeprod</pre>
interval	Optional. The default is the scheduling interval of the probe.	The scheduling interval of the command.

Name	Optional or required	Description
percentage	Optional. The default is -1 (disabled).	A floating point value specifying at which time within the interval the command has to be scheduled. If equal to -1 the command will be run at probe startup, and subsequently at every interval, so if the probe is (re-)started at 2 minutes past the hour and the interval is equal to 5 minutes, the command will run at 2, 7, 12 minutes etc. past the hour. If the percentage is not equal to -1 the command will be run at exactly (to within 5 seconds and possibly limited by other scheduled commands) at the percentage of the interval past the hour and subsequently at every interval. So if the interval is equal to 5 minutes and the percentage is equal to 0 the command will be scheduled at 0, 5, 10.. minutes past the hour; if the interval is equal to 15 minutes and the percentage is 33.33, the command will be scheduled at 5, 20, 35 minutes past the hour.
qos_on_exit_code	Optional. The default is yes.	(Unix Only) if yes a QoS message will be published for the exit code of the command.
alarm_on_exit_code	Optional. The default is no.	A comma-separated list of valid Perl eval() expressions that refer to the Perl variable \$e to determine if an alarm has to be sent. The Perl variable \$e will be set to the exit code of the command. Optionally the expression can be prefixed by the alarm level using any of the keywords INFO:, WARNING:, MINOR:, MAJOR: or CRITICAL: For example if you want a warning alarm to be sent if the exit code is between 1 and 2 and a critical alarm to be sent if the exit code is larger than 3 you use: alarm_on_exit_code = WARNING:(\$e > 1) and (\$e < 2),CRITICAL:\$e > 3.

Name	Optional or required	Description
header and header_ <u>matchstart</u>	Optional. The default is undefined.	<p>Commands like <code>iostat</code> have to run for a certain amount of time (say 60 seconds) to collect data for the last 60 seconds. The output of these commands consists of two blocks: a header, the data since last boot (that often is not needed), the same header and the data for the last 60 seconds. If you only want the data from the second block header should contain a Perl regular expression to match the header of the block, and header_<u>matchstart</u> is the number of the block that contains the data. The supplied configuration file gives an example on how to do this for <code>iostat</code>.</p> <p>Now the probe will wait for a command like <code>iostat 60 2</code> to finish, so cannot execute other commands during that time. This is not an issue if there is only one command that takes only 60 seconds to finish in the configuration file, but is an issue if there is are multiple commands like <code>iostat 300 2</code>, <code>vmstat 300 2</code> in the configuration file, each taking 300 seconds to finish and the scheduling interval of the probe has been set to 300 seconds. Therefore the probe provides a Unix shell-script <code>async.sh</code> to launch these commands in the background. The probe does not wait for the command to finish, but reads the output of the last run of the command. You prefix the command that you would like to execute in the background by <code>./async.sh</code>, so:</p> <pre>./async.sh iostat 60 2</pre> <p>If you like you can also use <code>async.sh</code> to launch commands like <code>df</code> in the background that exit immediately, but because <code>df</code> exits immediately it would run continuously causing high CPU load. For these commands you should specify the time in seconds to wait before running the command again as the first argument to <code>async.sh</code>, so:</p> <pre>./async.sh -60 df</pre>


Name	Optional or required	Description
prid	Optional.	If defined QoS's will be sent as if originating from probe prid. This allows you to enhance other probes, and makes it very easy to add these QoS's to existing list-views and performance charts.

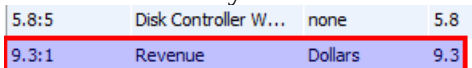
Now to turn the output of the commands into QoS and/or alarms you define watchers. For each watcher in the <watchers> section you specify:

Name	Optional or required	Description
regex	Required. The default is undefined.	A Perl regular expression without the leading and trailing '/' to turn the output into QoS messages and alarms. Each line of the output will be matched against the regexp. You use the usual Perl grouping metacharacters () to extract parts of the line into the Perl special variables \$1, \$2, etc. These special variables can be used to construct the samplevalue of the QoS and the name of target as follows: For each special variable that contains a samplevalue you define a sub-section with the name of that variable without the '\$' in the <matches> section. So if the special variables \$3 and \$5 contain samplevalues you define the sub-sections <3> and <5> in the matches section. For each sub-section you specify:
qos_definition	Optional.	If defined a QoS message for the samplevalue will be published. The qos_definition consists of the name, the group, the description, the unit and the abbreviation of the unit separated by a ':'. For example: IOSTAT_TPS:QOS_APPLICATION:tps:number:nr The QoS name will be prefixed by the (impersonated) probe name.
qos_float	Optional. The default is 0.	If equal to 0 the QoS messages will be published as an integer value. If equal to 1 the QoS will be published as a floating-point value.
samplevalue	Optional. The default is the value of the Perl special variable corresponding to the current sub-section.	If defined any Any valid Perl eval() expression that refers to the Perl special variables \$1, \$2 etc. <i>The evaluated expression will also be used in the alarm expression.</i>

Name	Optional or required	Description
source	Optional. The default is the name of the resource, which by default is the hostname.	<p>The source of the samplevalue. The source can be set to a static name, but also to any valid Perl eval() expression that refers to the Perl variable \$resource and/or the Perl special variables \$1, \$2 etc. For example, if the source should be set to the resource name you use</p> <pre>source = \$resource</pre> <p>This is also the default. If resources have not been defined \$resource will be set to the hostname.</p> <p>If the source should be set to the Perl special variable \$1 you use:</p> <pre>source = \$1</pre> <p>If the source should be set to the concatenation of the Perl special variables \$1 and \$2 you use:</p> <pre>source = \$1 . '_' \$2</pre> <p>If you do not specify the CI type, the CI name and a metric ID (see below) there is quite some freedom in choosing the source and target. Let's take data collected from stores as an example. You can use say QOS_STORE_METRIC as the qos_definition, the name of the store as the source and total revenue, web store revenue etc. as the targets, but you could also use total revenue, web store revenue as the source and the name of the stores as the targets. However, if you want to specify a CI type, a CI name and a metric ID for the QoS the source HAS to be an IP-addressable device (the robot or a discovered network device), so using the name of the store as the source will not work. You have to define QOS_TOTAL_REVENUE, QOS_WEB_STORE_REVENUE as the qos_definitions, use \$ENV{'NIM_ROBOT_NAME'} as the source and the name of the stores as the targets in that case.</p> <p>NOTE THAT THE ALIAS-NAME WILL BE SUBSTITUTED FOR \$resource IN THESE EXPRESSIONS IF YOU SPECIFIED AN ALIAS.</p>

Name	Optional or required	Description
target	Required.	<p>The name of the target. The target can be set to a static name, but also to any valid Perl eval() expression that refers to the Perl variable \$resource and/or the Perl special variables \$1, \$2 etc. For example if the target should be set to the resource name you use</p> <pre>target = \$resource</pre> <p>If the target should be set to the Perl special variable \$1 you use:</p> <pre>target = \$1</pre> <p>If the target should be set to the concatenation of the Perl special variables \$1 and \$2 you use:</p> <pre>target = \$1 . '_' . \$2</pre> <p>If you do not specify the CI type, the CI name and a metric ID (see below) there is quite some freedom in choosing the source and target. Let's take data collected from stores as an example. You can use say QOS_STORE_METRIC as the qos_definition, the name of the store as the source and total revenue, web store revenue etc. as the targets, but you could also use total revenue, web store revenue as the source and the name of the stores as the targets. However, if you want to specify a CI type, a CI name and a metric ID for the QoS the source HAS to be an IP-addressable device (the robot or a discovered network device), so using the name of the store as the source will not work. You have to define QOS_TOTAL_REVENUE, QOS_WEB_STORE_REVENUE as the qos_definitions, use \$ENV{'NIM_ROBOT_NAME'} as the source and the name of the stores as the targets in that case.</p> <p>NOTE THAT THE ALIAS-NAME WILL BE SUBSTITUTED FOR \$resource IN THESE EXPRESSIONS IF YOU SPECIFIED AN ALIAS.</p>
alarm_eval	Optional. The default is undefined.	<p>A comma-separated list of any valid Perl eval() expression that refers to the Perl variable \$v to determine if an alarm has to be sent. The Perl variable \$v will be set to the samplevalue.</p> <p>Optionally the expression can be prefixed by the alert level using any of the keywords INFO:, WARNING:, MINOR:, MAJOR: or CRITICAL: For example if you want a warning alarm to be sent if the samplevalue is between 1 and 2 and a critical alarm if the value is greater than 3 you use</p> <pre>alarm_eval = WARNING:(\$v > 1) and (\$v < 2),CRITICAL:(\$v > 3)</pre>

Name	Optional or required	Description																		
alarm_description	Optional. The default is the target <u>followed by the alarm expression where \$v has been set to the samplevalue:</u>	<u>Any valid Perl eval() expression that refers to the Perl variable \$resource and/or the Perl special variables \$1, \$2</u> If defined the alarm will consist of the alarm_description followed by the alarm_eval expression where the and/or the Perl variable \$v. The Perl variable \$v will have been replaced <u>set by to the actual samplevalue.</u> The source of the alarm will be the resource-name. <i>Note that the alias-name will be used for the source of the alarm if you specified an alias.</i>																		
<u>suppression_key</u>	Optional. The default is a combination of <u>the command, the regular expression, the match group and the alarm expression.</u>	<u>Any valid Perl eval() expression that refers to the Perl variable \$resource and/or the Perl special variables \$1, \$2 etc. The evaluated expression will be used as the suppression key for the alarm.</u>																		
ci_type	Optional.	<p>A CI type specified as a sequence of numbers separated by dots as found in the cm_configuration_item_definition table. You can check the contents of this table using a SQL editor or the supplied cm_configuration_item_definition.csv file for some inspiration. If you cannot find a suitable ci_type you can add your own by extending the private CI section starting with the number '9' using a SQL editor.. The following screenshot shows the addition of the 9.3 Private.Store entry:</p>  <table border="1"> <tbody> <tr><td>9</td><td>NULL</td><td>Private</td></tr> <tr><td>9.1</td><td>9</td><td>Private.Device</td></tr> <tr><td>9.1.1</td><td>9.1</td><td>Private.Device.IP_Device</td></tr> <tr><td>9.2</td><td>9</td><td>Private.Network</td></tr> <tr><td>9.2.1</td><td>9.2</td><td>Private.Network.Data</td></tr> <tr><td>9.3</td><td>9</td><td>Private.Store</td></tr> </tbody> </table>	9	NULL	Private	9.1	9	Private.Device	9.1.1	9.1	Private.Device.IP_Device	9.2	9	Private.Network	9.2.1	9.2	Private.Network.Data	9.3	9	Private.Store
9	NULL	Private																		
9.1	9	Private.Device																		
9.1.1	9.1	Private.Device.IP_Device																		
9.2	9	Private.Network																		
9.2.1	9.2	Private.Network.Data																		
9.3	9	Private.Store																		
ci_name	Optional.	The name of the CI. It can be set to a static value, but also to any valid Perl eval() expression that refers to the Perl variable \$resource and/or the Perl special variables \$1, \$2 etc. For example if the ci_name should be set to the resource name you use ci_name = \$resource.																		

Name	Optional or required	Description
met_id	Optional.	<p>A metric ID specified as the number behind the colon of the met_type as found in the the cm_configuration_item_metric_definition table. You can check the contents of this table using a SQL editor or the supplied cm_configuration_item_metric_definition.csv file for some inspiration. If you cannot find a suitable met_type you can add your own by extended the private CI section starting with the number '9' using a SQL editor. The following screenshot shows the addition of a Revenue entry corresponding to the Private.Store entry:</p>  <p>You only have to specify the number behind the colon, the probe will join the ci_type and the met_id into the full met_type.</p>

The supplied configuration file gives examples on how to monitor iostat locally, uptime for systems in a USM group remotely, disk space for a couple of Unix systems remotely and how to retrieve the native read- and write-latencies of Netapp volumes using the Netapp rsh interface.

Example

The following explains the section in the supplied configuration file that monitors disk space for a couple of Unix systems remotely. The command we are going to use is df and we are going to use ssh to execute it remotely. The typical output of this command is:

```
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/sdd5        66053884  15681100   47017292  26% /
/dev/sdd6        66053884  42532012   20166380  68% /tmp2
/dev/sdd7        697102016 430678648   231012580  66% /tmp4
/dev/sdg1       1922859824 951893716   873290436  53% /tmp6
```

The output of df consists of five columns: the device-name, the total disk space in 1K blocks, the used disk space in 1K blocks, the available disk space in 1K blocks and the used disk space as a percentage. We want to use the device-name as the target of the QoS and we want to generate QoS's and/or alarms on the number of Used blocks, the Available disk space as a percentage and the used disk space as a percentage.

First we make sure that on the system that will run the probe it is possible to execute the df command remotely (if you don't know how to do this Google 'how to configure ssh to execute a command remotely without a password'):

```
[root@fedora191int129 cmd]# ssh root@172.16.4.125 df
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/sda2        74311880   4728916   65808144   7% /
tmpfs            961336         0     961336    0% /dev/shm
Dropbox          66053884  45887504   20166380  70% /media/sf_Dropbox
```

cmd 2.00

Next we define the remote systems and the command we want to use:

```
resources = root@172.16.4.125:centos64int125,freebsd90int136
cmd = ssh $resource df
```

The probe will substitute `root@172.16.4.125` and `freebsd90int136` for the Perl variable `$resource`. As we do not want to use `root@172.16.4.125` as the target name we use the alias `centos64int125`.

We only need one watcher to extract the columns from the `df` output. The required Perl regular expression is:

```
regexp = (\S+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)%
```

The probe provides a Windows tool `conf_cmd.exe` that you can use to validate the regular expression. You paste the output of the command in the top Memo, you construct your Regexp in the middle Edit and press <Enter> to evaluate it. The matches (if any) will be shown in the bottom StringGrid:

The screenshot shows a Windows application window titled "MainForm". It contains three main sections:

- Top Memo:** Displays the output of the `df` command. The text is as follows:

```
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/sda2        31366044    10034104   19715556   34% /
none              4              0          4    0% /sys/fs/cgroup
udev            16412276      4    16412272    1% /dev
tmpfs           3285228      1336    3283892    1% /run
none             5120          0      5120      0% /run/lock
none            16426132     908    16425224    1% /run/shm
none            102400        80     102320     1% /run/user
/dev/sda1        497696       3428    494268     1% /boot/efi
datastore       132822144     128    132822016   1% /datastore
datastore/tmp2  178098176   45276160  132822016  26% /tmp2
datastore/tmp4  377994624   245172608 132822016  65% /tmp4
```
- Middle Edit:** Contains a text box with the regular expression `(\S+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)%` and a red instruction: "Construct your RegEx here and press <Enter> to evaluate it."
- Bottom StringGrid:** A table showing the matches extracted from the `df` output. The matches are organized into columns labeled \$1 through \$9. The data is as follows:

	\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9
	The matches will be shown here								
	/dev/sda2	31366044	10034104	19715556	34				
	none	4	0	4	0				
	udev	16412276	4	16412272	1				
	tmpfs	3285228	1336	3283892	1				
	none	5120	0	5120	0				
	none	16426132	908	16425224	1				
	none	102400	80	102320	1				
	/dev/sda1	497696	3428	494268	1				
	datastore	132822144	128	132822016	1				

This will extract the Filesystem-name as the Perl special variable \$1, the total disk space in 1K blocks as the Perl special variable \$2, the Used disk space in 1K blocks as the Perl special variable \$3, the Available disk space in 1K blocks as the Perl special variable \$4 and the Use% disk space as the Perl special variable \$5. As we want to report on the number of Used blocks, the number of Available disk space as a percentage and the used disk space as a percentage we define matches sub-sections for each of the Perl special variables \$2, \$3 and \$4:

```
<2>
target = $1
qos_definition = DF_USED:QOS_APPLICATION:Used blocks:number:nr
alarm_eval = WARNING:$v > 0
alarm_description = used blocks
</2>
```

This sub-section will generate a QoS message for the Used disk space in 1K blocks with the name of the device as the target and a WARNING alarm if the number of 1K blocks is greater than 0 (clearly a threshold value of 0 is useful for testing and demonstration purposes only).

```
<3>
target = $1
qos_definition = DF_AVAILABLE:QOS_APPLICATION:Available:Percentage:%
samplevalue = int($4 * 100.0 / $2 + 0.5)
alarm_eval = WARNING:$v > 0
alarm_description = available space
</3>
```

This sub-section will generate a QoS message for the Available disk space as a percentage with the name of the device as the target and a WARNING alarm if the Available disk space as a percentage is greater than 0 (clearly a threshold value of 0 is useful for testing and demonstration purposes only). As the Available disk space as a percentage is not directly available in the output of the df command we have to calculate it from the Total disk space in 1K blocks (Perl special variable \$2) and the Available disk space in 1K blocks (Perl special variable \$4) using the following Perl expression:

```
samplevalue = int($4 * 100.0 / $2 + 0.5)
```

The value of this expression will also be used in the alarm expression.

```
<4>
target = $1
qos_definition = DF_USE:QOS_APPLICATION:Use:Percentage:%
alarm_eval = WARNING:$v > 0
alarm_description = use
</4>
```

This sub-section will generate a QoS message for the Used disk space as a percentage with the name of the device as the target and a WARNING alarm if the Used disk space as a percentage is greater than 0 (clearly a threshold value of 0 is useful for testing and demonstration purposes only).