



EMC® Watch4net®

Version 6.3u2

SNMP Collector Administration Guide

P/N: pending

Rev : 43341

EMC²

Copyright ©2009 - 2014 EMC Corporation. All rights reserved. Published in the USA.

Published April, 2014

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC², EMC, and the EMC logo are registered trademarks or trademarks of EMC Corporation in the United States and other countries. All other trademarks used herein are the property of their respective owners.

For the most up-to-date regulatory document for your product line, go to EMC Online Support (<https://support.emc.com>).

Contents

Preface	v
Document Organization	vi
Common Abbreviations and Acronyms	vii
Where to get help	ix
1 Functional overview	1
2 SNMP Collector configuration architecture	3
3 General Installation and configuration	7
4 SNMP Collector functional configuration	9
1 Editing the slave-snmp-poller.xml file	9
2 Configuring the SNMP Collector's runtime parameters	11
3 SNMP Collector network configuration	14
4 SNMP Collector polling group configuration	16
5 Polling distribution configuration	21
1 The snmp-slave-poller configuration	23
2 SNMP agent configuration	24
3 SNMP agents grouping	26
4 Multiple SNMP agents on the same IP address configuration	27
6 Basic collection configuration	29
1 Single oids polling configuration	31
2 Simple table polling configuration	33
2.1 <i>snmp-table</i> element attributes	33
3 Simple table column polling	36
4 <i>snmp-column</i> attributes	38
5 Simple filtering on table columns	39

6 AND OR column filtering conditions	42
7 <i>snmp-filter</i> attributes	44
8 SNMP properties configuration	45
9 MIB files specification	46
7 Translations configuration	47
8 Advanced collection configuration	49
1 Optional <i>snmp-table</i> parameters	51
2 Optional <i>snmp-table</i> nested elements	54
2.1 <i>snmp-reference-table</i> element	54
2.2 The <i>inner-join-on</i> element	56
3 Optional <i>snmp-filter</i> attributes and nested elements	57
3.1 Optional <i>snmp-filter</i> attributes	57
3.2 The <i>outer-join-on</i> element	58
4 SNMP filtering limitations	60
Appendix	61
A External commands	61
1 The <i>add-snmp-agents-list.sh</i> command	61
2 The <i>extract-from-snmp-polling-distribution.sh</i> command	67
3 The <i>send-snmp-command.sh</i> command	68
4 The <i>snmp-collector-ws-cli.sh</i> command	69
B Raw value properties	71
C Troubleshooting and other tips	73
1 Setting the polling session timeouts	73
2 Using the disk cache for polled <i>snmp-reference-tables</i>	74
3 Changing batch size when polling single oids	74
D Selective Table Row Polling Example	77
E Semi Selective Table Row Polling Example	81
F IPv6 support	87

G	64-bit counter wrap around support	89
H	SNMP Reachability polling	91
I	SNMP packet decoding logging	93

Preface

Document Organization

1. Functional overview	This section provides a quick functional overview of the features provided by the SNMP collector
2. SNMP Collector configuration architecture	This section provides an architectural view of the configuration files used with the SNMP collector
3. General Installation and configuration	This section provides general information on configuring the SNMP collector component.
4. SNMP Collector functional configuration	This section provides information about the configuration of the SNMP collector itself.
5. Polling distribution configuration	This section provides information on the configuration of the SNMP agents that are polled by the SNMP collector.
6. Basic collection configuration	This section describes the basic configuration of the SNMP oids and tables collection.
7. Translations configuration	This section describes the configuration of the translation of SNMP enum values to numeric values.
8. Advanced collection configuration	This section provides further information and examples of SNMP tables collection.

Common Abbreviations and Acronyms

<i>DCA</i>	Data Collecting Adapter
<i>DCF</i>	Data Collecting Filter
<i>DCC</i>	Data Collecting Connector
<i>CI</i>	Control Interface
<i>SC</i>	Storage Connector

Where to get help

EMC support, product, and licensing information can be obtained as follows:

PRODUCT INFORMATION - For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

<https://support.emc.com>

TECHNICAL SUPPORT - Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

Functional overview

Watch4Net's SNMP Collector is an APG DCA used to perform queries on any SNMP enabled agent. Most MIB files are supported, while full support for polling single OIDs and Tables is provided. Polling can be fully distributed, as per every APG collector, in order to decrease the CPU and network load on the hosts.

Contrary to many closed polling solutions, Watch4Net's SNMP Collector is fully configurable: any and every OIDs and Tables can be polled. This includes tables that have non-accessible indices, multiple index columns and even undefined index lengths. It also provides ways to filter out values from a table in order to keep only the relevant ones so that storage space can be kept to a minimum.

Table joins are also provided so that it is possible to collect all properties from other related tables. This feature is especially useful when polling complex MIBs, such as CISCO's Service Assurance Agent, Service Class Based QOS and Server Load Balancing MIB.

All of those advanced polling features allow users to fully customize their SNMP data collection to suit their requirements and produce complete and complex reports via APG's frontend.

Polling performance is also completely configurable, as the administrator may specify the number of concurrent connections and handling threads to optimize polling time or reduce the CPU load on the running hardware. Other network options, such as address binding, and remote communication command port and address are also provided.

Configuration and deployment is easy and flexible, as it allows combination of any OIDs and agents to be polled at any frequency. Polling periods are not hardcoded and may be modified to any suitable values, depending upon the metrics' criticality and usage. SNMP agents may be specified individually or by applying a regular expression on their IP address (or hostname, if defined). Several agents may be queried on a single host as they are treated as separate agents.

Finally, all properties usually reported by APG's SMARTS collector are provided for every raw value produced by this collector. This allows seamless integration with already existing APG reports based on SMARTS data.

2

SNMP Collector configuration architecture

As opposed to other simple DCAs, several XML files are required to configure the backend-collector-dca-snmp (SNMP Collector) in order to make it run properly. These files may be classified in 3 categories:

1. DCA configuration file:

- **collecting.xml**: specifies the file path for `snmpcollector.xml`
- **snmpcollector.xml**: specifies the file paths

2. SNMP collecting configuration files:

- **slave-snmp-poller.xml**: associates *SNMP Agents* to *SnmpMasks* in *PollingGroups*.
- **snmp-polling-distribution.xml**: contains the *SNMP Agents* to be polled by the SNMP Collector.
- **snmp-masks.xml**: defines the *SNMP* collecting.
- **translations.xml**: defines numerical translations for polled text values.

3. MIB files:

- Minimally, the RFC-1213 MIB file will be used by the SNMP Collector.
- Other MIB files used for *SNMP* polling.

The DCA configuration files are used to configure the collector's integration in the collecting component architecture while the *SNMP* collecting configuration files describe the collector's behaviour.

Figure 1 describes the relations between those files.

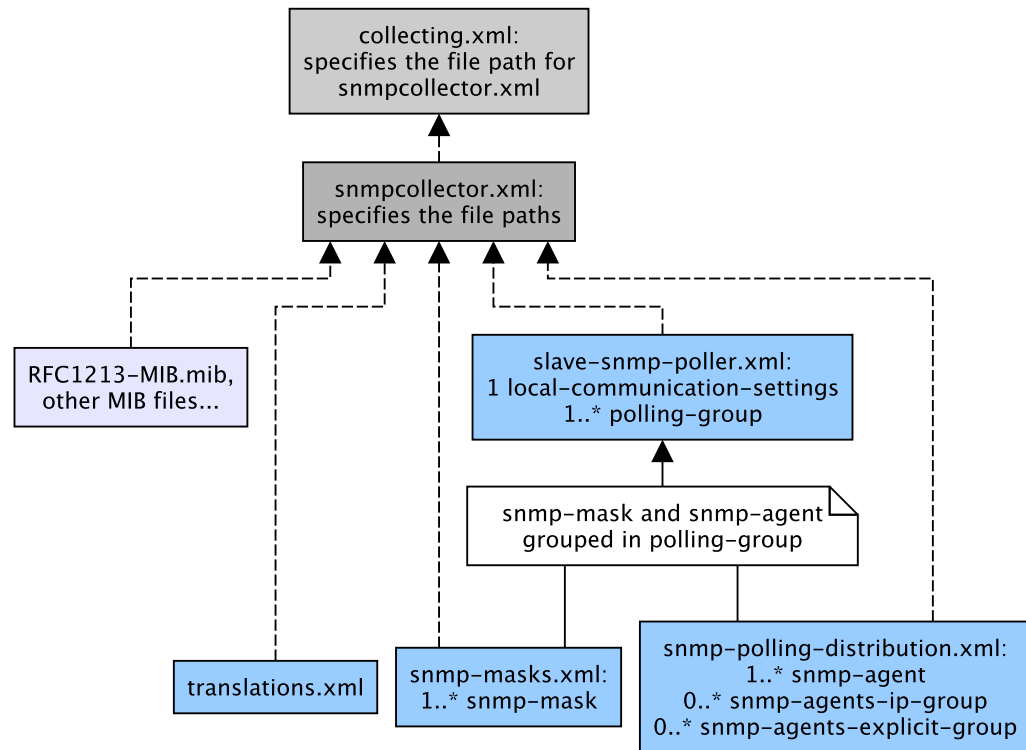


Figure 1: Configuration files relationships

3

General Installation and configuration

The SNMP Collector module should already be installed. If not, please use the module manager to do this. The SNMP Collector module name is snmp-collector.

Like every other APG backend collecting component, the SNMP Collector must be added to the `collecting.xml` file. This is done as follows:

```
<collectors>
...
<collector name="SnmpCollector" next="Backend" config="SNMP-Collector/<Instance>/conf/
snmpcollector.xml" />
...
</collectors>
```

The config file, usually named `snmpcollector.xml`, contains the path and names of the XML files required to configure the SNMP Collector.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE config SYSTEM "snmpcollector.dtd" --> -->
<config>
  <slave-snmp-poller-file>conf/slave-snmp-poller.xml</slave-snmp-poller-file>
  <snmp-masks-file>conf/snmp-masks.xml</snmp-masks-file>
  <snmp-polling-distribution-file>conf/snmp-polling-distribution.xml</snmp-polling-
distribution-file>
  <translations-file>conf/translations.xml</translations-file>
  <mib-files-directory>mibs</mib-files-directory>
</config>
```

We strongly recommend that those file names and path be kept unchanged unless several instances of the SNMP Collector are defined in the `collecting.xml` XML file.

Moreover, please note that the basic configuration provided with the SNMP Collector is inert, in the sense that the SNMP Collector will launch along with the other configured collectors but it will not collect anything until the polling distribution (`snmp-polling-distribution.xml`) and *SNMP* collecting (`snmp-masks.xml`) have been configured.

4

SNMP Collector functional configuration

1 Editing the `slave-snmp-poller.xml` file

The first thing to do when configuring the SNMP Collector is to edit the `slave-snmp-poller.xml` file. This XML file describes how the SNMP Collector will interact with its host networking hardware.

The following file fragment describes this in further detail:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "slave-snmp-poller.dtd">
<slave-snmp-poller name="slave1" pollerPrefix="thisPollerOnThatNetwork" groupName="group3600"
oidsThreads="50" tablesThreads="50">
  <local-communication-settings hostIpAddress="localhost" communicationPort="2008"
communicationInterfaceIpAddress="10.128.1.28" maxConcurrentSessions="48">
    <out-interface-ip-addresses>
      <out-interface-ip-address>10.128.9.29</out-interface-ip-address>
    </out-interface-ip-addresses>
    <accept-commands-from-list>
      <accept-commands-from>10.128.1.25</accept-commands-from>
    </accept-commands-from-list>
  </local-communication-settings>
  <polling-group ...>
    ...
  </polling-group ...>
```

Note: It is strongly recommended to use an XML editor which supports DTD validation in order to edit any of the configuration files described in this document. However, please note that the *DOCTYPE* tag *MUST BE COMMENTED OUT* of every file mentioned in this document prior to starting the SNMP Collector.

2 Configuring the SNMP Collector's runtime parameters

The first configuration line of the `slave-snmp-poller.xml` file describes the SNMP Collector's runtime parameters:

```
<slave-snmp-poller name="slave1" pollerPrefix="thisPollerOnThatNetwork"
  groupName="group3600" oidsThreads="100" tablesThreads="100">
```

- **name:** This attribute identifies the SNMP Collector. It is also used to identify the list of *SNMP* agents to use in the `snmp-polling-distribution` file. This name must be unique among all used SNMP Collectors.
- **pollerPrefix:** This attribute contains the value for source property. All raw values generated by this collector will have their source property set to this value.
- **usePollerPrefixInVariable:** Setting this to "true" adds the poller prefix to the generated raw values variables. This has the effect of effectively binding the polled raw values from a SNMP Agent to its collector's pollerPrefix. This can be beneficial when several collectors are used to poll different networks with overlapping IP addresses.

However, this behavior causes problems when using a distributed polling configuration, where the SNMP Agent may be polled by any collector. There are two solutions to this problem:

- Use the same pollerPrefix on all `SnmpCollectors` and set `usePollerPrefixInVariable="true"`. The poller prefix will still be in the raw value variable, thus binding this value (and SNMP Agent) to a specific group of SNMP Collectors which share the same pollerPrefix. This can be used when polling a network with several SNMP Collectors. A typical use case for this is when several networks, belonging to several clients, are polled by more than two SNMP Collector per network. Those individual networks could belong to distinct clients, causing IP Address overlapping. In this specific case, the poller prefix would in fact represent the network name (or the name of the client who actually uses or own this network).
- Use different pollerPrefix on the SNMP Collectors and set `usePollerPrefixInVariable="false"`. This may be used when several SNMP Collectors are used to poll a network without IP Address overlapping. In this case, the polling is distributed among several collectors and a specific SNMP Agent may be polled by any of them at any one time. No effect will be seen on the polled raw value variables when the SNMP Agent switches poller as the pollerPrefix is not part of the variable.

Note: The `usePollerPrefixInVariable` attribute should be left to "true" when upgrading existing instal-

lation in order to avoid doubling the number of variables after the upgrade. Please contact Watch4Net for an upgrade procedure in case of doubt.

- **groupName:** This attribute contains the retention group name. The valid group names are defined in the config.xml file. All raw values generated by this collector will have their group property set to this value, unless it is overridden in the polling group where the raw value comes from.
- **oidsThreads:** The maximum number of threads used to poll single oids.
- **tablesThreads:** The maximum number of threads used to poll tables.
- **alwaysPushProperties:** Set this to “true” to send the raw values properties on every polling cycle. This is the default behaviour. Setting this to false will remove the properties from raw values, except when they are being refreshed.
- **oidAsPropertyInRawValues:** Setting this to “true” adds the polled oid to every raw value output by the SNMP Collector. This may be useful for debugging purposes. The default behaviour is not to add the oid to the raw value.

One must note the importance of setting the Threads attributes with adequate values as they will directly impact the *SNMP* collector's performance. Providing values that are too small will cause contention at the thread level, therefore dramatically slowing down the polling process, while providing an excessively large value may have adverse consequences on system performance due to memory consumption and thread scheduling.

The following are guidelines to set those Threads attributes values: For a polling group:

- 1 oidThread per (*SNMP Agent* times *snmp-oids*) element.
- 1 tableThread per (*SNMP Agent* times *snmp-table*) element.

Basically, one should try to find out which of the polling-group is the most demanding for every Thread type (oidThread and tableThread) and assign the maximum calculated value for each Thread type to the Thread attribute. Experience will allow you to reduce those numbers in order to find the sweet spot between system load and polling time for every polling groups. These times are available in the collecting*.log files.

Also worth mentioning is that polling groups should be relatively similar, in terms of gathered data volume size, and that many smaller polling groups running in parallel may be faster than one huge group, which takes a long time to poll. One must keep in mind that single OIDs are polled on every agent in a polling group before tables are polled. Therefore, if polling OIDs takes a long time (say 90 seconds on a 240 polling period), it may be better to build a polling group which contains only single OIDs and another group with tables only, so that the polling group does not take more than 240 seconds to complete. When a polling group takes more time to complete than its polling period, its next poll is skipped, thus creating holes in the data, therefore it is really important that polling groups should be configured as efficiently as possible.

3 SNMP Collector network configuration

The second configuration block in the `slave-snmp-poller.xml` file defines the *SNMP* collector's networking parameters:

```
<local-communication-settings hostIpAddress="localhost" communicationPort="2008"
  communicationInterfaceIpAddress="10.128.1.28"
  maxConcurrentSessions="300">
  <out-interface-ip-addresses>
    <out-interface-ip-address>10.128.9.29</out-interface-ip-address>
  </out-interface-ip-addresses>
  <accept-commands-from-list>
    <accept-commands-from>10.128.1.25</accept-commands-from>
  </accept-commands-from-list>
</local-communication-settings>
```

The `local-communication-setting` element has the following attributes:

- `hostIpAddress`: This specifies the collector's host ip address.
- `communicationPort`: This specifies the collector's communication command port. This port is used to receive *SNMP* requests as well as control and configuration commands from Watch4Net's mib browser.
- `communicationInterfaceIpAddress`: This specifies the ip address of the interface on which the communication command port is bound. This attribute is optional as all of the host's defined interfaces will be used if it is not specified.
- `maxConcurrentSession`: This specifies the maximum number of concurrent sessions used for polling.

The `maxConcurrentSession` attribute has a very high impact on the collector's performance. Setting it too low will severely slow down polling performance while setting it too high may cause unexpected crashes as the system (O/S) may be unable to support a large number of connections. A good starting point for setting this value is the default, which is 98. Please check the `collecting-*.log` files to determine whether this is enough for the polling configuration. Increasing it will likely decrease polling times when numerous *snmp-agent* are polled. This setting value must be less than `oidsThreads` and `tablesThreads`.

The elements contained in the local-communication-settings element are both optional:

- `out-interface-ip-address`: Specifies which interface to use when polling the *SNMP* agents. Currently, only one interface may be specified here. All available interfaces will be used for polling if none is specified here.
- `accept-commands-from`: Specifies the ip addresses from which *SNMP* requests and control commands may be sent. Adding one of those elements will tell the *SNMP* collector to accept commands from the specified ip address, while rejecting commands coming from any other addresses. Commands coming from all ip addresses will be accepted if no *accept-commands-from* element is provided.

4 SNMP Collector polling group configuration

The remainder of the `slave-snmp-poller.xml` file is used to define the polling groups:

```
<polling-group name="AllEquipments" pollingPeriod="10"
tableDiscoveryTime="03:30:00" groupName="group1" enabled="true">
  <polling-masks>
    <polling-mask-name>AllEquipmentMask</polling-mask-name>
  </polling-masks>
  <polled-snmp-agents>
    <snmp-agent-ip-address>10.128.2.28:161</snmp-agent-ip-address>
    <snmp-agent-ip-address>10.128.7.2:161</snmp-agent-ip-address>
    <snmp-agent-ip-address>10.128.7.1:1690</snmp-agent-ip-address>
    <snmp-agent-ip-address>10.128.3.1:161</snmp-agent-ip-address>
    <snmp-agents-ip-group-name>Network</snmp-agents-ip-group-name>
    <snmp-agents-explicit-group-name>ExplicitGroup1</snmp-agents-explicit-group-name>
    <snmp-agents-explicit-group-name>ExplicitGroup2</snmp-agents-explicit-group-name>
  </polled-snmp-agents>
  <subgroups>
    <polling-group...>
      ...
    </polling-group>
  </subgroups>
</polling-group>
```

The *polling-group* element describes a polling group. Many polling groups may be defined in the *slave-snmp-poller.xml* file, the only limitation is that they *MUST* have a unique name, as it is the polling group's identifier. The *polling-group* element has the following attributes:

- **name:** The polling group's name. As stated before, this name must be unique as it is an identifier. The value of this attribute is used to set the pollgrp property of the raw values resulting from this polling group.
- **pollingPeriod:** The polling group's polling period, specified in seconds. This value indicate the time interval between each polling. 60 second is a minute, 240 seconds is 4 minutes and so on.
- **tableDiscoveryTime:** This attribute indicate the time at which the raw values' properties will be refreshed, as well as any reference table resident in the SNMP collector. Refreshing the raw value's properties may be slightly more demanding on the database side of the backend, therefore, it is recommended to set this value at some off-hour time.

Please note that this value is formatted in 24h time, 00:00:00 being midnight, 08:00 being 8:00am, 20:00 being 8:00pm and so on. Property refresh may also be prevented by using the

“onRefreshPropertiesCommandOnly” value, instead of a valid 24h time. In this case, properties will be refreshed on the agent only when a refresh command is sent to the collector.

- **groupName:** This attribute overrides the retention group name specified in the *slave-snmp-poller* element for this polling group and all of the ones defined under it.
- **uniformTimestamps:** When set to true, the polling will start at the next “round” period (0:04, 0:08... if polling period is set to 4 minutes) and all timestamps for the generated raw values will be the rounded to the polling cycle start time (0:04, 0:08...).
- **startAtDefinedPeriod:** When set to true, the polling will start at the next “round” period (0:04, 0:08... if polling period is set to 4 minutes). Raw value timestamps will be accurate.
- **enabled:** When set to false, the polling group will not run. True is the assumed default when not specified. This is a simple way to define polling groups but not use them.
- **startTimeOffset:** Setting this to a value other than 0 delays the polling group start time by 'n' seconds. This modifies the 'startAtDefinedPeriod' and 'uniformTimeStamps' behavior. This attribute can be used to spread the polling group start time through a polling period window.

Note: About the uniformTimestamps and startAtDefinedPeriod behaviour: These behaviours will function correctly when the polling period is set to the following values: 60 (1 min), 120 (2 min), 180 (3 min), 240 (4 min), 300 (5 min), 360 (6 min), 600 (10 min), 720 (12 min), 900 (15 min), 1200 (20 min), 1800 (30 min), 3600 (1 hr), 7200 (2 hr), 10800 (3 hr), 14400 (4 hr), 21600 (6hr), 28800 (8hr), 43200 (12 hr), 86400 (1 day), 172800 (2 day)...

For 1 day (2, 3, 4day) polling period: the polling will be performed at the time specified by the tableDiscoveryTime attribute.

In turn, the *polling-group* element contains other elements describes below:

- *polling-masks*: This element set contains *polling-mask-name* elements. At least one *polling-mask-name* element must be provided per polling-group. The *polling-mask-name* is used to locate the correct *snmp-mask* in the *snmp-masks.xml* file. The *snmp-mask* element contains information about the *SNMP* oids to collect in this polling group. See section 6, page 30 of this document for further details on *snmp-mask* elements.
- *polled-snmp-agents*: This element set contains at least one *snmp-agent-ip-address*, *snmp-agents-ip-group-name* or *snmp-agents-explicit-group-name* element. These elements actually indicate which *SNMP* agent to poll in this polling group. *SNMP* agent and group definitions are done in the *snmp-polling-distribution.xml* file, which is presented in section 5, subsection 2 and 3 of this document. Please note that the port number should be specified when providing *snmp-agent-ip-address* as several agents may be running on the host specified by the ip address. If none is specified, port 161 (:161 suffix) will be used and it might not be suitable for your purposes.
- *subgroups*: This optional element set contains nested polling group. This allows the creation of polling groups hierarchies. These hierarchies can be used to set up polling groups by equipment types, the most general type being at the top of the hierarchy, while more specific types will be at the bottom. The following example typifies this:

```

<polling-group name="AllEquipments" pollingPeriod="60" tableDiscoveryTime="03:30:00"
  groupName="grp3600">
  <polling-masks>
    <polling-mask-name>AllEquipmentMask</polling-mask-name>
  </polling-masks>
  <polled-snmp-agents>
    <snmp-agent-ip-address>10.128.9.9</snmp-agent-ip-address>
    <snmp-agents-explicit-group-name>Network2</snmp-agents-explicit-group-name>
  </polled-snmp-agents>
  <subgroups>
    <polling-group name="Switches" pollingPeriod="30" tableDiscoveryTime="03:15:00"
      groupName="grp1800">
      <polling-masks>
        <polling-mask-name>SwitchMask</polling-mask-name>
      </polling-masks>
      <polled-snmp-agents>
        <snmp-agent-ip-address>10.128.9.8:161</snmp-agent-ip-address>
      </polled-snmp-agents>
      <subgroups>
        <polling-group name="CiscoSwitches" pollingPeriod="20" tableDiscoveryTime="03:00:00">
          <polling-masks>
            <polling-mask-name>CiscoSwitchMask</polling-mask-name>
          </polling-masks>
          <polled-snmp-agents>
            <snmp-agent-ip-address>10.128.9.1:161</snmp-agent-ip-address>
            <snmp-agents-explicit-group-name>Network3</snmp-agents-explicit-group-name>
          </polled-snmp-agents>
        </polling-group>
      </subgroups>
    </polling-group>
  </subgroups>
</polling-group>

```

In this case, all of the *SnmpAgents* defined under the "AllEquipments" polling group (10.128.9.9 and Network2) will be polled every 60 seconds, using the AllEquipmentMask. All *SNMP* agents defined in the subgroups under this group will also be polled with this mask. Therefore, 10.128.9.9, Network2, 10.128.9.8, 10.128.9.1, Network3, 10.128.9.20 and Network1 will be polled every 60 seconds, using the AllEquipmentMask as well. The retention group name for this polling group is "group3600".

Then, all equipments defined in the Switches polling group and below will be polled every 30 seconds, using the SwitchMask. The retention group name for this polling group is "grp1800".

Finally, all equipments defined in the CiscoSwitches polling group will be polled every 20 seconds, using the CiscoSwitchMask. The retention group name for this polling group is "grp1800", as it is defined in the polling group which contains this one. Please see the example slave-snmp-poller.xml file, which comes on the install CD, for further explanations on how the polling group hierarchy works.

5

Polling distribution configuration

The polling distribution configuration is in fact the list of *SNMP* agents which may be polled by the SNMP Collector. It also defines the SNMP agents groups which may be used, alongside the individual agents, in the polling groups defined in the *slave-snmp-poller.xml* XML file. The *snmp-polling-distribution.xml* XML file describes those agents and groups:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "snmp-polling-distribution.dtd">
<snmp-polling-distribution>
  <snmp-slave-poller name="slave1" enabled="true" host="10.128.2.28" port="2008"
    slaveSnmpPollerXmlFile="slave-snmp-poller.xml">
    <snmp-agent name="Switch1" ipAddress="10.128.9.1" timeout="3000" retries="1"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="123456" devtype="Switch"
      transportProtocol="udp" snmpPort="161" maxBulkSize="50" onFailTryWith="v1"/>
    <snmp-agent name="Switch2" ipAddress="10.128.9.2" timeout="3000" retries="2"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="123" devtype="Switch" maxBulkSize="50"
      onFailTryWith="v2c"/>
    <snmp-agent name="Switch3" ipAddress="10.128.9.3" timeout="3000" retries="2"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="123" devtype="Switch" maxBulkSize="50"
      onFailTryWith="doNotRetry"/>
  </snmp-slave-poller>
</snmp-polling-distribution>
```

```

<snmp-agent name="Switch4" ipAddress="10.128.9.4" timeout="3000" retries="2"
  snmpVersion="v2c" snmpCommunityOrUserV3Name="123" devtype="Switch" maxBulkSize="20"/>
<snmp-agent name="Switch7" ipAddress="10.128.9.7" timeout="3000" retries="2"
  snmpVersion="v2c" snmpCommunityOrUserV3Name="123" devtype="Switch" maxBulkSize="10"/>
<snmp-agent name="Switch8" ipAddress="10.128.9.8" timeout="3000" retries="2"
  snmpVersion="v2c" snmpCommunityOrUserV3Name="123" devtype="Switch" maxBulkSize="30"/>
<snmp-agent name="Switch9" ipAddress="10.128.9.9" timeout="3000" retries="2"
  snmpVersion="v2c" snmpCommunityOrUserV3Name="123" devtype="Switch" maxBulkSize="20"/>
<snmp-agent name="Switch10" ipAddress="10.128.10.10" timeout="3000" retries="2"
  snmpVersion="v2c" snmpCommunityOrUserV3Name="123456" devtype="Switch"/>
<snmp-agent name="Switch11" ipAddress="10.128.10.11" timeout="3000" retries="2"
  snmpVersion="v2c" snmpCommunityOrUserV3Name="123456" devtype="Switch"
  snmpPort="161"/>
<snmp-agent name="Switch11" ipAddress="10.128.10.11" timeout="3000" retries="2"
  snmpVersion="v1" snmpCommunityOrUserV3Name="123456" devtype="Switch"
  snmpPort="1691" v1TableRowPolling="true"/>
<snmp-agent name="Host1" ipAddress="10.128.11.1" timeout="1000" retries="1"
  snmpVersion="v3" snmpCommunityOrUserV3Name="123456" devtype="Host"
  snmpV3AuthenticationPassword="12345" snmpV3AuthenticationProtocol="md5"
  snmpV3PrivacyPassword="123" snmpV3PrivacyProtocol="aes"/>
<snmp-agents-ip-group name="Network1" ipAddressMask="10.128.10.*"/>
<snmp-agents-explicit-group name="Network2">
  <agent-ip-address>10.128.9.2:161</agent-ip-address>
  <agent-ip-address>10.128.9.3:161</agent-ip-address>
</snmp-agents-explicit-group>
<snmp-agents-explicit-group name="Network3">
  <agent-ip-address>10.128.9.6:161</agent-ip-address>
  <agent-ip-address>10.128.9.8:161</agent-ip-address>
</snmp-agents-explicit-group>
<snmp-agents-explicit-group name="Network4">
  <agent-ip-address>10.128.10.11:161</agent-ip-address>
  <agent-ip-address>10.128.10.11:1691</agent-ip-address>
</snmp-agents-explicit-group>
</snmp-slave-poller>
<snmp-slave-poller name="slave2" enabled="true" host="10.128.2.28" port="2008"
  slaveSnmpPollerXmlFile="slave-snmp-poller.xml">
  <snmp-agent name="Switch100" ipAddress="10.128.9.100" timeout="3000" retries="1"
    snmpVersion="v2c" snmpCommunityOrUserV3Name="123456" devtype="Switch" \
    transportProtocol="udp" snmpPort="161"/>
</snmp-slave-poller>
</snmp-polling-distribution>

```

1 The snmp-slave-poller configuration

The following fragment describes the polling distribution for the SNMP collector named "slave1":

```
<snmp-slave-poller name="slave1" enabled="true" host="10.128.2.28" port="2008"
slaveSnmpPollerXmlFile="slave-snmp-poller.xml">
```

Several *snmp-slave-poller* elements may be defined in the *snmp-polling-distribution.xml* file. This allows several SNMP Collectors to use this file for *SNMP* agent configuration. The *snmp-slave-poller* element requires the following attributes:

- **name**: The value of this attribute must have the same value as the name attribute in the *slave-snmp-poller* element located in the *slave-snmp-poller.xml* file. This is the name of the list of agents that are accessible by the same named SNMP collector. Please note that this is an identifier and that it must be unique.
- **enabled**: Reserved for future use. Leave this value to true for the time being.
- **host**: The ip address of the host on which the SNMP collector is running. It must be set to a valid address.
- **port**: This specifies the SNMP Collector's communication command port on it's host. This port is used to receive SNMP requests as well as control and configuration commands from Watch4Net's mib browser. This must be set to a valid port number, which is the same as the one specified in the *local-communication-settings's communicationPort* attribute of the *slave-snmp-poller.xml* configuration file.
- **slaveSnmpPollerXmlFile**: This specifies the name of the *slave-snmp-poller.xml* XML file. It is reserved for future use but must be set to a correct value.

2 SNMP agent configuration

At least one SNMP agent must be defined in the *snmp-slave-poller* element:

```
<snmp-agent name="Host1" ipAddress="10.128.11.1" timeout="1000" retries="1"
  snmpVersion="v3" snmpCommunityOrUserV3Name="123456" devtype="Host"
  snmpV3AuthenticationPassword="12345" snmpV3AuthenticationProtocol="md5"
  snmpV3PrivacyPassword="123456" snmpV3PrivacyProtocol="aes"/>
```

The *snmp-agent* element have the following attributes:

- **ipAddress**: This is the *ipAddress* of the SNMP agent. The combination of this *ipAddress* *snmpPort* must be unique in the *snmp-slave-poller* element as it identifies a specific *snmp* agent running on a specific host. It is required as this is address which will be used when polling the SNMP agent. This will set the raw value's *ip* property.
- **devType**: This attribute must be set. It indicates the type of device which is described by the SNMP agent. All of the raw values polled on the SNMP agent will have their *devtype* property set to this value.
- **doNoUseGetBulkRequests**: Applies to SNMP v2c and v3 agents only: do not use *getBulk* request for polling this agent when set to true. Must be set to true or false.
- **name**: This attribute is required. It will set the device property for the raw values polled on this agent.
- **maxBulkSize**: This attribute sets the maximum number of *varbinds* that can be specified in a *snmpGetBulk* request sent to this *snmp-agent*. The default value is 50, which should work with most cases. However, some device *snmp* agents do not implement RFC3416 correctly and simply do not answer to requests which response cannot fit in a single pdu instead of sending the largest possible response, thus triggering timeouts during polling and loss of polled data. Please note that this attribute only applies to SNMP v2c and v3 agents. Also, lowering this value to a very small number, such as 1, will negatively impact polling performance.
- **onFailTryWith**: This attribute applies to table polling, using column-based methods. It tells the collector to retry a failed polling on a column, from the top of the column. The default behaviour does not perform this retry. However, it is possible to perform the retry by specifying the use of either SNMP v1 *getNext* requests (slow, but very reliable on most agents) or SNMP v2c *getBulk* requests (faster, but not supported on all agents). Possible values are: "doNotRetry", "v1", "v2c". Single SNMP oid polling is also affected by this setting, when the value is set to either "v1" or "v2c". But in this case, the retry will be performed by using atomic *get* requests (only 1 oid per request) instead of grouping the oids in the request. This gives the SNMP Agent the maximum number of chances to respond before finally timing out.

- **retries**: The number of retries when polling unresponsive SNMP agents. 0 means no retry.
- **snmpCommunityOrUserV3Name**: The community string (when using SNMP v1 or v2c) or the user name (when using SNMP v3). This attribute's value is required.
- **snmpPort**: This is the SNMP port to use when polling the SNMP agent. Port 161 is set by default when not specified.
- **snmpVersion**: the SNMP version to use when polling the agent. May be either v1, v2c or v3. Defaults to v2c when not specified.
- **snmpV3AuthenticationProtocol**: This must be set when using SNMP v3 only. It may be either md5 or sha. Usually required when polling SNMP V3 agents.
- **snmpV3AuthenticationPassword**: The authentication password, used only with SNMP v3. Optional: if not provided: noAuthNoPriv is used. auth mode is used if no privacy password is set. authPriv mode is used when both snmpV3AuthenticationPassword and snmpV3PrivacyPassword are set.
- **snmpV3PrivacyProtocol**: The SNMP v3 privacy protocol. May be set to aes or des. May be optional, depending upon the SNMP V3 agent's configuration.
- **snmpV3PrivacyPassword**: The SNMP v3 privacy password. Optional: when set, authPriv mode is used. When not: Auth mode or noAuthNoPriv mode is used, depending if snmpV3AuthenticationPassword is set or not.
- **snmpV3ContextName**: The SNMP v3 context name. The default context name is used if not specified or an empty string is provided.
- **useV3ContextNameInVariable**: If set to "true", the raw value's variable will use the context name as part of it.
- **snmpV3ContextEngineId**: The SNMP v3 context engine id. Will be used if specified. Format is an hexadecimal character string (i.e.: 414242 for "ABC") which conforms with RFC3411's snmpEngineID.
- **useV3useV3ContextEngineIdInVariable**: If set to "true", the raw value's variable will use the context engine ID as part of it.
- **timeout**: The SNMP timeout for the agent's response, stated in milliseconds.
- **transportProtocol**: This may be set to either udp or tcp. udp is set by default when not specified.
- **v1TableRowPolling**: This attribute affects the way SNMP V1 agents are polled. Setting this to false is the standard failsafe mode. Table polling will be column-based, while setting this to true enables a more efficient row-based polling model (not correctly supported by all SNMP V1 agents). This attribute is optional and defaults to false.

3 SNMP agents grouping

The SNMP agents defined in section 5, subsection 2 may be grouped in two fashions: either by applying a mask on their ip address or by grouping them explicitly. The following XML fragment from the *snmp-polling-distribution.xml* file shows how to do both:

```
<snmp-agents-ip-group name="Network1" ipAddressMask="10.128.10.*"/>
<snmp-agents-explicit-group name="Network2">
  <agent-ip-address>10.128.9.2:161</agent-ip-address>
  <agent-ip-address>10.128.9.3:161</agent-ip-address>
</snmp-agents-explicit-group>
```

The first group definition tells the SNMP Collector to create an agent group based on their ip addresses. In this case, all of the *SNMP* agents which have an address that matches the 10.128.10.* regular expression will belong to the Network1 snmp-agents-ip-group. Any regular expression may be used to create a group. Also of note, the match will be performed on the ip address, suffixed by :portNumber. Therefore, an ipAddressMask="10.128.10.2:1*" will match on agents defined on ip address 10.128.10.2, with port numbers starting with "1" (161, 1690, and any other agent defined on this ip address, with a port number starting with 1, will be part of the group).

The second group definition tells the SNMP Collector to create an agent group with the *SNMP* agents specified by their ip addresses and port number (www.xxx.yyy.zzz:portNumber).

Please note that, in both cases, the *snmp-agents* used to create the SNMP agent groups must be defined in the snmp-slave-poller where the agent group is defined. The agent group names must be unique as they are identifiers.

4 Multiple SNMP agents on the same IP address configuration

Configuring the polling of multiple SNMP agents on the same host (IP address) is possible, as shown in the example below:

```
<snmp-slave-poller name="slave1" enabled="true" host="10.128.2.28" port="2008"
  slaveSnmpPollerXmlFile="slave-snmp-poller.xml">
  <snmp-agent name="Switch11" ipAddress="10.128.10.11" timeout="3000" retries="2"
    snmpVersion="v2c" snmpCommunityOrUserV3Name="123456" devtype="Switch"
    snmpPort="161"/>
  <snmp-agent name="Switch11" ipAddress="10.128.10.11" timeout="3000" retries="2"
    snmpVersion="v2c" snmpCommunityOrUserV3Name="123456" devtype="Switch"
    snmpPort="1691" />
  <snmp-agents-ip-group name="Network5" ipAddressMask="10.128.10.11:.*"/>
  <snmp-agents-explicit-group name="Network4">
    <agent-ip-address>10.128.10.11:161</agent-ip-address>
    <agent-ip-address>10.128.10.11:1691</agent-ip-address>
  </snmp-agents-explicit-group>
</snmp-slave-poller>
```

In this case, the SNMP agent's key will be formed by its IP address and the SNMP port number it uses. In this case, there are two SNMP agents on the host at IP 10.128.10.11. One listens on port 161 and the other one listens on port 1691. These two agents may be part of a *snmp-agents-explicit-group*, such as the Network4 group shown above, or even a *snmp-agents-ip-group*, such as the Network5 group (as long as the regular expression used to define it matches the IPAddress:SnmpPortNumber pattern). Also, one must note that the IPAddress:SnmpPortNumber format must be used to specify those SNMP agents in the polling groups defined in the slave-snmp-poller.xml file, otherwise a port number 161 will be used by default.

6

Basic collection configuration

Basically, the SNMP Collector allows collecting the following entities:

- `Single` oids, such as `sysUpTime`, or any other single oid.
- `Tables`, such as the `ifTable`.

Moreover, any SNMP variable type may be polled, as long as a numeric value may be provided to generate the raw value. Therefore, it is possible to poll `Counters`, `Gauges`, `Timeticks`, `Integers` and even `OctetsStrings` (as long as a translation is provided to transform the polled value to a numeric value as `Integers` should be used as enum types. See [section 7](#) of this document for further information on value translation configuration).

The SNMP collecting process must be defined in the `snmp-masks.xml` XML file, for which a simple example of the structure is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "snmp-masks.dtd">
<!-- SNMP polling masks definitions. -->
<snmp-masks>
  <snmp-mask name="mask1" doNotEdit="true" version="myVersion1">
    <snmp-properties>
      <snmp-property .../>
    </snmp-properties>
    <snmp-oids>
      <snmp-oid .../>
      <snmp-oid .../>
    </snmp-oids>
    <snmp-tables>
      <snmp-table .../>
      <snmp-table .../>
    </snmp-tables>
    <mib-files>
      <mib-file-name>RFC1213-MIB.mib</mib-file-name>
      ...
    </mib-files>
  </snmp-mask>
  <snmp-mask name="mask2">
    ...
  </snmp-mask>
</snmp-masks>
```

The structure itself is quite simple as the *snmp-masks* element must contain at least one named *snmp-mask* element. The name of the *snmp-mask* is the key used in the `snmp-slave-poller.xml`'s polling groups' *polling-mask-name* element.

Optional attributes can also be specified in the *snmp-masks* element:

- *doNotEdit*, which can be either true or false. This flag is used with the GUI configuration tool. When set to true, the GUI will not allow the editing of this *snmp-masks*.
- *version*, which is used for version management. Any string can be inserted in this attribute. It is up to you, or the *snmp-mask* editor, to keep it up to date.

1 Single oids polling configuration

The single oids polling configuration is very simple to accomplish. The XML fragment below shows how to do this:

```
<snmp-oids >
  <snmp-oid id="oid1" name="udpOutDatagramsRate" oid=".1.3.6.1.2.1.7.4.0"
    as="RATE" unit="datagrams/s"/>
  <snmp-oid id="oid2" name="udpInDatagrams" oid=".1.3.6.1.2.1.7.1.0"
    as="COUNTER" unit="datagrams"/>
  <snmp-oid id="oid3" name="snmpInTraps" oid=".1.3.6.1.2.1.11.19.0"
    as="DELTA" unit="traps"/>
  <snmp-oid id="oid4" name="snmpOutTraps" oid=".1.3.6.1.2.1.11.29.0"
    as="DELTA" unit="traps"/>
  <snmp-oid id="oid5" name="hrMemorySize" oid=".1.3.6.1.2.1.25.2.2.0"
    unit="traps" partType="Main_memory" part="Memory"/>
</snmp-oids>
```

There are no limit to the number of *snmp-oid* elements which may be present in the *snmp-oids* element. The following attributes must be provided:

- *id*: the id for this oid. This value must be unique among all *snmp-oid* and *snmp-table* elements present in the *snmp-masks.xml* file.
- *name*: the human readable name for this oid. i.e.: sysUpTime. It's value fills the name property value in the resulting rawValue.
- *oid*: the oid itself. i.e.: .1.3.6.1.2.1.1.3.0

The following attributes are optional: as, unit, part, partType.

- **as**: specifies how the polled values must be handled.

Possible values are:

- **COUNTER** or nothing: Do not apply any calculation to the polled value. Send it as is. This is the default behaviour and does not require to be specified.
 - **DELTA**: Calculate the difference between polled values and send this result as the polled value. $\text{PolledValue} = \text{PV2} - \text{PV1}$. Counter wrap around is handled, except on host restart, where the generated value will be negative.
 - **DELTAP**: computes the same value as the DELTA transformation. However, negative values for types that are not COUNTERs will be discarded.
 - **RATE**: Calculate the rate between polled values and send this result as the polled value. $\text{PolledValue} = (\text{PV2} - \text{PV1}) / (\text{T2} - \text{T1})$. Counter wrap around is handled, except on host restart, where the generated value will be negative.
 - **RATEP**: computes the same value as the RATE transformation. However, negative values for types that are not COUNTERs will be discarded.
- **unit**: specifies the unit to apply to the polled value. Does not involve any calculation. It's value fills the unit property in the resulting RawValue.
 - **part**: this specifies the [SMARTS] part property value. It's value fills the part property in the resulting RawValue.
 - **partType**: this specifies the SMARTS parttype property value. It's value fills the parttype property in the resulting RawValue. This is a hardcoded value.

2 Simple table polling configuration

SNMP table polling is one of the most powerful feature offered by the SNMP collector. Table polling is completely configurable, with very few limitations to what may be achieved. This power, however, comes at the price of complex configuration. This section will provide information on most of the basic table polling cases, while section 8 of this document covers more complex examples of what may be achieved.

2.1 *snmp-table* element attributes

Let's start with the basic *snmp-table* element attributes, which must be provided:

- **id**: the table's id, which must be unique among all *snmp-oid* and *snmp-table* elements.
- **name**: the table's name.
- **oid**: the table's base oid. i.e.: ifTable is .1.3.6.1.2.1.2.2
- **tableIndex**: the table's index columns. The form MUST BE AS FOLLOWS: ".1.1 .1.2" This indicates column 1 and 2 of the table.

The raw value's variable name will be made using the tableIndex values, if nothing else is available (*tableColNameOid* is not set, no *varNameGenOidSequence* specified in *snmp-column* element and no *usePropertyValueInVariableName*="true" is set in any *snmp-reference-table*).

When polling tables that have index columns whose values are not accessible (not readable), the user must specify a way to retrieve the index values from another polled column. Oids polled in a table always contain a suffix which identifies the row. It is the user's task to tell the snmp collector that it must extract this suffix to use it as the table's index. In order to do this, specify a non-existent or non-accessible oid in table index, as follows:

tableIndex=".1.100" generate pseudo column 100 for indexing. This is the case for a table that has only one index column.

In this case, the indexEntryMask MUST be defined as follows:

indexEntryMask="oid.....100" : the oid itself contains column 100, located 2 places after the table's base oid. (Standard case)

Another valid form would be the following (not related to the previous example):

tableIndex=".1.100 .1.101" indexEntryMask="oid.....100.101.101.101"

Where the index column 100 would be taken from the oid, 2 places after the table's base oid, and index column 101 would be extracted from the 4 following places. This models the case of a table that has 2 index: an integer, followed by an IP address. This could be used with the ipNetToMediaTable, which has this type of index structure, although it's index columns are specified explicitly.

There is also another way to specify the indexEntryMask when the OID structure is not stable. In certain cases, OID's length may vary on a row-by-row basis. This case warrants the use of the following syntax: tableIndex=".1.100 .1.101" indexEntryMask="oid.....100.101+"

In this case, it means that the OID index structure is made of one int, followed by an unspecified number of ints. One must note that the "oid.....100.101+" syntax allows the use of only ONE element with an undefined number of occurrences. Valid variations of this structure could be as follows:

- indexEntryMask="oid.....100+"
- indexEntryMask="oid.....100+.101"
- indexEntryMask="oid.....100.101+"
- indexEntryMask="oid.....100.101+.102"

Note: This index pseudo-column generation can be used on tables which have index columns that are not accessible (the column is defined in the MIB but the device does not reply to queries on the column's oid. But it can also be used with standard tables as it has the advantage of reducing the required polling bandwidth because the generated index columns do not require to be polled to properly index the table rows.

The following attributes are optional:

- `tableColNameOid`: specifies which column values must be used, in the table, to set the part property for all polled columns. i.e.: `tableColNameOid=".1.1.1.2"` to use column 1 and 2 values.
- `indexEntryMask`: this describes the table's rows oid structure and is used to generate index pseudo-columns when the table does not have explicit index columns. The following is an example which describes how it works: When polling table `cpmProcessTable` (.1.3.6.1.4.1.9.9.109.1.2.2.1), which is indexed by `cpmCPUTotalIndex` and `cpmProcessPID`, column 1, the resulting oid will have this form: `cpmProcExtMemAllocated.1.2` where the .1 maps to `cpmCPUTotalIndex` and .2 is the `cpmProcessPID`.

By using `indexEntryMask="oid....100.101"`, it is possible to assign the pseudo-column 100 to `cpmCPUTotalIndex` and pseudo-column 101 to `cpmProcessPID`.

It then becomes possible to use those index columns for joining external tables for filtering or referencing properties.

- `partType`: specifies the raw value's property parttype. This is user defined.

Finally, the *snmp-table* element may contain *columns-and-filters*, *snmp-reference-table* and *inner-join-on* elements. We will start with simple examples, using only the *columns-and-filters* element. This element, as it's name implies, contains only *snmp-column* (at least 1!) and *snmp-filter* elements.

3 Simple table column polling

Now, let's see some useful examples. First, a very simple one:

```
<snmp-table id="tableZ" name="hrStorageTable" oid=".1.3.6.1.2.1.25.2.3"
  tableIndex=".1.1.1.1.3" tableColNameOid=".1.3">
  <columns-and-filters>
    <snmp-column name="hrStorageSize" appendOid=".1.5" as="COUNTER" unit="KB"/>
    <snmp-column name="hrStorageUsed" appendOid=".1.6" as="COUNTER" unit="KB"/>
  </columns-and-filters>
</snmp-table>
```

This tells the SNMP collector to poll the hrStorageTable (.1.3.6.1.2.1.25.2.3) columns 5 (hrStorageSize) and 6 (hrStorageUsed). Handle the polled values as COUNTER, with their units set as KB. All of the raw values will have their part property set with the value in column 3 (hrStorageDescr) of the table.

The variable names will be created using the values provided by the index columns, in this case column 1 and 3. According to the HOST-RESOURCES_MIB, the index for this table should be column 1 only. However, there are no guarantee that the relation between the index column value and the hrStorageDescr will always be the same, that is why we must specify to use both columns 1 and 3: if the relation changes, a new variable will be generated but it will still be possible to find the older variables in the backend, therefore ensuring survival of the variable for reporting purposes. One would be tempted to use column 3 only instead of this combination but we must keep in mind that SNMP agents may be buggy and that column 3 may not contain any data, which would result in generating the raw value multiple times. Again, one must check the behaviour of the polled *SNMP* agent in order to write valid and useful configuration.

Here another equivalent version of the first example:

```
<snmp-table id="tableZ" name="hrStorageTable" oid=".1.3.6.1.2.1.25.2.3"
  tableIndex=".1.100.1.3" tableColNameOid=".1.3" indexEntryMask="oid._._.100">
<columns-and-filters>
  <snmp-column name="hrStorageSize" appendOid=".1.5" as="COUNTER" unit="KB"/>
  <snmp-column name="hrStorageUsed" appendOid=".1.6" as="COUNTER" unit="KB"/>
</columns-and-filters>
</snmp-table>
```

In this case, we are telling the SNMP Collector that it may retrieve the index values from the values polled in columns 5 and 6 instead of polling the table's column 1. Using this approach will improve polling time and reduce network traffic (by 25% in this case) as the index column 100 value will be extracted from the value found in column 3 of the table.

There are also cases where no other alternative is available. For instance, let's look at the hrProcessorTable in which two columns may be polled: hrProcessorFrwld and hrProcessorLoad. One may note that no index column is present in this table, therefore we must use this approach to poll the table:

```
<snmp-table id="tableZ" name="hrProcessorTable" oid=".1.3.6.1.2.1.25.3.3"
  tableIndex=".1.100" tableColNameOid=".1.100" indexEntryMask="oid._._.100">
<columns-and-filters>
  <snmp-column name="hrProcessorFrwld" appendOid=".1.1" as="COUNTER"/>
  <snmp-column name="hrProcessorLoad" appendOid=".1.2" as="COUNTER"/>
</columns-and-filters>
</snmp-table>
```

In this case, the part property of the generated raw values will be set to the polled value's corresponding index value. This is not ideal as the real device name would be preferable. Unfortunately, this device name is not available in this table, therefore, it cannot be retrieved directly. Section 8 of this document will show a more advanced polling configuration which allows polling external tables for property values.

4 *snmp-column* attributes

Now, before continuing with further example, a description of the *snmp-column* element may be in order: The *snmp-column* element defines the table columns to poll. It may contain 0 or several *snmp-filter* elements which are used to determine which rows will produce rawValues.

The following attributes are required:

- **name**: the oid name. It may or may not be the same as the oid name specified in a MIB. This is the name property in the resulting rawValue.
- **appendOid**: This specifies the table's column to poll. It's format is ALWAYS ".1.X", where X is the column number.

The following attributes are optional:

- **as**: see *snmp-oid* element for description.
- **unit**: see *snmp-oid* element for description.
- **varNameGenOidSequence**: Use the appendOids (".1.2 .1.3") values to generate the variable name for this column. This is useful in certain cases, where the variable index cannot otherwise be generated.
- **propertyName**: add the named property to the RawValues which results from polling this column. Poll the property value in the column specified by **propertyValueAppendOid**.
- **propertyValueAppendOid**: Used with the **propertyName** attribute to specify the column in which the value for the property will be found. It's format is ALWAYS ".1.X", where X is the column number. Several columns can be specified here, following this format: ".1.X .1.Y .1.Z".
- **refreshOnPropertyChange**: refresh the raw value's properties on change. Set to "true" to enable this behaviour. Defaults to "false" if not specified.

5 Simple filtering on table columns

The following is an example of filtering applied on polled columns. In this case, the filter is applied on all columns as it resides directly in the *snmp-table* element:

```
<snmp-table id="table1a" name="ifTable" oid=".1.3.6.1.2.1.2.2"
  tableIndex=".1.1" tableColNameOid=".1.2">
  <columns-and-filters>
    <snmp-column name="ifInOctetsRate" appendOid=".1.10" as="RATE" unit="octets"
      propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2"/>
    <snmp-column name="ifOutOctetsRate" appendOid=".1.16" as="RATE" unit="octets"
      propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2"/>
    <snmp-filter name="ifTypeEthernet" onTableName="ifType" appendOid=".1.3"
      valueEquals="etherne.*"/>
  </columns-and-filters>
</snmp-table>
```

In this case, we are telling the SNMP collector to poll ifInOctets (column 10) and ifOutOctets (column 16) of the ifTable (.1.3.6.1.2.1.2.2) but keep only the table rows where the ifType matches the "etherne.*" regular expression. Then, set the resulting raw value's physAddr property to the value found in column 6 of the ifTable, which is ifPhysAddr. Set the raw value's part property to the value found in column 2 of the table (ifDescr) and, finally, use the values found in columns 1 and 2 of the table to create the variable name.

The next example is similar to the previous one, except that filtering is applied only on the ifInOctetsRate column, while all of the values present in the ifOutOctets column will be polled:

```
<snmp-table id="table1y" name="ifTable" oid=".1.3.6.1.2.1.2.2" tableIndex=".1.1"
tableColNameOid=".1.2">
<columns-and-filters>
  <snmp-column name="ifInOctetsRate" appendOid=".1.10" as="COUNTER" unit="octets"
    propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2">
    <snmp-filter name="ifTypeEthernet" onTableColumnName="ifType" appendOid=".1.3"
      valueEquals="etherne.*"/>
  </snmp-column>
  <snmp-column name="ifOutOctetsRate" appendOid=".1.16" as="COUNTER" unit="octets"
    propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2"/>
</columns-and-filters>
</snmp-table>
```

The *snmp-filter* elements may also be present on both levels but bear in mind that their effect can be cumulative.

The next example shows this more clearly:

```
<snmp-table id="table1x" name="ifTable" oid=".1.3.6.1.2.1.2.2" tableIndex=".1.1"
tableColNameOid=".1.2">
<columns-and-filters>
  <snmp-column name="ifInOctetsRate" appendOid=".1.10" as="COUNTER" unit="octets"
    propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2">
    <snmp-filter name="ifTypeEthernet" onTableColumnName="ifType" appendOid=".1.3"
      valueEquals="etherne.*"/>
  </snmp-column>
  <snmp-column name="ifOutOctetsRate" appendOid=".1.16" as="COUNTER" unit="octets"
    propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2"/>
  <snmp-filter name="ifSpeed=100000000" onTableColumnName="ifSpeed"
    appendOid=".1.5" valueEquals="100000000"/>
</columns-and-filters>
</snmp-table>
```

In this case, the ifInOctetsRate column will be filtered through two criteria: ifType matches "etherne.*" and ifSpeed matches "100000000" while the ifOutOctetsRate will be filtered only by the ifSpeed criteria.

One must note that the power of regular expression may be used to provide OR filtering conditions. The following is a valid filter:

```
<snmp-filter name="ifSpeed=100000000" onTableColumnName="ifSpeed" appendOid=".1.5"
valueEquals="10000000|57600"/>
```

In this case, we are filtering rows where the ifSpeed matches "10000000" or "57600".

6 AND OR column filtering conditions

It is also possible to configure OR condition filtering by having several *snmp-filter* elements on the same level (in either *snmp-column* or *snmp-filter* element):

```
<snmp-table id="table1w" name="ifTable" oid=".1.3.6.1.2.1.2.2" tableIndex=".1.1"
  tableColNameOid=".1.2">
  <columns-and-filters>
    <snmp-column name="ifInOctetsRate" appendOid=".1.10" as="COUNTER" unit="octets"
      propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2">
      <snmp-filter name="ifTypeEthernet" onTableColumnName="ifType" appendOid=".1.3"
        valueEquals="etherne.*"/>
      <snmp-filter name="ifSpeed=100000000" onTableColumnName="ifSpeed" appendOid=".1.5"
        valueEquals="100000000"/>
    </snmp-column>
  </columns-and-filters>
</snmp-table>
```

In this case, we are telling the SNMP collector to poll the ifInOctets column and keep the rows that have their ifSpeed column value match the “100000000” regular expression OR their ifType column value match the “etherne.*” regular expression. The same can be done with *snmp-filter* elements located at the *snmp-column* level.

Finally, it is also possible to nest *snmp-filter* elements into a *snmp-filter* element. This is used to create AND conditions such as the one in the following example:

```
<snmp-table id="table1w" name="ifTable" oid=".1.3.6.1.2.1.2.2" tableIndex=".1.1"
  tableColNameOid=".1.2">
  <columns-and-filters>
    <snmp-column name="ifInOctetsRate" appendOid=".1.10" as="COUNTER" unit="octets"
      propertyName="physAddr" propertyValueAppendOid=".1.6" varNameGenOidSequence=".1.1.1.2">
      <snmp-filter name="ifTypeEthernet" onTableColumnName="ifType" appendOid=".1.3"
        valueEquals="etherne.*">
        <snmp-filter name="ifSpeed=100000000" onTableColumnName="ifSpeed" appendOid=".1.5"
          valueEquals="100000000"/>
        </snmp-filter>
      </snmp-column>
    </columns-and-filters>
  </snmp-table>
```

In this case, we are telling the SNMP collector to poll the ifInOctets column and keep the rows that have their ifSpeed column value match the "100000000" regular expression AND their ifType column value match the "etherne.*" regular expression.

7 *snmp-filter* attributes

On a last note, here are the descriptions of the *snmp-filter* attributes.

The following attributes are required:

- `appendOid`: The column on which filtering is applied. Must be in the ".1.X" form, where X is the column number in the SNMP table.
- `valueEquals`: The value used for filtering. It is in effect a regular expression, which simplest form is an equality:
 - `valueEquals="value"` performs the SQL equivalent of `=`, as in: `SELECT ... FROM table WHERE column='value'`
 - `valueEquals="ethernet.*"` performs the SQL equivalent of `LIKE`, as in: `SELECT ... FROM table WHERE column LIKE 'ethernet%'`

The following attributes are optional:

- `indexEntryMask`: Set this when filtered table does not have index columns for all its indexes. See *snmp-table* element for details on how this works.
- `onTableColumnName`: This is user defined and has no effect on polling. It is used for comment only.
- `name`: This user defined and has no effect on polling. It is used for comment only.
- `invertMatch`: When set to true, this inverts the regular expression match. So, if the regular expression matches a value, the line will be filtered out instead of being kept. The default behaviour is false.

As stated earlier on, further SNMP table polling cases are presented in [section 8](#) of this document.

8 SNMP properties configuration

SNMP properties are attached to every raw value produced by the *snmp-mask* which contains them. The following example shows how to configure them:

```
<snmp-properties>
  <snmp-property id="outsideProp1" name="someProp" value="hardCodedValue"/>
  <snmp-property id="outsideProp2" name="ipfrwd" oid=".1.3.6.1.2.1.4.1.0"/>
</snmp-properties>
```

The following attributes are required:

- **id**: The property's identifier. This is used internally.
- **name**: The property name. This will be the property key in the raw value.
- **value**: The property value. This is hardcoded.
- **oid**: The oid to poll in order to get the property value.

Please note that either value or oid attribute may be present. In this case, all values resulting from this mask will have a property "someProp" set to "hardCodedValue" while the "ipfrwd" will be set to the value found at the specified oid. MIB translation will be applied to this value, if the correct MIB was loaded.

9 MIB files specification

MIB Files are used to get textual values from integer enumerated values. The textual value will be set as the property value if polled as a property. However, if polled as a value, the correct translation (see next section) must be defined in order to actually translate this textual value back to an actual floating point value. Starting with APG 5.2, specifying a MIB file in the snmp-mask is now optional as the RFC1213 MIB is always loaded by default.

```
<mib-files>  
  <mib-file-name>RFC1213-MIB.mib</mib-file-name>  
  <mib-file-name>MY-TECHNOLOGY.mib</mib-file-name>  
</mib-files>
```

Translations configuration

Value translations are required when polling SNMP oids of String-octet (which contain non-numeric values) or Integer (enum) type as the resulting raw values can only contain numeric values. Configuring the translation table is done by editing the `translations.xml` XML file, which is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "translations.dtd">
<translations>
  <variable-specific-translations>
    <variable-value-translation name="Availability">
      <value-match value="up" rename="100"/>
      <value-match value="down" rename="0"/>
      <value-match value="testing" rename="0"/>
    </variable-value-translation>
  </variable-specific-translations>
</translations>
```

The only translation configured in this file is applied to any polled value named Availability. This specific case could apply to the ifTable ifAdminStatus column if it were to be polled like this:

```
<snmp-table id="table200" name="ifTable" oid=".1.3.6.1.2.1.2.2"
```

```
tableIndex=".1.1.1.2" tableColNameOid=".1.2">
<columns-and-filters>
  <snmp-column name="Availability" appendOid=".1.7" unit="%"/>
</columns-and-filters>
</snmp-table>
```

Or, alternately, like this:

```
<snmp-oids>
  <snmp-oid id="oid15" name="Availability" oid=".1.3.6.1.2.1.2.2.1.7.1" unit="%"/>
</snmp-oids>
```

In this case, any polled oid which returns the “up” value will have its value translated to “100” while other values like “down” and “testing” will be translated to “0”. Note that in both cases, the unit is set to “%”, which reflect the availability percentage.

This translation process requires the correct MIB file to be loaded in the SNMP collector. In this case, the RFC-1213 MIB file would be required as it contains the value translation for the ifAdminStatus oid.

Note: This translation process does not yield any value if the translated value does not exist for a polled non numeric value, Except for the TruthValue type, which will return 1 for true, 2 for false, if no translation is configured

8

Advanced collection configuration

As stated earlier in this document, the polling of SNMP table is one of the most advanced feature of the SNMP collector. However, configuring complex table polling is a little bit less straightforward than the polling cases which were presented in [section 6](#) of this document. Therefore, we recommend that you fully understand the *snmp-table* configuration before reading this section.

Now, let's use the following *snmp-mask* XML fragment to provide a complete example of what may be achieved with those advanced configuration options:

```
<snmp-table id="table201" name="hrDiskStorageTable" oid=".1.3.6.1.2.1.25.3.6"
  tableIndex=".1.100" indexEntryMask="oid._._.100" nextRefTableColIdxOid=".1.100">
  <snmp-reference-table name="hrDeviceTable" referenceTableOid=".1.3.6.1.2.1.25.3.2"
    refTableColIdxOid=".1.100" parentTablecolIdxOidPositions="1" propertyName="hrDevDes"
    propertyValueAppendOid=".1.3" indexEntryMask="oid._._.100"
    usePropertyValueInVariableName="true"/>
  <inner-join-on name="hrDeviceIndex" appendOid=".1.100"/>
  <columns-and-filters>
    <snmp-column name="hrDiskStorageAccess" appendOid=".1.1">
      <snmp-filter name="all" appendOid=".1.1" valueEquals=".*">
        <snmp-filter name="device_1554_or_1555" onTableName="hrDeviceTable1"
          onTableOid=".1.3.6.1.2.1.25.3.2" appendOid=".1.1" valueEquals="1554|1555">
          <outer-join-on name="hrDeviceIndex" appendOid=".1.1"/>
        </snmp-filter>
      </snmp-filter>
    </snmp-column>
    <snmp-column name="hrDiskStorageMedia" appendOid=".1.2" />
    <snmp-column name="hrDiskStorageRemovable" appendOid=".1.3" />
    <snmp-column name="hrDiskStorageCapacity" appendOid=".1.4">
      <snmp-filter name="Removable_only" appendOid=".1.3" valueEquals="false"/>
    </snmp-column>
    <snmp-filter name="SCSI_disk_only" onTableName="hrDevTable"
      onTableOid=".1.3.6.1.2.1.25.3.2" appendOid=".1.3" valueEquals="SCSI.*"
      indexEntryMask="oid._._.100">
      <outer-join-on name="hrDeviceIndex" appendOid=".1.100"/>
    </snmp-filter>
  </columns-and-filters>
</snmp-table>
```

1 Optional *snmp-table* parameters

Let's start by presenting the optional *snmp-table* parameters that were intentionally skipped when first presenting the SNMP table polling configuration:

- `nextRefTableColIdxOid`: specifies ALL of the columns that may be used to join this table to 1 or several *snmp-reference-table* objects.
- `useSnmPV2cMultipleRowPolling`: The collector will poll this table (and all nested tables) by using a row based method, using SNMP v2c `getBulk` requests. This ensures row data integrity is maintained.
- `snmpRepeaters`: The number of repeaters to use when multiple row polling is enabled. This number is used by the `getBulk` request to tell the agents how many varbinds should be returned in the pdu. Leaving this unset tells the collector to calculate this by itself. A good value to start with, when polling counters only, is 50.
- `cacheAllSnmReferenceTables`: When set to true, all polled *snmp-reference-table* defined in this *snmp-table* will be put in a device-level cache until properties are refreshed on the table or the device. All *snmp-reference-tables*, even the ones with `usePropertyValueInVariableName="true"` are included here, which means some of the table entries will NOT be polled correctly if they were created after the table or device properties were refreshed. The *snmp-reference-table* that are cached may be used by other *snmp-tables* in other polling groups to accelerate polling. This attribute default value is "false". Please refer to the "longLifeCaching" attribute set on *snmp-reference-tables* for finer-grained control over what is being cached.
- `tableRowSpecificationFile`: This specifies a file which contains tab-separated `propertyName` and regular expression used to filter out lines. Raw values are output only if their properties match the criteria on one of this file's line. Also, users shall be aware that polling will be done selectively when specifying this attribute, meaning that only the table rows which matched the criteria on the first poll (or after a device refresh command was received) will be polled again. This behaviour effectively suppresses any unrequired polling on very long tables (tables with a thousand rows may be polled extremely quickly when few rows are selected). The side effect to this approach is that new rows which match the criteria will not be polled until the next property refresh cycle (either on the agent or the table).

This file format is described below:

```
property1[TAB]regex1[TAB]property2[TAB]regex2[TAB]...propertyN[TAB]regexN*
```

When the table is polled the first time, the raw values' properties are checked against the regular expressions. If all regular expressions match on one line (several lines may be present in the file), the table row's oid is added to set of oids to poll for this table on the snmp agent. The set

of oids is polled on the next polling, instead of polling the entire table. This allows quicker polling times on long tables for which only a few lines should be polled. This works especially well on static tables (such Cisco QoS statistics tables, and interface tables on routers and switches).

Please refer to Appendix D for an example and further explanation on this matter.

- `matchNFirstOidSuffixOnly`: This parameter MUST be combined with the `tableRowSpecificationFile` parameter as it alters the way rows are selected in the table. The value to input in this parameter is numeric and should be higher than 0, otherwise selection will occur as described in the `tableRowSpecificationFile` paragraph. Any value higher than 0 specifies the length of the static oid suffix to use in order to poll the table. This option is used when polling tables with rows that are added or modified over time.

As an example, let's use the Cisco IPSLA `rttMonStatsCaptureTable`, which is indexed by `rttMonCtrlAdminIndex`, `rttMonStatsCaptureStartTimeIndex` and 3 other fields. In this case, the first part of the table's oid suffixes (`rttMonCtrlAdminIndex`) does not change over time, while the rest of the oid suffixes will change over time (a new `rttMonStatsCaptureStartTimeIndex` is created on every probe operation). Therefore, selective row polling can be done by specifying only the partial "static" part of the oid so that all rows having the partial oid will be retrieved.

Please refer to Appendix E for an example and further explanation on this matter.

- `sortOnOid`: Setting this to "true" ENSURES that the SNMP Collector will output the tables raw values in their sorted order (ordered by their oid). Default value is "false" as it increases overhead and is not required in most cases.

The `snmp-reference-table` object will specify which column to use for the join in this set through it's `parentTableColIdxOidPositions` attribute.

The following is an example of use:

```
<snmp-table ... nextRefTableColIdxOid=".1.1.1.2.1.6.1.8">
  <snmp-reference-table ... parentTableColIdxOidPositions="1,3"
    refTableColIdxOid=".1.1.1.2" />
  <snmp-reference-table ... parentTableColIdxOidPositions="2,3,4"
    refTableColIdxOid=".1.2.1.5.1.8" />
</snmp-table>
```

In this case, the `snmp-table` will use columns 1,2,6,8 for joining `snmp-reference-table`. The first `snmp-reference-table` will use values from columns 1 and 6 (positions 1 and 3 in the `nextRefTableColIdxOid` attribute) from the `snmp-table` and match them on it's columns 1 and 2 values to perform the join. The second `snmp-reference-table` will use values from columns 2, 6 and 8 (positions 2, 3 and 4 in the `nextRefTableColIdxOid` attribute) from the `snmp-table` and match them on it's columns 2, 5 and 8 values to perform the join.

```
<snmp-table id="table201" name="hrDiskStorageTable" oid=".1.3.6.1.2.1.25.3.6"
  tableIndex=".1.100" indexEntryMask="oid._._.100" nextRefTableColIdxOid=".1.100">
  <snmp-reference-table name="hrDeviceTable" referenceTableOid=".1.3.6.1.2.1.25.3.2"
    refTableColIdxOid=".1.100" parentTablecolIdxOidPositions="1" propertyName="hrDevDes"
    propertyValueAppendOid=".1.3" indexEntryMask="oid._._.100"
    usePropertyValueInVariableName="true"/>
  <inner-join-on .../>
  <columns-and-filters>
    ...
  </columns-and-filters>
</snmp-table>
```

Note: The name attributes in the *snmp-reference-table* elements must be different, otherwise mismatch may occur when retrieving property values from those tables. However, it is possible (and desirable) to name identical *snmp-reference-table* declared within the same snmp-mask with the same name as only one copy will be polled and put in cache, thus avoiding any further unrequired polling of *this snmp-reference-table*.

2 Optional *snmp-table* nested elements

There are also two elements, nested in *snmp-table*, which require to be described fully in order to configure complex polling. These elements are: *snmp-reference-table* and *inner-join-on*.

2.1 *snmp-reference-table* element

The *snmp-reference-table*, as it's name implies, is used to provide property values coming from a table that is not the one being polled. It may also be used to provide values for supplementary properties which values come from the polled table, if more than one property is taken from that table.

The following attributes are required:

- **name**: the reference table name. It must be unique as it used as an id in the polling process.
- **referenceTableOid**: the base table oid. i.e.: ifTable's oid is .1.3.6.1.2.1.2.2
- **refTableColIdxOid**: this specifies the *snmp-reference-table* index columns. These columns are used to join the *snmp-reference-table* to the parent *snmp-table* or *snmp-reference-table*. It is always specified in the ".1.X .1.Y .1.Z" form, where X, Y, Z are column numbers.
- **parentTableColIdxOidPositions**: mandatory when the *snmp-reference-table* is joined to a *snmp-table*. Not used when a nested *snmp-reference-table* is contained in another *snmp-reference-table*.

The following attributes are optional:

- **propertyName**: the property name to define. Must be less than 8 characters long.
- **propertyValueAppendOid**: the column where the property value will be taken. It must be specified in the ".1.X" form, where X is the column number.
- **usePropertyValueInVariableName**: true or false. This tells the SNMP collector to use the value polled in this table when creating the raw value variable name. It is therefore possible to create a variable name based entirely on values polled from a *snmp-reference-table*.
- **nextRefTableColIdxOid**: specifies the join columns to use when a further *snmp-reference-table* is defined within this one. It is possible to define a chain of *snmp-reference-table* elements in order to get properties that are several tables away from where the value is polled. It is always specified in the ".1.X .1.Y .1.Z" form, where X, Y, Z are column numbers.
- **indexEntryMask**: see *snmp-table* element for description and usage.
- **longLifeCaching**: set to true to put this *snmp-reference-table* in cache for the device. Cached *snmp-reference-table* are refreshed only the table properties are refreshed when sending a refresh agent command to the snmp collector. Cached *snmp-reference-tables* may be used by any

polling group with snmp-tables that refer to it. This allows shorter polling times on slow devices as well as bandwidth usage reduction.

```
<snmp-table id="table201" name="hrDiskStorageTable" oid=".1.3.6.1.2.1.25.3.6"
  tableIndex=".1.100" indexEntryMask="oid._._.100" nextRefTableColIdxOid=".1.100">
  <snmp-reference-table name="hrDeviceTable" referenceTableOid=".1.3.6.1.2.1.25.3.2"
    refTableColIdxOid=".1.100" parentTablecolIdxOidPositions="1"
    propertyName="hrDevDes" propertyValueAppendOid=".1.3"
    indexEntryMask="oid._._.100" usePropertyValueInVariableName="true"/>
  <inner-join-on .../>
  <columns-and-filters>
    ...
  </columns-and-filters>
</snmp-table>
```

2.2 The *inner-join-on* element

The *inner-join-on* element specifies which of the table's (or parent filter's) index columns to use for joining filters that are applied on external tables.

The following attributes are required:

- **name**: the name of the *inner-join-on* condition. It is used to identify it's *outer-join-on* counterpart. The *outer-join-on* is defined in *snmp-filter* located within the *snmp-table* or within a filter itself.
- **appendOid**: Specifies the index columns to use for joining the table and the table resulting from the filtering.

```
<snmp-table id="table201" name="hrDiskStorageTable" oid=".1.3.6.1.2.1.25.3.6"
  tableIndex=".1.100" indexEntryMask="oid.__.100" nextRefTableColIdxOid=".1.100">
  <snmp-reference-table ... />
  <inner-join-on name="hrDeviceIndex" appendOid=".1.100"/>
  <columns-and-filters>
    <snmp-column name="hrDiskStorageAccess" appendOid=".1.1">
      <snmp-filter ... >
        <snmp-filter name="device_1554_or_1555" onTableName="hrDeviceTable1"
          onTableOid=".1.3.6.1.2.1.25.3.2" appendOid=".1.1" valueEquals="1554|1555">
          <outer-join-on name="hrDeviceIndex" appendOid=".1.1"/>
        </snmp-filter>
      </snmp-filter>
    </snmp-column>
    ...
    <snmp-filter name="SCSI_disk_only" onTableName="hrDevTable"
      onTableOid=".1.3.6.1.2.1.25.3.2" appendOid=".1.3" valueEquals="SCSI.*"
      indexEntryMask="oid.__.100">
      <outer-join-on name="hrDeviceIndex" appendOid=".1.100"/>
    </snmp-filter>
  </columns-and-filters>
</snmp-table>
```

In this case, we are telling the SNMP collector to use column .1.100 to join the hrDiskStorageTable to tables hrDeviceTable and hrDevTable used in the *snmp-filter* elements in the table. One must notice that this is the hrDiskStorageTable index column, even though it does not exist explicitly in the table. That is why the indexEntryMask attribute must be used to retrieve it.

3 Optional *snmp-filter* attributes and nested elements

As one may have noticed, a few new optional parameters were introduced in the example shown earlier in this section. All of these parameters and attributes are used when filtering on external tables.

3.1 Optional *snmp-filter* attributes

- `onTableName`: Specify this only when:
 - The filtering must be applied on a table which is not the same as the *snmp-table* which contains this *snmp-filter*.
 - The filtering must be applied on a table which is not the same as the *snmp-filter* which contains this *snmp-filter*.

The `onTableName` must be specified in a logical way and is used as an identifier by the table polling process. Therefore, it must be unique in the defined *snmp-table* element which contains it.

- `onTableOid`: Specify this with the `onTableOidName`. This is the external table base Oid.

3.2 The *outer-join-on* element

The *outer-join-on* element is used exclusively in *snmp-filter* elements. It is either present or not in the *snmp-filter* (0..1). It is used to specify the table columns used for joining it with the table defined by the parent *snmp-filter* or *snmp-table*.

The following attributes are required:

- **name**: the name of the *outer-join-on* condition. This must match the *inner-join-on* matching condition's name.
- **appendOid**: The join columns definition, in the ".1.X .1.Y .1.Z" form.

```
<snmp-table id="table201" name="hrDiskStorageTable" oid=".1.3.6.1.2.1.25.3.6"
  tableIndex=".1.100" indexEntryMask="oid.1.100" nextRefTableColIdxOid=".1.100">
  <inner-join-on name="hrDeviceIndex" appendOid=".1.100"/>
  <columns-and-filters>
    <snmp-column ...>
      <snmp-filter ...>
        <snmp-filter ...>
          <outer-join-on name="hrDeviceIndex" appendOid=".1.1"/>
        </snmp-filter>
      </snmp-filter>
    </snmp-column>
    ...
    <snmp-filter name="SCSI_disk_only" onTableName="hrDevTable" onTableOid=".1.3.6.1.2.1.25.3.2"
      appendOid=".1.3" valueEquals="SCSI.*" indexEntryMask="oid.1.100">
      <outer-join-on name="hrDeviceIndex" appendOid=".1.100"/>
    </snmp-filter>
  </columns-and-filters>
</snmp-table>
```

In this case, we can clearly see the relation between both *inner-join-on* and *outer-join-on* elements as they actually represent the columns used for joining the tables. What we mean by joining the tables is that, in this case, the values found in column 100 of the *hrDevTable* match the values found in column 100 of the *hrDiskStorageTable*. In this case (as with most cases), the match is performed on index columns. For all practical purposes, this means we are simply taking both tables and aligning each other's rows via their index columns.

The following figures show this relationship between tables:

TABLE hrDevTable

Column 1 (100)	Column 2	Column 3
hrDeviceIndex	hrDeviceType	hrDeviceDescr
768	.1.3.6.1.2.1.25.3...	AuthenticAMD: AMD Athlon(tm) 64 Processor 3500+
1025	.1.3.6.1.2.1.25.3...	network interface lo
1026	.1.3.6.1.2.1.25.3...	network interface eth0
1027	.1.3.6.1.2.1.25.3...	network interface eth1
1028	.1.3.6.1.2.1.25.3...	network interface sit0
1280	.1.3.6.1.2.1.25.3...	LEXMARKOC710
1536	.1.3.6.1.2.1.25.3...	ST3200021A
1538	.1.3.6.1.2.1.25.3...	LITE-ON DVDRW SOHW-1633S
1539	.1.3.6.1.2.1.25.3...	SAMSUNG CD-ROM SC-148A
1552	.1.3.6.1.2.1.25.3...	SCSI disk (/dev/sda)
1553	.1.3.6.1.2.1.25.3...	SCSI disk (/dev/sdb)
1554	.1.3.6.1.2.1.25.3...	SCSI disk (/dev/sdc)
1555	.1.3.6.1.2.1.25.3...	SCSI disk (/dev/sdd)
3072	.1.3.6.1.2.1.25.3...	Guessing that there's a floating point co-processor

Figure 2: Table on which filtering takes place

TABLE hrDiskStorage

Column 1	Column 2	Column 3	Column 4	Column 100 (taken from 1)
hrDiskStorageAccess	hrDiskStorageMedia	hrDiskStorageRemoveble	hrDiskStorageCapacity	Name/OID
readWrite	unknown	2	134217727	hrDiskStorageAccess.1536
readWrite	unknown	1	0	hrDiskStorageAccess.1538
readWrite	unknown	1	0	hrDiskStorageAccess.1539
readWrite	unknown	2	0	hrDiskStorageAccess.1552
readWrite	unknown	2	0	hrDiskStorageAccess.1553
readWrite	unknown	2	0	hrDiskStorageAccess.1554
readWrite	unknown	2	0	hrDiskStorageAccess.1555

Figure 3: Table which contains the column to be polled

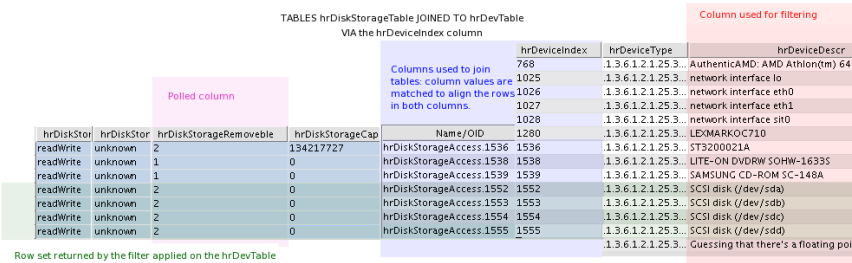
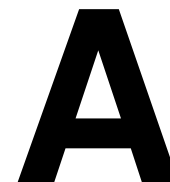


Figure 4: Tables joined via inner-join-on and outer-join-on elements

4 SNMP filtering limitations

There are currently two known limitations which applies to *snmp-filter* elements:

- 1. Filtering on an external table may not be nested directly in *snmp-column* elements. Filtering on the base table must be done first, as shown in the example presented in section 8 of this document. The impact on polling time of this filtering may be minimized by filtering on a column declared in one of the *snmp-table* element as no supplementary polling is required.
- 2. Values taken from index pseudo-columns cannot be translated as their OID does not correspond to one defined in any MIBs. Therefore, when filtering on those column, it is required to filter on the raw (possibly integer) value instead of on the text translation.



External commands

A set of external SNMP Collector tools is provided by the `remote-snmp-collector-cli` package. Please install it to use the following commands:

- `add-snmp-agents-list.sh`
- `extract-from-snmp-polling-distribution.sh`
- `send-snmp-command.sh`
- `snmp-collector-ws-cli.sh`

1 The `add-snmp-agents-list.sh` command

A script is provided in order to easily add *snmp-agents* to an existing `snmp-polling-distribution.xml` file. This script default location is `<APG>/Collecting/SNMP-Collector/<Instance>/bin`.

Note: The script name for the Linux / Solaris environment is: *add-snmp-agents-list.sh*.

The script's usage is as follows:

```
./addSnmpAgentsList seedFile snmp-polling-distribution.xml [slavePollerName]
```

The `seedFile` contains snmp agents definitions to be added to the existing polling distribution configuration (`snmp-polling-distribution.xml`). Please note that a backup of this file will be created upon completion of the operation and that a copy of the corresponding DTD (`snmp-polling-distribution.dtd`) must exist in the directory where the `snmp-polling-distribution.xml` file is located.

Three syntaxes are available for the given seed file, from the simplest one to the most complicated one:

- basic: IP,name
- standard: IP,name,[community string],[port],[SNMP version]
- full: Name,IP,port,protocol,timeout,retries,version,community,authProto,authPass,privProto,privPass,devtype,onFailTryWith[,explicit-group][,snmpV3ContextName,useV3ContextName]

Lines in the file may not be interrupted by a carriage return.

Note: When the `slavePollerName` parameter is not provided on the command line, then the first line of the `seedFile` *MUST* contain it!

```
Corresponds to the attribute <snmp-slave-poller name=""...> : slave2
Snmp v1 and v2c syntax:
dev1,10.128.2.50,161,udp,3000,3,v2c,public,,,,Router,doNotRetry,Group1
Snmp V3 syntax (no context name, no context engine id):
dev2,10.128.2.55,161,udp,3000,3,v3,userV3,sha,authName,aes,privName,Router,doNotRetry,Group3
Snmp V3 syntax (with context name and context engine id; leave value as empty if not specified)
dev2a,10.128.2.58,161,udp,3000,3,v3,userV3,sha,authName,aes,privName,Router,doNotRetry,Group3,
context,true,A0A1A2A3A4A5A6,true

Common syntax (directly applicable to SNMP V1 and V2c):
EquipmentName,IPAddress,SnmpPort,transportProtocol,timeout(in ms),retries,snmpVersion,
communityString,,,,EquipmentType,onFailTryWith,SnmpAgentExplicitGroupName
```

Note: The `onFailTryWith` value may be: `doNotRetry`, `v1` or `v2c`. Please check section [5](#) for further information about the “onFailTryWith” options behavior.

The last field may be left empty: if the agent is not to be included in a specific explicit snmp agent group.

The following is an example of the content present in a newSnmpAgentsFile:

```
slave2
Routr8,10.128.2.50,161,udp,3000,3,v2c,public,,,,Router,v1,Group1
Routr9,10.128.2.51,161,udp,3000,3,v2c,public,,,,Router,v2c,Group2
Routr10,10.128.2.52,161,udp,3000,3,v2c,public,,,,Router,v1,Group3
Routr11,10.128.2.53,161,udp,3000,3,v3,userV3,md5,authName,des,privName,
Router,doNoRetry,Group1
Routr12,10.128.2.54,161,udp,3000,3,v3,userV3,sha,authName,aes,privName,
Router,doNotRetry,Group2
Routr13,10.128.2.55,161,udp,3000,3,v3,userV3,sha,authName,aes,privName,
Router,v1
```

This file, added to the following snmp-polling-distribution.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE config SYSTEM "snmp-polling-distribution.dtd" -->
<snmp-polling-distribution >
  <snmp-slave-poller name="slave2" enabled="true" host="10.128.1.31" port="2000"
    slaveSnmpPollerXmlFile="slave-snmp-poller-slave2.xml" >
    <snmp-agent name="RH-SERVEUR-YYZ-REGIONAL" ipAddress="192.168.6.1"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" onFailTryWith="v1"/>
    <snmp-agent name="RH-SERVEUR-YYG-REGIONAL" ipAddress="192.168.4.4"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" onFailTryWith="v1"/>
    <snmp-agent name="RH-SERVEUR-YBG-REGIONAL" ipAddress="192.168.4.2"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v3" snmpCommunityOrUserV3Name="userV3"
      snmpV3AuthenticationProtocol="md5" snmpV3AuthenticationPassword="v3pass"
      snmpV3PrivacyProtocol="des" snmpV3PrivacyPassword="private"
      devtype="Server" onFailTryWith="v1"/>
    <snmp-agent name="RH-ROUTEUR-YQX-REGIONAL" ipAddress="192.168.5.7"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Router" onFailTryWith="v1"/>
    <snmp-agent name="RH-FRIGO-YGK-REGIONAL" ipAddress="192.168.7.1"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v1" snmpCommunityOrUserV3Name="public" devtype="Router" onFailTryWith="v1"/>
    <snmp-agent name="RH-SERVEUR-YYC-REGIONAL" ipAddress="192.168.4.3"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" onFailTryWith="v1"/>
    <snmp-agent name="RH-SERVEUR-YMM-REGIONAL" ipAddress="192.168.4.5"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" onFailTryWith="v1"/>
    <snmp-agent name="RH-ROUTEUR-YFC-REGIONAL" ipAddress="192.168.5.6"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Router" onFailTryWith="v1"/>
    <snmp-agent name="RH-ROUTEUR-YYR-REGIONAL" ipAddress="192.168.5.8"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Router" onFailTryWith="v1"/>
    <snmp-agents-ip-group name="ROUTERS" ipAddressMask="192.168.5.*"/>
    <snmp-agents-ip-group name="MACHINES" ipAddressMask="192.168.*.1"/>
  </snmp-slave-poller>
</snmp-polling-distribution>
```

Will yield the following snmp-polling-distribution.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE config SYSTEM "snmp-polling-distribution.dtd"> -->
<snmp-polling-distribution >
  <snmp-slave-poller name="slave2" enabled="true" host="10.128.1.31" port="2000"
    slaveSnmpPollerXmlFile="slave-snmp-poller-slave2.xml" >
    <snmp-agent name="Routeur8" ipAddress="10.128.2.50" transportProtocol="udp"
      snmpPort="161" timeout="15000" retries="3" snmpVersion="v2c"
      snmpCommunityOrUserV3Name="public" devtype="Router" onFailTryWith="v1"/>
    <snmp-agent name="Routeur9" ipAddress="10.128.2.51" transportProtocol="udp"
      snmpPort="161" timeout="15000" retries="3" snmpVersion="v2c"
      snmpCommunityOrUserV3Name="public" devtype="Router" onFailTryWith="v2c"/>
    <snmp-agent name="Routeur10" ipAddress="10.128.2.52" transportProtocol="udp"
      snmpPort="161" timeout="15000" retries="3" snmpVersion="v2c"
      snmpCommunityOrUserV3Name="public" devtype="Router" onFailTryWith="v1"/>
    <snmp-agent name="Routeur11" ipAddress="10.128.2.53" transportProtocol="udp"
      snmpPort="161" timeout="15000" retries="3" snmpVersion="v2c"
      snmpCommunityOrUserV3Name="userV3" devtype="Router" onFailTryWith="doNotRetry"/>
    <snmp-agent name="Routeur12" ipAddress="10.128.2.54" transportProtocol="udp"
      snmpPort="161" timeout="15000" retries="3" snmpVersion="v2c"
      snmpCommunityOrUserV3Name="userV3" devtype="Router" onFailTryWith="doNotRetry"/>
    <snmp-agent name="Routeur13" ipAddress="10.128.2.55" transportProtocol="udp"
      snmpPort="161" timeout="15000" retries="3" snmpVersion="v2c"
      snmpCommunityOrUserV3Name="userV3" devtype="Router" onFailTryWith="v1"/>
    <snmp-agent name="RH-SERVEUR-YBG-REGIONAL" ipAddress="192.168.4.2"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v3" snmpCommunityOrUserV3Name="userV3"
      snmpV3AuthenticationProtocol="md5" snmpV3AuthenticationPassword="passv3"
      snmpV3PrivacyProtocol="des" snmpV3PrivacyPassword="private" devtype="Server" />
    <snmp-agent name="RH-SERVEUR-YYC-REGIONAL" ipAddress="192.168.4.3"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" />
    <snmp-agent name="RH-SERVEUR-YYG-REGIONAL" ipAddress="192.168.4.4"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" />
    <snmp-agent name="RH-SERVEUR-YMM-REGIONAL" ipAddress="192.168.4.5"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" />
    <snmp-agent name="RH-ROUTEUR-YFC-REGIONAL" ipAddress="192.168.5.6"
      transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
      snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Router" />
    <snmp-agent name="RH-ROUTEUR-YQX-REGIONAL" ipAddress="192.168.5.7"
```

```

transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Router" />
<snmp-agent name="RH-ROUTEUR-YYR-REGIONAL" ipAddress="192.168.5.8"
transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Router" />
<snmp-agent name="RH-SERVEUR-YYZ-REGIONAL" ipAddress="192.168.6.1"
transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
snmpVersion="v2c" snmpCommunityOrUserV3Name="public" devtype="Server" />
<snmp-agent name="RH-FRIGO-YGK-REGIONAL" ipAddress="192.168.7.1"
transportProtocol="udp" snmpPort="161" timeout="3000" retries="3"
snmpVersion="v1" snmpCommunityOrUserV3Name="public" devtype="Router" />
<snmp-agents-ip-group name="MACHINES" ipAddressMask="192.168.*.1" />
<snmp-agents-ip-group name="ROUTERS" ipAddressMask="192.168.5.*" />
<snmp-agents-explicit-group name="Group1" >
<agent-ip-address>10.128.2.50</agent-ip-address>
<agent-ip-address>10.128.2.53</agent-ip-address>
</snmp-agents-explicit-group>
<snmp-agents-explicit-group name="Group2" >
<agent-ip-address>10.128.2.51</agent-ip-address>
<agent-ip-address>10.128.2.54</agent-ip-address>
</snmp-agents-explicit-group>
<snmp-agents-explicit-group name="Group3" >
<agent-ip-address>10.128.2.52</agent-ip-address>
</snmp-agents-explicit-group>
</snmp-slave-poller>
</snmp-polling-distribution>

```

When using the *Basic* or *Standard* mode, default values will be assigned to the missing attributes:

- SNMP community string: public
- SNMP port: 161
- SNMP version: v1
- Transport protocol: UDP
- Retries: 2
- Timeout: 3000
- Devtype: Router
- onFailTryWith: v1

2 The extract-from-snmp-polling-distribution.sh command

This command is used to extract data from a snmp-polling-distribution file.

The options are:

- -poller : The slave poller name to extract information from in the specified snmp-polling-distribution.xml file. REQUIRED.
- -input : The input snmp-polling-distribution.xml file. REQUIRED.
- -ipDeviceMap : Outputs the ip,host name map file for use with a mapped-tagged-property-filter. Default file name is ipDeviceMap
- -ip : Outputs the ip seed file to use with the icmp collector. Default file name is icmpTargetsFile
- -ipDeviceMapFile : The name of the ipDevice map file
- -ipDevtypeMap : Outputs the ip, devtype map file. Default file name is ipDevtypeMap
- -ipDevtypeMapFile : The name of the ipDevtype map file
- -ipSeedFile : The name of the ip seed file for use with the icmp collector.

3 The send-snmp-command.sh command

A send-snmp-command script is provided to remotely send commands to the SNMP Collector. These commands only affect the SNMP Collector and will have no effect on the other collectors running in the same APG collector instance. This script default location is <APG>/Collecting/SNMP-Collector/<Instance>/bin.

Commands include:

- stop To stop the collecting process.
- start To start the collecting process.
- restart To restart the collecting process.

Those commands must be supplied with the IP address on which the SNMP Collector is running as well as the communication port it is using (typically, 2008). They are invoked at the prompt as follows:

```
./send-snmp-command.sh -command stop -target 192.168.1.1 -port 2008
```

Other commands are used to upload configuration files to the SNMP Collector:

- uploadPollingDistribution To upload a snmp-polling-distribution.xml file to the SNMP Collector. The -distribution argument and file name must be supplied.
- uploadSnmpMasks To upload a snmp-masks.xml file to the SNMP Collector. The -masks argument and file name must be supplied.
- uploadPollerConfiguration To upload a slave-snmp-poller.xml file to the SNMP Collector. The -poller argument and file name must be supplied.
- uploadTranslations To upload a translations.xml file to the SNMP Collector. The -translations argument and file name must be supplied.
- uploadCompleteConfiguration To upload all configuration files to the SNMP Collector. All of the arguments and file names must be provided.
- refreshAgentProperties To refresh the properties of the snmp agents polled by the collector.

Invocation is done as follows, at the command prompt:

```
./send-snmp-command.sh -command uploadCompleteConfiguration -target 192.168.1.1  
-port 2008 -distribution snmp-polling-distribution.xml -masks snmp-masks.xml  
-translations translations.xml -poller slave-snmp-poller.xml
```

Sending those commands to the SNMP Collector will commit the configuration files in its local configuration directory. The SNMP Collector must then be restarted for the configuration to take effect. Under heavy load, it is preferable to stop the SNMP Collector (by sending it a stop command), upload

the configuration files and start it again (by sending a start command).

Sending the following command tells the SNMP Collector to refresh the specified agents properties on the next polling cycle. The “agentsToRefresh.txt” file contains a list of agents to refresh. It is a CSV file with the agent IpAddress followed by the agent snmp port (usually 161), 1 agent per line.

```
./send-snmp-command.sh -command refreshAgentProperties -target 192.168.0.1
-port 2008 -agentsToRefresh agentsToRefresh.txt
```

agentsToRefresh.txt file content:

```
192.168.1.1,161
192.168.1.2,161
...
```

4 The snmp-collector-ws-cli.sh command

A snmp-collector-ws-cli.sh script is provided to remotely send commands directly to the SNMP Collector through the CollectorManager or via the webservice gateway (if installed). These commands only affect the SNMP Collector and will have no effect on the other collectors running in the same APG collector instance. This script may be launched from the <APG>/bin directory.

This script is also available in <APG>/Collecting/SNMP-Collector/<Instance>/bin directory.

Webservice parameters may be set in a configuration file:

<APG>/Tools/Remote-SNMP-Collector-CLI/Default/conf/ws.properties

or provided on the command line, when invoking the script, through the appropriate arguments.

The summary of the available commands is listed below:

- `compareConfiguration` Compare the configuration files with the ones currently available to the SNMP Collector.
- `isStarted` Get the SNMP Collector collecting status.
- `listConfiguration` List the available configuration files.
- `listMibs` List the MIB files currently available to the SNMP Collector.
- `pullConfiguration` Retrieve the configuration files from the SNMP Collector.
- `pullMibs` Retrieve MIB files from the SNMP Collector.
- `pushConfiguration` Push configuration files to the SNMP Collector and restart it.
- `pushMibs` Push MIB files to the SNMP Collector.
- `refreshProperties` Refresh the properties of the specified agent ids.

- `reloadConfiguration` Reload the configuration files and perform a HOT restart on the SNMP Collector.
- `snmpConnectivity` Get the connectivity status for the specified SNMP agent.
- `snmpGet` Perform a `snmpGet` request on specified SNMP agent.
- `snmpGetBulk` Perform a `snmpGetBulk` request on specified SNMP agent.
- `snmpGetNext` Perform a `snmpGetNext` request on specified SNMP agent.
- `snmpGetSubtree` Perform a `snmpGetSubtree` request on specified SNMP agent.
- `snmpGetTable` Perform a `snmpGetTable` request on specified SNMP agent.
- `snmpGetTableColumn` Perform a `snmpGetTableColumn` request on specified SNMP agent.
- `snmpStepWalk` Perform a `snmpStepWalk` request on specified SNMP agent.
- `snmpWalk` Perform a `snmpWalk` request on specified SNMP agent.
- `startCollecting` Start the SNMP Collector collecting process.
- `stopCollecting` Stop the SNMP Collector collecting process.
- `validateConfiguration` Validate the configuration files.

Arguments required for every command are listed and described in the usage message. This message is available when invoking the script with the `-h` argument.



Raw value properties

All raw values produced by the snmp collector have the following properties:

- `contact` This value is polled on the SNMP agent, found at the `sysContact.0` OID.
- `devdesc` This value is polled on the SNMP agent, found at the `sysDescr.0` OID.
- `device` This is the device name, as input in the SNMP agent definition found in the `snmp-polling-distribution.xml` file.
- `devtype` This is the device type, as input in the SNMP agent definition found in the `snmp-polling-distribution.xml` file.
- `ip` The ip address of the polled SNMP agent.
- `location` This value is polled on the SNMP agent, found at the `sysLocation.0` OID.
- `model` This value is polled on the SNMP agent, found at the `sysObjectOID.0` OID. The value output for this property is normally the OID which corresponds to the polled device type.
- `pollgrp` The polling group, as defined in `slave-snmp-poller.xml`, where the raw value is generated. This property can be used for debugging purposes.

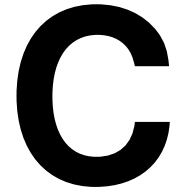
- `source` The snmp collector prefix. This property can be used when polling is distributed throughout several snmp collectors. It is also useful when polling several networks with overlapping IP addresses: devices with the same ip address polled from both networks may be distinguished by their source property.

Other common (and useful) raw value properties include:

- `parttype` The part type. This is especially useful when polling a table. As an example, the `ifTable` describes network interfaces, therefore the `parttype` associated to the raw values polled from this table should be "Interface". The same logic applies to most data polled from a table.
- `part` The part name. Again, this is especially useful when polling tables, as it is the standard way to identify a row in the table. As an example, the `ifTable` contains interfaces, which can usually be identified by their `ifDescr` column. Therefore, assigning this column's values to the `part` property is a logical choice.

Optional raw value properties are:

- `oid` The raw value's polled oid. This property may be added to the raw value by setting `oidAsPropertyInRawValues="true"` in the `slave-snmp-poller.xml` file, in the `<slave-snmp-poller>` element.



Troubleshooting and other tips

1 Setting the polling session timeouts

SNMP polling is based on several software layers which may, sometimes, fail to behave properly. Some OS networking stacks, for example, might fail to receive corrupted udp packets correctly. This may cause a timeout in one of the polling groups. Such effect can be observed in the log files (`collecting-*.log`), where a stalled polling group simply disappears. It is also visible in the APG front end, as data collected by this polling group stops being updated.

In order to circumvent this type of problem, forcibly terminating the offending snmp session is usually the only option. This is done automatically after the session `TimeToLive` expires.

By default, those `TimeToLive` values are set as follows:

- Sessions used to poll single snmp oids: 300000ms (5 minutes)
- Sessions used to poll tables: 600000ms (10 minutes)

These settings are reasonable for most cases (a 5 minute polling period) as forcibly terminating a session should occur very rarely. The Total timeout for polling both oids and tables should not exceed

15 minutes in the worst case (extremely rare occurrence, unless something is seriously wrong with the OS network stack). Please note that setting these session timeouts to a low value is not a good way to prevent skipped polling cycles: it is only meant to circumvent network stack problems.

The timeouts can be set for both sessions, by adding the following attributes to the <APG>/Collecting/CollectorManager/ <Instance>/conf/*service.properties file and updating the service. The properties to set are:

- SnmpTableSessionTTL=value1
- SnmpOidSessionTTL=value2

2 Using the disk cache for polled snmp-reference-tables

This release of the SNMP Collector allows the use of a long life cache for polled snmp-reference-tables. This cache is especially effective when polling table values that have many static properties defined in snmp-reference-table.

Using a cache has many benefits:

- Reduction of network bandwidth requirement
- Device snmp agent workload (cpu usage) is reduced
- Reduction of network polling time is reduced

The polled snmp-reference-tables are normally cached in memory. Unfortunately, this increases memory usage. One way to avoid this is to use a disk-based cache to store these polled snmp-reference-tables.

In order to activate the disk-based cache (this cache uses JCS technology), the apg-collector launch script shall be modified by adding a “-classpath conf” argument to the java invocation line.

The disk cache configuration file is <APG>/Collecting/SNMP-Collector/ <Instance>/conf/cache.ccf. The default configuration assumes that the collector name is Watch4NetSnmpCollector-1. In order to change this, edit the file by replacing every occurrence of the “Watch4NetSnmpCollector-1” name by the SNMP Collector’s name.

3 Changing batch size when polling single oids

By default, single oids (the <snmp-oid> element) are polled as a batch. The default batch size is 10 varbinds per request. This default value should work with most devices but in some cases, it may be necessary to reduce the number of requested varbinds to work around device behavior problems.

The single oid batch size can be set by adding the attribute "SnmpVarbindsPerSnmpOidsGetRequest=batchSize" to the <APG>/Collecting/CollectorManager/ <Instance>/conf/*service.properties file and updating the service.



Selective Table Row Polling Example

This example describes how to setup selective table row polling. The main goal here is to minimize network traffic and cpu load by selectively polling the device's tables. Please keep in mind that this method is most suitable to stable scenarios, as described below.

This example could be used to poll a switch (ifTable) with several physical ports assigned to several clients. Let's suppose that the client is specified in the ifXTable.1.ifAlias, as is sometimes the case to identify interfaces, and the interfaces to poll are assigned to "Acme Engineering" or "Pinnacle Oil Co". Let's also assume that the switches port assignment is relatively stable over time (and that change management accurately reflects new assignments quickly). This is a typical case to use the selective table row polling mode.

The content of ifTable is as follows:

ifIndex	ifDescr	ifType	...	ifInOctets	...	ifOutOctets	...
1	FastEthernet0	6	...	1024	...	1024	...
2	FastEthernet1	6	...	1024	...	1024	...
3	FastEthernet2	6	...	1024	...	1024	...
4	FastEthernet3	6	...	1024	...	1024	...
5	FastEthernet4	6	...	1024	...	1024	...
...	...	...	...	...	...	...	...

While ifXTable contains:

ifName	...	ifAlias
Fa0	...	Acme Engineering
Fa1	...	Dome Petrol Co.
Fa2	...	Pinnacle Oil Co
Fa3	...	Internal use only
Fa4	...	Unassigned

Both table's rows oids are formed by the table's base OID, followed by the entry OID (always 1), the column's OID and the index value OID (ifIndex in this case).

ifInOctets on interface 3 is .1.3.6.1.2.1.2.2.1.10.3

This scenario assumes that the transitional relation is stable over time:

ifTable.ifIndex -> ifXTable.ifAlias

The tableRowSpecificationFile attribute must be set at the snmp-table level in order to use selective table row polling. The file specified in this attribute must be present and "apg-user" readable, otherwise an error will occur when starting the collector.

The conf/interfacesToPoll file contains (field separator is the TAB character):

```
device customerDedicatedDevice.* ifalias Acme Engineering|Pinnacle Oil Co
```

Note: A ! character may be used in order to invert the match result, as per the following:

```
! device customerDedicatedDevice.* ifalias Acme Engineering
```

In this case, the polled lines will be the one which do not match the criteria.

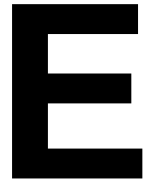
In this case, the polled interfaces are on ifIndex 1 and 3 (Fa0, Fa2). A simplified ifTable snmp-mask is used here:

```
<snmp-table name="stdIfTable2" id="stdIfT2" oid=".1.3.6.1.2.1.2.2" tableIndex=".1.100"
  tableColNameOid=".1.2" indexEntryMask="oid._._.100" nextRefTableColIdxOid=".1.100"
  partType="Interface" cacheAllSnmpReferenceTables="false"
  tableRowSpecificationFile="conf/interfacesToPoll">
<snmp-reference-table name="ifTable.ifDescr" referenceTableOid=".1.3.6.1.2.1.2.2"
  indexEntryMask="oid._._.100" refTableColIdxOid=".1.100" parentTableColIdxOidPositions="1"
  propertyName="part" propertyValueAppendOid=".1.2" usePropertyValueInVariableName="true"/>
...
<snmp-reference-table name="ifXTable.ifAlias" referenceTableOid=".1.3.6.1.2.1.31.1.1"
  refTableColIdxOid=".1.100" indexEntryMask="oid._._.100" parentTableColIdxOidPositions="1"
  propertyName="ifalias" propertyValueAppendOid=".1.18" longLifeCaching="true"/>
<columns-and-filters>
<snmp-column name="ifSpeed" appendOid=".1.5" unit="bits/s"/>
<snmp-column name="ifInOctets" appendOid=".1.10" as="RATEP" unit="octets/s"/>
<snmp-column name="ifOutOctets" appendOid=".1.16" as="RATEP" unit="octets/s"/>
<snmp-column name="ifInUcastPkts" appendOid=".1.11" as="RATEP" unit="packets/s"/>
<snmp-column name="ifInNUcastPkts" appendOid=".1.12" as="RATEP" unit="packets/s"/>
<snmp-column name="ifOutUcastPkts" appendOid=".1.17" as="RATEP" unit="packets/s"/>
<snmp-column name="ifOutNUcastPkts" appendOid=".1.18" as="RATEP" unit="packets/s"/>
<snmp-filter valueEquals="up" appendOid=".1.7"/>
</columns-and-filters>
</snmp-table>
```

Note that the ifAlias property is provided in this snmp-mask. It is therefore possible to use it in order to select the polled interfaces. also note that this property is kept in the cache, so that it does not have to be polled until the device properties are refreshed.

The following sequence of events occurs when using this polling mode:

1. The snmp-collector starts.
2. The configuration files and, in this case, the conf/interfacesToPoll file are read and parsed.
3. The first polling cycle begins.
4. The ifTable is polled completely, as usual, as well as the required ifXTable columns.
5. RawValues for the ifTable are generated.
6. RawValues which match the criteria described in conf/interfacesToPoll are sent to the next filter or connector. Other RawValues are discarded. The collector memorizes the table's rows oids to poll on the next polling cycle.
7. The ifTable is polled on the second polling cycle: only the required table rows are polled this time.
8. Polling proceeds as previously described until it is time to refresh the properties. The ifTable is polled completely when this occurs.



Semi Selective Table Row Polling Example

Selective Row Polling cannot be used with tables that have non-static rows index OID suffixes. This is the case with some tables that add rows over time, where at least part of the index OID suffix is stable (the N first oid in the suffix are actually static) and the rest changes over time. One such case is Cisco's IPSLA `rttMonStatsCaptureTable`, which rows index are defined as follows:

```
rttMonStatsCaptureTable.1.column.A.B.C.D
```

Where:

A: `rttMonCtrlAdminIndex`;
B: `rttMonStatsCaptureStartTimeIndex`;
C: `rttMonStatsCapturePathIndex`;
D: `rttMonStatsCaptureHopIndex`

The `rttMonStatsCaptureTable` rows indexing works as follows:

- *rttMonCtrlAdminIndex*: Uniquely identifies a probe in the `rttMonCtrlAdminTable`. This id is used in every Cisco IPSLA table and does not change over the life of the probe.

- *rttMonStatsCaptureStartTimeIndex*: The row 's creation timestamp: this changes over time as rows are continuously added to the table. This models a sliding window, in which 2 rows are kept: the current one (created within the last hour) and an older row, which age is less than 2 hours. Once the current row ages beyond one hour, a new (indexed by it's creation timestamp) row is added to the table, while the older row is deleted.
- *rttMonStatsCapturePathIndex* and *rttMonStatsCaptureHopIndex*: These index oid suffixes are used with *icmpPathEcho* probe types.

Special measures must be taken to poll this table selectively, as only part of the row's OID index is static over time. In this case, only the first part (*rttMonCtrlAdminIndex*) remains constant, while the rest changes. The collector cannot simply query the device for specific rows as they might cease to exist after the first polling cycle.

A *tableRowSpecificationFile* must be provided in order to poll this table selectively in a correct manner, as well as the length of the static part of the row's index OID. The *tableRowSpecificationFile* format and usage is provided in [Appendix D](#). In this case, the length of the static part of the row's index OID is 1 (one), therefore *matchNFirstOidSuffixOnly="1"* must be supplied in the table definition. Please note that a runtime error will occur if this value is not set correctly: this is due to the fact that the collector cannot anticipate the snmp query results and is not be able to correctly isolate the static part of the index oid if the *matchNFirstOidSuffixOnly* is not set to a usable value.

In this case, the only static part of the row's index OID is the first one: rttMonCtrlAdminIndex.
A typical snmp-table definition used to poll this type of scenario is shown here:

```
<snmp-table name="rttMonStatsCaptureTable" id="rttMSCaT" oid=".1.3.6.1.4.1.9.9.42.1.3.1"
  tableIndex=".1.100.1.101.1.102" indexEntryMask="oid.1.100.101.102+"
  nextRefTableColIdxOid=".1.100" partType="SAAPROBE" tableColNameOid=".1.100"
  cacheAllSnmpReferenceTables="false" useSnmpV2cMultipleRowPolling="true" snmpRepeaters="50"
  tableRowSpecificationFile="conf/selectedIpslaProbes.txt" matchNFirstOidSuffixOnly="1">
  <inner-join-on name="probeNumber" appendOid=".1.100" />
  <snmp-reference-table name="rttMonCtrlAdminTable.2"
    referenceTableOid=".1.3.6.1.4.1.9.9.42.1.2.1" refTableColIdxOid=".1.100"
    indexEntryMask="oid.1.100" parentTableColIdxOidPositions="1" propertyName="owner"
    propertyValueAppendOid=".1.2" longLifeCaching="true"/>
  <snmp-reference-table name="rttMonCtrlAdminTable.3"
    referenceTableOid=".1.3.6.1.4.1.9.9.42.1.2.1" refTableColIdxOid=".1.100"
    indexEntryMask="oid.1.100" parentTableColIdxOidPositions="1" propertyName="tag"
    propertyValueAppendOid=".1.3" longLifeCaching="false"/>
  <columns-and-filters>
    <snmp-column name="StatsCaptureCompletions" appendOid=".1.5"
      propertyName="statsStartTimeIndex" propertyValueAppendOid=".1.101"
      varNameGenOidSequence=".1.100"/>
    <snmp-column name="StatsCaptureOverThresholds" appendOid=".1.6"
      propertyName="statsStartTimeIndex" propertyValueAppendOid=".1.101"
      varNameGenOidSequence=".1.100"/>
    <snmp-column name="StatsCaptureSumCompletionTime" appendOid=".1.7" unit="ms"
      propertyName="statsStartTimeIndex" propertyValueAppendOid=".1.101"
      varNameGenOidSequence=".1.100"/>
  </columns-and-filters>
</snmp-table>
```

Example breakdown:

- Polled table named "rttMonStatsCaptureTable" base oid is .1.3.6.1.4.1.9.9.42.1.3.1
- Table indexes are non accessible, therefore indexEntryMask is defined as

```
''oid._._.100.101.102+''
```

- In this case, only the first 2 suboids (rttMonCtrlAdminIndex and rttMonStatsCaptureStartTimeIndex) matter as we are not polling icmpPathEcho probes, which require the other index suffixes. The tableIndex is defined as ".1.100 .1.101 .1.102". Note that the table index columns are not directly used in the RawValue variable names: only the rttMonCtrlAdminIndex is used here (specified at the snmp-column level, with the varNameGenOidSequence=".1.100" attribute).
- nextRefTableColIdxOid=".1.100" is used to join the snmp-reference-tables with the polled table. Only the first index suboid (rttMonCtrlAdminIndex) is used for this.
- tableNameOid=".1.100" is set so that the RawValue part is its rttMonCtrlAdminIndex.
- cacheAllSnmpReferenceTables="false" tells the collector not to cache any snmp-reference-table, unless otherwise specified.
- The useSnmpV2cMultipleRowPolling="true" snmpRepeaters="50" are setup to ensure data consistency through the polled data: StatsCaptureCompletions, StatsCaptureOverThresholds and StatsCaptureSumCompletionTime are polled in a "line-by-line" fashion, ensuring that they are polled atomically, in a single snmp request.
- The tableRowSpecificationFile="conf/selectedIpslaProbes.txt" trigger the Table Selective Row Polling mode, while the matchNFirstOidSuffixOnly="1" indicates that the polling would be done on rows which have their first index suffix included in the set of RawValues accepted by the "conf/selectedIpslaProbes.txt" table row specification.

When polling a table containing this data:

100	101	102	103	104	...Completions	...OverThresholds	...CompletionTime
5	1152408806	1	1	1	0	0	0
5	1152768806	1	1	1	0	0	0
6	1152721569	1	1	1	15	0	1204
6	1153081569	1	1	1	0	0	0
8	1152721569	1	1	1	15	0	1809
8	1153081569	1	1	1	0	0	0
14	1152721574	1	1	1	15	0	1107
14	1153081574	1	1	1	0	0	0
15	1152721574	1	1	1	15	2	4050
15	1153081574	1	1	1 0	0	0	

And the conf/selectedIpslaProbes.txt specifies this:

ip 10.128.3.1 part 1|8|15

The collector's output will be:

100	101	102	103	104	...Completions	...OverThresholds	...CompletionTime
8	1152721569	1	1	1	15	0	1809
8	1153081569	1	1	1	0	0	0
15	1152721574	1	1	1	15	2	4050
15	1153081574	1	1	1 0	0	0	0

Here is a sample RawValues output for this configuration (StatsCaptureCompletions only, contact and location properties were removed) :

```
1221137806: 12345::Watch4NetSnmpCollector.10.128.3.1.rttMSCaT..1.5.56({model=.1.3.6,
pollgrp=ipsla, tag=Probe8-Ping-Google, rtttype=echo, colladdr=0x40 0xE9 0xA1 0x68,
ip=10.128.3.1, devdesc=Cisco Internetwork Operating System Software ..IOS (tm) 4500,
statsStartTimeIndex=1152001569, source=Watch4NetSnmpCollector, parttype=SAAPROBE,
oid=.1.3.6.1.4.1.9.9.42.1.3.1.1.5.8.1152001569.1.1.1, name=StatsCaptureCompletions,
owner=Watch4Net, device=SwitchPrincipale, devtype=Router, part=8})=15.0
1221137806: 12345::Watch4NetSnmpCollector.10.128.3.1.rttMSCaT..1.5.56({model=.1.3.6,
pollgrp=ipsla, tag=Probe8-Ping-Google, rtttype=echo, colladdr=0x40 0xE9 0xA1 0x68,
ip=10.128.3.1, devdesc=Cisco Internetwork Operating System Software ..IOS (tm) 4500,
statsStartTimeIndex=1152361569, source=Watch4NetSnmpCollector, parttype=SAAPROBE,
oid=.1.3.6.1.4.1.9.9.42.1.3.1.1.5.8.1152361569.1.1.1, name=StatsCaptureCompletions,
owner=Watch4Net, device=SwitchPrincipale, devtype=Router, part=8})=3.0
1221137806: 12345::Watch4NetSnmpCollector.10.128.3.1.rttMSCaT..1.5.1572({model=.1.3.6,
pollgrp=ipsla, tag=Probe15-Ping-Russia, rtttype=echo, colladdr=0xC2 0x43 0x39 0x1A,
ip=10.128.3.1, devdesc=Cisco Internetwork Operating System Software ..IOS (tm) 4500,
statsStartTimeIndex=1152001574, source=Watch4NetSnmpCollector, parttype=SAAPROBE,
oid=.1.3.6.1.4.1.9.9.42.1.3.1.1.5.15.1152001574.1.1.1, name=StatsCaptureCompletions,
owner=Watch4Net, device=SwitchPrincipale, devtype=Router, part=15})=15.0
1221137806: 12345::Watch4NetSnmpCollector.10.128.3.1.rttMSCaT..1.5.1572({model=.1.3.6,
pollgrp=ipsla, tag=Probe15-Ping-Russia, rtttype=echo, colladdr=0xC2 0x43 0x39 0x1A,
ip=10.128.3.1, devdesc=Cisco Internetwork Operating System Software ..IOS (tm) 4500,
statsStartTimeIndex=1152361574, source=Watch4NetSnmpCollector, parttype=SAAPROBE,
oid=.1.3.6.1.4.1.9.9.42.1.3.1.1.5.15.1152361574.1.1.1, name=StatsCaptureCompletions,
owner=Watch4Net, device=SwitchPrincipale, devtype=Router, part=15})=3.0
```



IPv6 support

This collector fully supports IPv6. The supported IPv6 address formats include the form defined in RFC 2732 or the literal IPv6 address format defined in RFC 2373. No framing square brackets are required.

IPv6 only operation is supported: it only requires the specification of a local usable IPv6 interface address in the `<out-interface-ip-address>` element present in the `slave-snmp-poller.xml` file.

Dual stack operation is also supported when the `<out-interface-ip-address>0.0.0.0</out-interface-ip-address>` is set to either "0.0.0.0" or left empty. The kernel will then be responsible in choosing the correct interface in order to poll the target snmp agent.

In order to poll IPv6 agents, the `<snmp-agent ... ipAddress="...">` should be set either to a valid IPv6 address or a valid host name (a DNS name resolve operation is done in this case).



64-bit counter wrap around support

SNMP 64 bit counters rarely, under normal circumstances, wrap around (overflow). However, it is possible that, under certain conditions (a buggy SNMP Agent, frequent device resets, etc), the SNMP Collector will detect that the previous counter value was greater than the one just polled and assume that the counter overflowed.

This may cause large peaks to appear in graphs. Unfortunately, this has the effect of significantly modifying aggregates such as the average and maximum values.

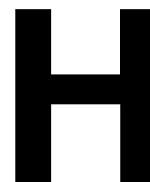
A mechanism has been added to the collector in order to better analyze the data when a 64-bit counter wrap around condition is detected. The assumptions are as follows:

- The SNMP Collector `handleCounterWrapAround` is set to true. (set in the `slave-snmp-poller.xml` file)
- a RATEP or DELTAP is used when handling the value (`handleAs="RATEP"` or `handleAs="DELTAP"`) defined in the `snmp-mask`.
- the default wrap around detection threshold factor is 1 000 000 000 000 (1e12). This is set in the `Snmp64bitCounterWrapAroundThreshold` property. This property may be set in the `*service.properties` file, located in the `<APG> /Collecting/ CollectorManager/ <Instance> / conf` directory.

A real counter wrap around is detected when the polled value delta is less than the specified `Snmp64bitCounterWrapAroundThreshold`. In this case, the delta (or rate) is sent in the generated raw value.

No value will be sent if the polled value delta is more than the specified `Snmp64bitCounterWrapAroundThreshold` value. This is to ensure aggregate correctness.

Note: The 64-bit counter wrap around threshold logic only applies to SNMP 64 bit COUNTERs declared in the MIB. Both variants of rates and deltas are affected. It does not apply to SNMP INTEGERS declared in the MIB, or if the MIB is not loaded (in which case there is no way for the collector to tell if the value is a Gauge, Integer or Counter).

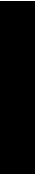


SNMP Reachability polling

The SNMP Collector outputs a reachability metric for every polled device, in every polling group. This metric, which name is usually `SnmpReachability`, is the result of a connectivity test performed at the beginning of a polling cycle. Its value will be 100 percent if the test is successful, 0 if not. The test is successful if the agent responded (for SNMP v1 and v2c agents) or if the agent's response is not a SNMP v3 error.

The SNMP Reachability polling behavior may be changed by editing the `<APG> /Collecting/CollectorManager/ <Instance>/conf/*service.properties` file and updating the collector manager service once done. The following aspects of the reachability polling behavior can be changed:

- The SNMP Reachability polling may be disabled by setting the `SnmpReachabilityEnabled` property to false. The default value is true.
- The SNMP OID used in the reachability request (`snmpGet`) can be changed by setting the `SnmpReachabilityMetricOid` property value. The default reachability OID is `.1.3.6.1.2.1.1.3.0` (`sysUp-time.0`).
- The name of the SNMP Reachability metric may be changed by setting the `SnmpReachability-MetricName` property value. The default metric name is `SnmpReachability`.



SNMP packet decoding logging

The SNMP packet decoding logging level may be set independantly from the general logging level. Be aware that this functionality may generate an extremely high volume of logging information in the collector manager .out log file if the level is set too low. This logging level may be set via the SnmpIReasoningLoggingLevel system property. This property may be set in the *service.properties file, located in the <APG> /Collecting/CollectorManager/ <Instance>/conf directory.

Available logging levels include:

- NONE: No SNMP packet decoding information is logged.
- FATAL: Only fatal conditions are logged. (SHOULD BE USED ON PRODUCTION SYSTEMS).
- ERROR: Error conditions and higher are logged.
- WARN: Warning conditions and higher are logged.
- INFO: Information conditions and higher are logged.
- DEBUG: Everything is logged. (SHOULD BE USED FOR DEBUGGING PURPOSES ONLY).