**“SNMP Agent” Functional Specification Document**

Release:0.1

Date:

Status: Released

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Revision** | **Author** | **Description** |
| 27-May-2014 | 0.1 | Lina | Initial Draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Review

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Revision** | **Reviewer** | **Signature** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Important Notice**

Remego, reserves the right to make changes to the contents of this publication, including functions and specifications identified herein, without notice.

In the absence of a written agreement to the contrary, Remego assumes no liability on applications, customer’s product/application/concepts, or infringements of patents or copyrights of third parties arising from the use of systems and architectures described herein. Nor does Remego warrant or represent that any license, either expressed or implied, is granted under any patent right, copyright, or other combination of technology, architecture, or software as might be or is already in use.

**Note:** The material contained in this document is only intended for Remego and Licensed Customers with a Non-Disclosure Agreement or Standard Contract.

**The techniques described herein are proprietary and should not be reproduced, disseminated, or otherwise disclosed without prior written consent from an officer of the Remego.**

Remego has copyrighted this document.

© 2014 Remego

**Table of Content**

[About this Document v](#_Toc390254741)

[Audience for this document v](#_Toc390254742)

[Document Organization v](#_Toc390254743)

[Document Conventions v](#_Toc390254744)

[1.0 Objective 6](#_Toc390254745)

[2.0 Project Scope 7](#_Toc390254746)

[3.0 Architecture 8](#_Toc390254747)

[3.1 SNMPInitiator 9](#_Toc390254748)

[3.2 Management Information Bases (MIB) 10](#_Toc390254749)

[3.3 AlarmInitiator 11](#_Toc390254750)

[3.4 SNMPInitiator 11](#_Toc390254751)

[3.5 SNMPManager (🡸 comes from NAGIOS) 11](#_Toc390254752)

[3.6 How to use 11](#_Toc390254753)

[3.7 Config File 12](#_Toc390254754)

[3.8 Activity Log 13](#_Toc390254755)

[4.0 Scope deliverables 14](#_Toc390254756)

# About this Document

This document presents the detailed design and application. In addition, this document provides customers with a detailed outline of how the application is structured and implemented. The objectives of this document are to

* capture all customer requirements
* document design of application software for enhancement

## Audience for this document

The contents of this design specification are intended for technical personnel and for customers that have a signed contract or nondisclosure agreement with Remego. The authors of this specification assume that the reader has a working knowledge of computers and communication devices and their applications.

## Document Organization

This document details the functional requirements and features of SNMP system application in SingTel. The call flow stated in the following topic clearly identifies the modification on the current system to support this requirement.

## Document Conventions

Throughout this specification, distinctive typeface and style conventions are used to help discern the information being presented (e.g., regular text, program variables, keypad entries, vocabulary and references)

General Conventions

|  |  |
| --- | --- |
| **Convention**: | **Description**: |
| refer to Section 2.3 | Internal reference |
| **1.1 Greeting** **Section** | title and number |

# Objective

Remego, to be able to provide a highly customizable piece of software capable of sending SNMP traps conformed to the customer’s localized requirement, hereby decided to develop internally our very own “SNMP agents” ‒ SNMPInitiator.

SNMPInitiator, in addition to the capability of sending SNMP traps; it is also a centralized SNMP traps a collector that receives internally defined messages/logs from the peers’ ‒ AlarmInitiator.

Before proceeding to discuss the interaction mechanism between the SNMPInitiator and AlarmInitiator, a brief introductory too of what is AlarmInitiator. AlarmInitiator is the **notification** processes (in UNIX/Linux, processes is called daemon) runs in the servers that are being monitored. AlarmInitiator monitors the health of the server and notifies SNMPInitiator if of any alarm.

A number of rules defining what are to be monitored is defined and applied in these monitored servers. Typically these rules cover areas such as:

* Core OS baseline monitoring
* Database monitoring)
* Application component monitoring (e.g. sqlclnt, SS7 links)
* Application specific monitoring (peripro programs)

The types of thing AlarmInitiator can monitor include:

* Availability (ping responses, CPU, memory, disk space utilization, etc.)
* Logfile monitoring for specific strings (OERR in oracle log)
* Process monitoring
* Filesystem monitoring

SNMPInitiator, being the centralized collector, only need to run in a single server that has a TCP/IP connection to the centralized SNMP Manager. Also, being the only processes that interact externally, any changes in terms of support (IP changes, MIB) will be easy to change/maintain. Another feature, we can also use the SNMPInitiator to send SMS (e.g. via email-2-sms).

To leverage on the current existing mechanism in SingTel monitoring ecology that uses scripts and cronjobs that generate files, to start, AlarmInitiator will monitor directory (a concept from ActiveDirectory ‒ a proven framework). Newly generated file will trigger an SNMP trap.

In-depth technical discussion, we will explain in subsequent sections.

# Project Scope

The project scope includes

* Client program (AlarmInitiator)
* Server program (SnmpInitiator)
* Activity log file in client and server
* Config file that can be configured

# Architecture

Below illustrates the high level architecture



1. Each client has multiple health scripts. Each script will generate 1 file in a specific directory.
2. AlarmInitiator program will watch for this specific directory if there are any files generated and triggers message to SnmpInitiator.
3. SnmpInitiator accepts the message from the client and send it to SNMP manager.
4. SnmpInitiator acts as a server for multiple clients and acts as a client for the SNMP manager.

## SNMPInitiator

Prior to discussion of the SNMPInitiator, a brief introduction of the SNMP implementation in general.



The **SNMP mode**l defines two entities, which works in a client-server mode.

The SNMP server is called a SNMP agent and is located on the device to monitor. The client part is the **SNMP manager** in charge of the data collection and display.

The SNMP agent listens to requests coming from the SNMP manager on the **UDP port 161**, while the SNMP manager listens to alarms “**TRAP**” coming from the agent on **port UDP 162**. Port 161 and port 162 are the default ports. One can have a different ports number. But, it has to be agreeable by the both entities.

As illustrated, the SNMPInitiator plays the role of the **SNMP Agent**. For the full specification of a SNMP agent, refer to RFC standard. For a start, the gist of this daemon is to be able to send SNMP traps.

The project scope includes (not limited):

* Initial
  + SNMPInitiator to send SNMP trap(s) via port 162
  + SNMPInitiator to receive notification from AlarmInitiator
  + SNMPInitiator to support ad-hoc configuration
  + SNMPInitiator to read MIBs
  + SNMPInitiator to support configurable logging (log4j)
* Future
  + Support request/response from SNMP Manager (Manager is also known as **client** in SNMP v3)
  + Sends SMS (e.g. use email-to-sms)

## Management Information Bases (MIB)

MIBs describe the structure of the management data of a device subsystem; they use a hierarchical namespace containing object identifiers (OID). Each OID identifies a variable that can be read or set via SNMP. Example is listed below where the application is hunting for the “system description” SNMP variable.



To illustrate, the top of the tree consists of standards organizations: iso(1), ccitt(2), joint-iso-ccitt(3).

iso(1)node is used in this project.

iso(1)

→ org(3) for other organizations

→ dod(6)for the Department of Defense

→ internet(1) a subtree for the Internet community

→ mgmt (2)

→ mib2-1(1)

→ system(1)

→ sysDescr(1)

So, the OID for the Internet tree is 1.3.6.1, the OID for the system object group is 1.3.6.1.2.1.1, and the OID for the sysDescr object is 1.3.6.1.2.1.1.1.

When we want to actually use this OID in practice, we’ll need to tack on another number to get the value of this variable. That is, we will need to append a .0, representing the first (and only, since a device cannot have more than one description) instance of this object.

An example output:

$ **snmpget -v 1 -c public solarisbox .1.3.6.1.2.1.1.1.0**

$ **snmpget -v 1 -c public solarisbox \**

**.iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0**

These lines both yield:

system.sysDescr.0 = Sun SNMP Agent, Ultra-1

### Detail Discussion of MIB

In this section, we continue to discuss MIB definition. We also illustrate the concept with an actual working MIB file (in the final implementation, it may be different). This section is important as the main deliverable of this project is the highly customizable MIB that well fitted in SingTel environment.

As explained earlier, MIB is a structure, a well defined or spoken protocol between two entities (Manager and Agent). Below lists the MIB structure that need to be conformed to.

The structure of extended MIBS:

your-label-here DEFINITIONS ::= BEGIN

-- insert your comments here

enterprise-name OBJECT IDENTIFIER ::= { OID-label(1) OID-label{2) 3 }

subtree-name1 OBJECT IDENTIFIER ::= { OID-label(3) 4 }

subtree-name2 OBJECT IDENTIFIER ::= { OID-label(123) 56 }

data-Identifier

This is sometimes called a leaf node, node, object, or MIB.

OBJECT-TYPE

SYNTAX Integer | Counter | Gauge | DisplayString

These are just to name a few supported datatypes.

ACCESS read-only | read-write

STATUS mandatory | optional | obsolete | deprecated

For now we will always use mandatory as our STATUS.

DESCRIPTION

"

Enter Your Description Here

READ-COMMAND: /your/command/here passed1 passed2

READ-COMMAND-TIMEOUT: timeout\_in\_seconds (defaults to 3)

FILE-COMMAND: /your/file-command/here passed1 passed2

FILE-COMMAND-FREQUENCY: frequency\_in\_seconds (defaults to 10)

FILE-NAME: /your/filename/here

"

::= { parent-subtree-name subidentifier }

END

Here’s a working snmpd.extend file that contains three definitions: psZombieNum,

prtDiagExitC, and whosOnCall. It branches out from branch: remego which is an enterprise and private type.

If one walks trough; the tree starts at the

OID .1.3.6.1.4.1.1357 (we use OID of Periphonics), which is equivalent to .iso.org.dod.internet.private.enterprises.periphonics. We can organize the sub-tree in many ways. The example below, we have 2 branches beneath remego (periphonics): periphonics.sysInfo (1357.3) will hold information about the status of the system itself (psZombieNum and prtDiagExitC), and periphonics.other (1357.255) will hold additional information (whosOnCall).



MIB structure in tree format

**The actual MIB File itself:**

AlertExt DEFINITIONS ::= BEGIN

-- comments appear here behind the dashes

internet OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) 1 }

enterprises OBJECT IDENTIFIER ::= { internet(1) private(4) 1 }

periphonics OBJECT IDENTIFIER ::= { enterprises(1) 1357 }

-- Now that we have defined periphonics, let's define some objects

sysInfo OBJECT IDENTIFIER ::= { periphonics 3 }

other OBJECT IDENTIFIER ::= { periphonics 255 }

**psZombieNum** OBJECT-TYPE

SYNTAX INTEGER

ACCESS read-only

STATUS mandatory

DESCRIPTION

"Search through ps and return the number of zombies.

READ-COMMAND: VALUE=`ps -ef | grep -v grep | grep -c \<defunct\>`; echo $VALUE"

::= { sysInfo 0 }

**prtDiagExitC** OBJECT-TYPE

SYNTAX INTEGER

ACCESS read-only

STATUS mandatory

DESCRIPTION

"On Solaris, prtdiag shows us system diagnostic information. The

manpage states that if this command exits with a non-zero value,

we have a problem. This is a great polling mechanism for some

systems.

READ-COMMAND: /usr/platform/`uname -m`/sbin/prtdiag > /dev/null; echo $?"

::= { sysInfo 1 }

**whosOnCall** OBJECT-TYPE

SYNTAX OctetString

ACCESS read-write

STATUS mandatory

DESCRIPTION

"This file contains the name of the person who will be on call

today. The helpdesk uses this file. Only the helpdesk and

managers should update this file. If you are sick or unable to

be on call, please contact your manager and/or the helpdesk.

FILE-NAME: /opt/local/oncall/today.txt"

::= { other 0 }

END

The first two objects, **psZombieNum** and **prtDiagExitC**, both use the READ-COMMAND in the DESCRIPTION. This tells the agent to **execute** the named command and send any output the command produces to the NMS.

By default, the program must complete within three seconds and have an exit value of 0 (zero). To increase the timeout, add a READ-COMMAND-TIMEOUT. E.g.

READ-COMMAND: /some/fs/somecommand.pl

READ-COMMAND-TIMEOUT: 10

This tells the agent to wait 10 seconds instead of 3 for a reply before killing the process and returning an error.

3rd object, **whosOnCall**, uses a FILE-NAME in the DESCRIPTION. This tells the agent to return the **first line** of the file, program, script, etc., specified after FILE-NAME.

## AlarmInitiator

AlarmInitiator is client program which will be put in each server where the healthcheck scripts reside. It is used to watch a specific directory and check for a new file created. If there are any, it will trigger messages to SNMPInitiator.

## SNMPInitiator

SNMPInitiator is server program. Multiple clients can connect and send messages to this program. It acts as an *agent* and will send traps to SNMP Manager. An *agent* is a network-management software module that resides on a managed device.

## SNMPManager (🡸 comes from NAGIOS)

Typical SNMP uses, one or more administrative computers, called managers. SNMP manager have the task of monitoring or managing a group of hosts or devices on a computer network. Each managed system executes, at all times, a software component called an *agent (in this case, SNMPInitiator)* which reports information via SNMP to the manager. SNMP agents send traps data on the managed systems as variables. The variables accessible via SNMP are organized in hierarchies. These hierarchies, and other metadata (such as type and description of the variable), are described by Management Information Bases (MIBs).

## How to use

There will be 1 jar file, 1 config file, 1 activity log file for each client (AlarmInitiator).

How to run:

Java –jar alarminitiator.jar

There will be 1 jar file, 1 config file, 1 activity log file for the server (SnmpInitiator).

How to run:

Java –jar snmpInitiator.jar

## Config File

There will be config file dedicated for each client and server. These config files contains configuration for the directory to watch, directory for activity log, error log, configuration to turn on the debug mode.

Example content:

commonDir=C:\\Users\\User\\Documents\\Client

watchDir=C:\\Users\\User\\Documents\\Client

hostServer=192.168.1.15

debug=1

## Activity Log

Each clients and server program will write into activity log file that would need to be configured inside config file. All activities will be logged, for example ping, sending message to the server, status of sending the message.

# Scope deliverables