Operation Guide for Smart-Snmpd

Jens Rehsack

Contents

1 Introduction

1.1 Audience

This document targets people who have to administer smart-snmpd, either because they operate machines with smart-snmpd running or they have to configure reasonable default installations for packaging or automated deployment.

Packagers should read the INSTALL document first.

2 Operate a Smart-Snmpd Server

2.1 Set up and Run a Simple Smart-Snmpd

The build procedure of smart-snmpd creates and installs an example configuration file into `$sysconfdir` (`$prefix/etc`) named smart-snmpd.conf.example. You need to copy this file into a file named smart-snmpd.conf and modify it to fit your requirements. The task of copying this configuration file might be allocated to the post-install script of a package.

The file `smart-snmpd.conf.example` is extensively documented; it is strongly recommended you keep the comments when editing the configuration file in daily business.

Note that unspecified values are set to reasonable defaults.

2.1.1 Basic daemon settings

These settings affect all (or at least multiple) components of the `smart-snmpd`:

Table 1: Basic Settings

Setting	Data Type	Restriction	Description
`daemonize`	`boolean`	-	When set true (default), `smart-snmpd` will daemonize when started
`port`	`integer`	> 0	Specifies the port to listen on, but listen on all available interfaces / addresses
`listen-on`	`string`	must include port	Specifies the UDP address to listen on (IPv4 or IPv6 address and port)
`status-file`	`string`	-	Specifies the fully qualified path name (FQPN) to the file to store the status of the snmpd
`pid-file`	`string`	-	Specifies the FQPN to the pid-file to use
`job-threads`[1]	`integer`	> 0	Specifies the number of threads which will be used to answer snmp requests
`rlimits`	`struct`	-	Provides settings for the (soft) resource limits of the daemon for `core (RLIMIT_CORE)`, `cpu (RLIMIT_CPU)`, `data (RLIMIT_DATA)`, `filesize (RLIMIT_FSIZE)`, `files (RLIMIT_NOFILE)`, `stack (RLIMIT_STACK)` and `mem (RLIMIT_AS)`. Allowed values are: `integer` (specifying an absolute limit), "`unlimited`", "`soft`" (current set soft limit), "`hard` (current set hard limit)". If not specified, the default is "`soft`".
`on-fatal`[2]	`enum`	-	defines the action on fatal errors (`ERROR_LOG` with log-level 0) - possible values: `ignore`, `raise`, `kill` (default), `exit`, `_exit`, `abort`
`su-cmd`[3]	`string`	-	Command to use to execute an external command as a specific user
`su-args`[3]	`string`	-	Arguments for above command

[1] only with threadpool support compiled into agent++

[2] only relevant when compiled with log4cplus (handler in snmp++ always uses raise(3) sending a `SIGTERM`)

`ignore` ignore the fatal error

`raise` send the signal `SIGTERM` using raise(3)

`kill` send the signal `SIGTERM` using kill(2) on getpid(2)

`exit` exits from the currently running process using exit(3), which means some cleanup tasks are done by the runtime library before the process is reaped by the kernel

`_exit` exits from the currently running process using the _exit(2) syscall, which means the process is reaped immediately

`abort` exits from the currently running process using abort(3), which means, a core file is written and after that the process is killed by a `SIGABRT` signal

Note that the last three actions might not flush all logging messages

[3] only with libjson (bundled or separate) and --with-su-cmd enabled in configuration

Example:

```
// listen on extra port
port = 8161

// default:
// status-file = /opt/smart-snmpd/var/db/smart-snmpd/status.db

su-cmd = "/usr/bin/sudo"
su-args = {"-u", "%u", "%c"}

// how many request-threads shall run
job-threads = 24

rlimits {
    core = "unlimited" // always write core dumps on SIGSEGV
    files = 512 // usually more than enough
}

on-fatal = _exit // rigorous exiting on fatal error
```

2.1.2 Logging settings

These settings affect the logging behavior of `smart-snmpd` and are only available when logging support wasn't disabled in snmp++.

Table 2: Basic Settings

Setting	Data Type	Description
`log-file`[1]	`string`	Specifies the FQPN of the file to write logs to
`log4cplus-property-file`[2]	`string`	Specifies the FQPN of the property file to configure log4cplus
`log-profile`[3]	`string`	Specifies the log-profile to use. In addition to the 9 profiles provided by snmp++, a profile named "individual" is provided which means "Use the individual log level configuration in `log-class`".
`log-class`	`struct`	Defines each each log-level per class (`error`, `warning`, `event`, `info`, `debug`, `user`[4]) individually. Permitted values are `-1` for disabled and a threshold value from `0` to `15`.

[1] only when compiled without log4cplus (using snmp++ logging builtin)

[2] only with log4cplus

[3] only with log-profiles enabled in snmp++

[4] snmp++ log level `user` is mapped to *log4cplus'* log level `TRACE` when logging via log4cplus.

Example:

```
log4cplus-property-file = /opt/smart-snmpd/etc/log.properties

/**
 * log levels
 *
 * log classes:
 * - error
 * - warning
 * - event
 * - info
 * - debug
 * - user
 *
 * level can be from 0 .. 15 or -1 for off
 * all log entries with a log level lower or equal of the
 * configured will be shown
 *
 * defaults:
 * - 15 for error, warning (show all)
 * - 10 for event
 * -  5 for info
 * -  0 for debug, user (only most important)
 */

// reduce log level for less important log classes

log-class {
    event = 5
    info = 1
    debug = -1
}
```

2.1.3 Statgrab settings

These settings affect the wy statistics are collected with *libstatgrab*. Currently only file systems can be filtered.

Table 3: Statgrab Settings

Setting	Data Type	Description
`valid-filesystems`	`list`	Specifies a list of file system types to collect statistics for. With a "!" as first item the specified list is subtracted from the initial list of valid file systems

Example:

```
statgrab {
    // filter all kind of networking file systems to
    // avoid hanging when the server isn't available
    valid-filesystems = { "!",
                          "nfs", "nfs3", "nfs4",
                          "cifs", "smbfs", "samba"
    }
}
```

2.1.4 Basic MIB Settings

The built-in basic MIBs are setup using either the *mibobject* or *inrobject*. The *mibobject* structure initializes the basic configuration settings of all managed `smart-snmpd` MIBs:

Table 4: Basic MIB Settings

Setting	Data Type	Description
`mib-enabled`	`boolean`	Specifies whether this MIB object is enabled and can be requested or not. By default all built-in mibs are enabled.
`async-update`	`boolean`	Specifies whether this MIB object will be updated asynchronously in a background thread or synchronously every time when requested. By default a MIB is updated synchronously.
`cache-timeout`	`integer`	Time in seconds before the data cache of a MIB object becomes invalid and must be refreshed. This happens all 30 seconds by default.

Example:

```
mibobject DaemonStatus {
    // mib-enabled = false
    async-update = true
    cache-timeout = 60 /* update once a minute */
}
```

2.1.5 Interval MIB Settings

Several MIBs support differences, too. Those MIBs can be configured using the *inrobject* which contains all settings of *mibobject* and extends it with:

Table 5: Interval MIB Settings

Setting	Data Type	Description
`mr-interval`	`integer`	Time in seconds of the most-recent interval to calculate the difference. The value must be a multiple of `cache-timeout`. **Note:** If no interval is given, the value of `cache-timeout` is used which results in a diff between the most current value set and the one before.

It is strongly recommended you configure a MIB object which delivers differences to be updated asynchronously via a background thread. Without this, it is not guaranteed that the difference time slices are valid. In the case of a synchronous configuration, `smart-snmpd` will issue out a warning.

If you do not configure eg. CpuUsage using the inrobject section, but mibobject, the most recent cache interval is by default set to the value of cache-timeout. This means at least the difference from the last statistics is kept.

Example:

```
inrobject CpuUsage {
    async-update = true // strongly recommended for
                        // interval monitoring
    cache-timeout = 30  // also: measure interval base
    mr-interval = 600   // calculation time difference interval
}
```

2.1.6 Setting up External Command MIBs

`smart-snmpd` supports MIBs from external commands in addition to the planned *AgentX* support. Those external command MIBs are designed for performance - the *AgentX* protocol slows down mass data retrieves.

The external command MIBs are below `1.3.6.1.4.1.36539.20` and need to be specified for each tree separately in the configuration using the *extobject* group, which extends the *mibobject* specification as follows:

Table 6: External Command MIB Settings

Setting	Data Type	Description
`command`	`string`	Specifies the FQPN to the command to execute and the full arguments as a shell expression
`args`	`list of strings`	Specifies the arguments passed to the executed command
`user`[1]	`string`	Specifies the name of the user to switch to for command execution.
`sub-oid`	`integer`	specifies the oid below `1.3.6.1.4.1.36539.20` where the received data should be addressed.
`rlimits`	`struct`	Allows setting the current (soft) resource limits for the executed process for `core (RLIMIT_CORE)`, `cpu (RLIMIT_CPU)`, `data (RLIMIT_DATA)`, `filesize (RLIMIT_FSIZE)`, `files (RLIMIT_NOFILE)`, `stack (RLIMIT_STACK)` and `mem (RLIMIT_AS)` individually. Allowed values are: absolute limit (`integer`), `unlimited`, `soft` (soft limit at daemon start), `hard` (hard limit at daemon start).

[1] only with --with-su-cmd enabled in configuration

If you configure a MIB for an external object using mibobject instead of extobject, no external command will be known and the MIB will remain empty.

Example:

```
extobject AppMonitoring {
    async-update = true
    cache-timeout = 600 # 10 minutes
    command = /opt/smart-snmpd/libexec/plugin/appmon
    args = { "--output", "json" }
    user = "" // must be defined even if no user-switch is wanted
    sub-oid = 1
    rlimits {
        core = "soft"
        files = "soft"
    }
}
```

2.1.7 Built-In Mibs and their Names

The following built-in MIB objects need settings from ...

Table 7: Settings for built-in MIBs

MIB Object	OID	Setting Structure
DaemonStatus	1.3.6.1.4.1.36539.10.1	*mibobject*
HostInfo	1.3.6.1.4.1.36539.10.2	*mibobject*
CpuUsage	1.3.6.1.4.1.36539.10.3	*inrobject*
MemoryUsage	1.3.6.1.4.1.36539.10.4	*mibobject*
SystemLoad	1.3.6.1.4.1.36539.10.5	*mibobject*
UserLogins	1.3.6.1.4.1.36539.10.6	*mibobject*
ProcessStatus	1.3.6.1.4.1.36539.10.7	*mibobject*
FileSystemUsage	1.3.6.1.4.1.36539.10.8	*mibobject*
DiskIO	1.3.6.1.4.1.36539.10.20	*inrobject*
NetworkIO	1.3.6.1.4.1.36539.10.21	*inrobject*
SwapIO	1.3.6.1.4.1.36539.10.22	*inrobject*
AppMonitoring	1.3.6.1.4.1.36539.20.1	*extobject*

Memorize that

```
mibobject ProcessStatus {
    ...
}
```

is the same as

```
mibobject 1.3.6.1.4.1.36539.10.7 {
    ...
}
```

2.1.8 Setting up External AgentX Satelites

Future Task (Unimplemented)

2.1.9 Setting up External MIBs in Shared Libraries

Future Task (Unimplemented)

2.1.10 Setting up Access Control

`smart-snmpd` implements full *SNMPv3* access control via embedded *agent++* library. For the specification see `http://www.ietf.org/rfc/rfc3415.txt`.

To configure *SNMPv3 VACM*, the following structures should be used:

Table 8: *user*: USM Configuration

Setting	Data Type	Values	Description
auth-proto	enum	none, md5, sha	hash algorithm for auth-key
auth-key	string		the authentication key
priv-proto	enum	none, des, 3des*, idea*, aes[1], aes128, aes192*, aes256*,	encryption algorithm for priv-key
priv-key	string		the private key

* not supported by all clients

[1] alias for aes128

The security name is taken from the section title of the individual user sections specified in the configuration file.

Table 9: *group*: Group Table Configuration

Setting	Data Type	Values	Description
`security-model`	`enum`	`usm`	Currently the one and only valid security model is USM
`security-name`	`list of strings`	Name of defined user	List of users (mapped to security name) valid for this group
`storage-type`	`enum`	`other`, `volatile`, `nonvolatile`, `permanent`, `readonly`	storage type for this entry

The group name is taken from the section title of the individual group sections specified in the configuration file.

Table 10: *access*: View Table Configuration

Setting	Data Type	Values	Description
`sub-tree`	`OID`	OID[1]	Valid, managed OID or a parent of any
`mask`	`Hex Integer`	Bitmask as hexadecimal string	See ViewTreeFamily description in RFC-3415
`view-type`	`enum`	`included`, `excluded`	include or exclude this view tree
`storage-type`	`enum`	`other`, `volatile`, `nonvolatile`, `permanent`, `readonly`	storage type for this entry

[1] **OID**: **O**bject **ID**entifier - also known as a "MIB object identifier" or "MIB variable" in the SNMP network management protocol. See `http://www.pcmag.com/encyclopedia_term/0,2542,t=OID&i=48334,00.asp` or `http://www.paessler.com/support/kb/questions/49` for details.

Table 11: *access*: Access Table Configuration

Setting	Data Type	Values	Description
`group-name`	`string`	Name of defined group	Name of group whose access configuration is done here
`context`	`string`		Context name for this configuration
`security-model`	`enum`	`usm`	Currently the one and only valid security model is USM
`security-level`	`enum`	`none`, `noauth`[1,2], `nopriv`[1,2], `auth`[1,2], `priv`[1,2]	Security level for this access entry
`match`	`enum`		...
`read-view`	`string / reference`	Name of defined view	Read view for this access entry
`write-view`	`string / reference`	Name of defined view	Write view for this access entry
`notify-view`	`string / reference`	Name of defined view	Notify view for this access entry
`storage-type`	`enum`	`other`, `volatile`, `nonvolatile`, `permanent`, `readonly`	storage type for this entry

[1] when not "none" used, the specified combination must be in the order of first auth/noauth and then priv/nopriv

[2] noauth,priv is an invalid combination

2.2 Invoking the Smart-Snmpd

2.2.1 Command Line Interaction

The available command line arguments are displayed when `smart-snmpd` is called using the "-h" switch. The one which will be discussed here in detail is `-k`. Using the `-k` command line switch allows you to choose the action to run from `start` (default), `stop`, `restart`, `graceful`, `reload`, `check`, `kill`.

Table 12: Actions on command line

Action Name	Action Description
`start`	starts the *Smart-SNMP-Daemon*
`stop`	Stops the currently running *Smart-SNMP-Daemon*. Requests in the queue are answered (up to 2 seconds*).
`restart`	Restarts[1] the currently running *Smart-SNMP-Daemon*. Requests in the queue aren't answered.
`graceful`	Gracefully restarts the currently running *Smart-SNMP-Daemon*. Requests in the queue are answered (up to 2 seconds*).
`reload`	Reloads[2] the configuration of the currently running *Smart-SNMP-Daemon*
`check`	Performs a check of the configuration file of the *Smart-SNMP-Daemon*
`kill`	Kills[1] the currently running *Smart-SNMP-Daemon*

* On small powered systems with heavy load a massive tailback of requests can result in dropping requests even when restarting gracefully. This is a limitation of UDP and cannot be fixed without forcing a denial of service for new requests.

[1] sends a `SIGKILL` instead of the usually sent `SIGTERM` which should force the operating system to remove the process.

[2] Reload (via sending the signal `SIGHUP` to the daemon process) the configuration does neither affect the community/authentication entries nor log4cplus configuration due locking bug in current versions

Runtime settings, which are refreshed when the smart-snmpd catches a SIGHUP signal, are:

- log-file
- log-profile (when snmp++ is compiled with –with-log-profile)
- log-class (for log-profile = "individual")
- all mibobject, inrobject and extobject definitions
- su-cmd and su-args

2.2.2 Overriding Configuration Settings

In some circumstances it is reasonable to temporarily override some configuration settings. You can do this from the command line when starting the daemon (and only then) by specifying one or more of following options:

-f loads config from specified file

-p uses specified pid-file

-s use specified status file

-l logs into specified file

-L uses specified log-profile

An already running daemon cannot be forced to reload into the foreground (or vice versa), you need to restart the daemon with appropriate `-d` or `-D` switch.

Note: command line switches are valid until the daemon ends or gets restarted with different command line switches. There is no way to override such settings without stopping the daemon at least for a moment.

2.2.3 Requirements to start a Smart-Snmpd

The *Smart-Snmpd* has temporary and permanent requirements to operate.

The permanent requirements are:

- The directories to store the log-file, the status-file and the pid-file must exist and the operating user must have write permissions to those directories and the optional existing files from the last run.
- The file system(s) containing above mentioned directories must have enough free space to modify the named files.
- No other process must operate on the configured UDP port
- The system must have enough resources to start the configured `job-threads`, the configured `async-threads`, the configured thread for asynchronous logging and the thread for the signal handler.
- Limits for the process must be large enough:
  - at least 64MB virtual memory,
  - at least the permission to open up to 16 files - more when multiple external MIBs are configured
  - a stack size of at least 16KB

The temporary requirements are:

- The system must have enough resources (eg. free process table entries) so the `smart-snmpd` process can be forked twice.
- The system must have at least one available System V Semaphore.
- The operating user must have read permissions to the specified (either on command line or during build configuration) configuration file

2.3 Run a Smart-Snmpd in the Build Environment

Command line arguments always override configuration file settings and default behaviours as described in "Setting up and Run a Simple Smart-Snmpd" (Section 2.1).

Usually you do not want to run a local smart-snmpd using elevated privileges. That implies, the daemon can neither bind to port 161 nor modify $prefix/var/run/smart-snmpd.pid nor any other required status file (as the boot counter). It implies further, that some statistics might be incomplete or unavailable, depending on the underlying operating system.

Thus you need to override those file locations using

```
$ ./src/smart-snmpd -f `pwd`/etc/smart-snmpd-test.conf \
                    -p `pwd`/smart-snmpd.pid \
                    -s `pwd`/smart-snmpd.db \
                    -l `pwd`/smart-snmpd.log
```

3 Platform Specific Issues

3.1 Windows NT

Unimplemented.

3.2 VMS

Unimplemented.

3.3 MacOS X

Partial working.

4 Language Bindings

Future Task