ONMS ONMSi 3.12.3.0

Back to the index page

ONMSI SNMP API

Table of content

SNMP Tables SNMP Traps

1. General SNMP principles

1.1. Overview

Simple Network Management Protocol (SNMP) is an UDP-based network protocol. It is mostly used in network management systems to monitor network-attached devices.

SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried (and sometimes set) by managing applications.

1.2. SNMP network

An SNMP-managed network consists of two key components:

- the agent (server): it is a network-management software module that resides on a managed device. An agent has local knowledge of management information and translates that information to or from an SNMP specific form.
- the manager (client): it is a monitoring and controlling software module, managing network devices.

1.3. Management Information Base (MIB)

SNMP itself does not define which information (which variables) a managed system should offer. Rather, SNMP uses an extensible design, where the available information is defined by management information bases (MIBs).
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.

2. ONMSi SNMP setup

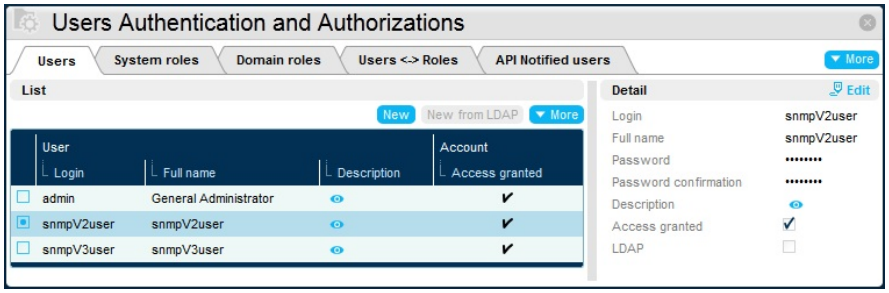
The setup process for SNMP is the following:

1. Create and configure the ONMSi user to be used as SNMP manager
2. Update the configuration files
3. Make sure the SNMP ports are opened on the firewall
4. Restart all the ONMSi services using ONMSiTools

2.1. SNMP user setup

2.1.1. Create an ONMSi user

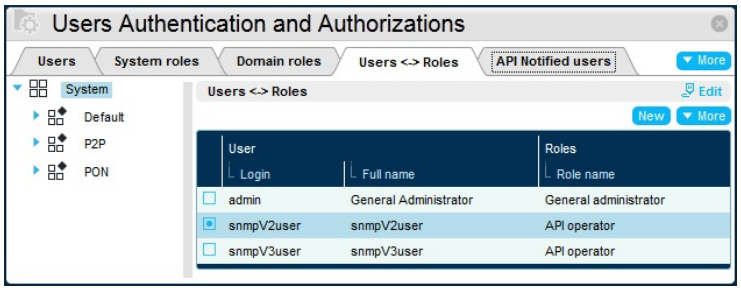
SNMP managers view the system and perform actions using the identity of an ONMSi user. Their associated ONMSi users should then be created as any other ONMSi user, with a login, a password, and permission roles on the system and/or some domains.



SNMP v2: the ONMSi user login must be the same as an SNMP v2 **community**
SNMP v3: the ONMSi user login must be the same as the SNMP **user name**

2.1.2. Setup the SNMP user privileges

Because API users tend to have administration tasks to manage, be sure to give them sufficient privileges.

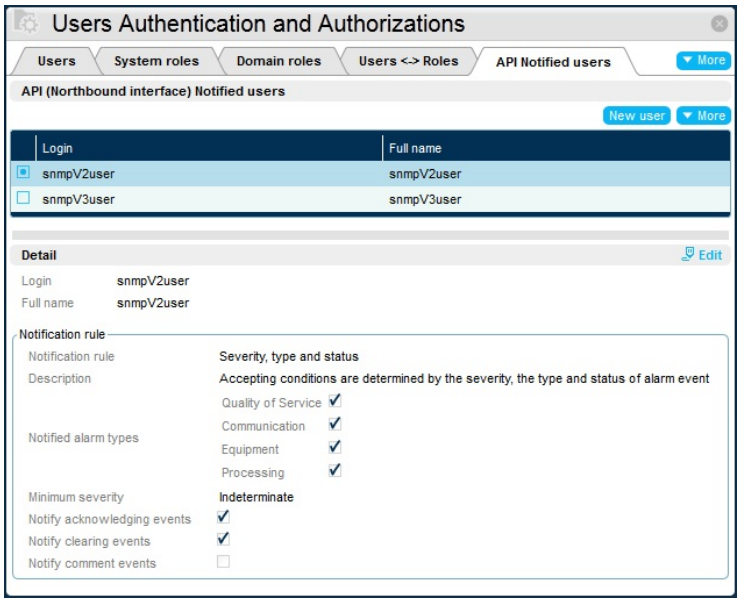


WARNING: be sure to give your API user at least the *Connect to API (northbound interface)* privilege.
You can also use the convenient *API operator* system role, which includes the following privileges:

- *Manage domains*: to manage all the domains of the system
- *Skip password policies*: to avoid security issues, such as password aging
- *Connect to API (northbound interface)*: to be able to login through the API

2.1.3. Register the user to be API notified

Add the user to the notified user list, and configure its notification rule.



2.2. Update SNMP configuration files

SNMP configuration files are located in `%TOPAZ_HOME%/jboss/standalone/topaz-conf/`:

2.2.1. jdmk.acl (V2 and V3)

Default configuration is the following:

```
acl = {
  {
  communities = snmpV2user
  access = read-write
  managers = managerV2hostname
  }
}

trap = {
  {
  trap-community = snmpV2user
  hosts = managerV2hostname
  }
#  {
#  trap-community = snmpV3user
#  hosts = managerV3hostname
#  }
}
```

This file contains 2 blocks :

2.2.1.1. acl (V2)

This block describes the **SNMP V2 manager** configuration.

You must change the community *snmpV2user* to the name of the ONMSi user created in the first step of the installation process.
You can also change *managerV2hostname* to your manager hostname/IP address.

To add another manager, please duplicate the inner block:

```
acl = {
  {
  communities = snmpV2user
  access = read-write
  managers = managerV2hostname
  }
  {
  communities = snmpV2otheruser
  access = read-write
  managers = managerV2otherhostname
  }
}
```

If you exclusively use SNMP V3, please comment the whole **acl** part:

```
#acl = {
#  {
#  communities = snmpV2user
#  access = read-write
#  managers = managerV2hostname
#  }
#}
```

2.2.1.2. trap (V2 and V3)

This block describes the **both SNMP V2 and SNMP V3** trap destination configuration.

To add other trap destinations, please duplicate the inner block:

```
trap = {
  {
```

```
  trap-community = snmpV2user
  hosts = managerV2hostname
  }
  {
  trap-community = snmpV2otheruser
  hosts = managerV2otherhostname
  }
  {
  trap-community = snmpV3user
  hosts = managerV3hostname
  }
}
```

More documentation about the ACL file format: http://docs.oracle.com/cd/E19206-01/816-4178/6madjde88/index.html

2.2.2. jdmk.uacl (V3)

This file contains describes the **SNMP V3 manager** configuration

Default configuration is the following:

```
#acl = {
#  {
#  context-names = null
#  access = read-write
#  security-level = authNoPriv
#  users = snmpV3user
#  }
#}
```

To setup the SNMP V3 manager configuration, you must uncomment the **acl** block, then change *snmpV3user* to the name of the ONMSi user created in the first step of the installation process and the **security-level** according to the jdmk.security configuration (see below):

```
acl = {
  {
  context-names = null
  access = read-write
  security-level = authNoPriv
  users = snmpV3user
  }
}
```

> **WARNING**: the *context-names* must always be **null**
> **WARNING:** be sure to match the **security-level** of the users according to the jdmk.security configuration:
>
> - **authNoPriv**: for authentication but no privacy (data encryption)
> - **authPriv**: for authentication and no privacy (data encryption)
>
> **WARNING:** **noAuthNoPriv** is not supported.

To add another V3 manager with the **same** security-level, append its username to the users attribute:

```
acl = {
  {
  context-names = null
  access = read-write
  security-level = authNoPriv
  users = snmpV3user,snmpV3otheruser
  }
}
```

To add another V3 manager with a **different** security-level, please duplicate the inner block:

```
acl = {
  {
  context-names = null
  access = read-write
  security-level = authNoPriv
  users = snmpV3user
  }
  {
  context-names = null
  access = read-write
  security-level = authPriv
  users = snmpV3otheruser
  }
}
```

More documentation about the UACL file format: http://docs.oracle.com/cd/E19206-01/816-4178/6madjde89/index.html

2.2.3. jdmk.security (V3)

> Because this file is rewritten after each SNMP agent reboot, it won't hold any documentation comment.

Default configuration is the following:

```
localEngineID=0x80008c21010a211169000000a1
localEngineBoots=0

userEntry=localEngineID,snmpV3user,snmpV3user,usmHMACMD5AuthProtocol,snmpV3password
```

You must change the *snmpV3user* and *snmpV3password* to the name and password of the ONMSi user created in the first step of the installation process.

If you intend to use different passwords for SNMP and ONMSi, you can set *snmpV3password* to any value, then override the password in snmp.properties to match your ONMSi user password (see below).

You can change the authentication to SHA algorithm:

```
localEngineID=0x80008c21010a211169000000a1
localEngineBoots=0

userEntry=localEngineID,snmpV3user,snmpV3user,usmHMACSHAAuthProtocol,snmpV3password
```

Supported authentication algorithms are:

- **usmHMACMD5AuthProtocol**: authentication using MD5
- **usmHMACSHAAuthProtocol**: authentication using SHA

> **WARNING: usmNoAuthProtocol** is not supported.

You can enable the privacy (data encryption) using the DES algorithm by appending the algorithm and its password to the entry:

```
localEngineID=0x80008c21010a211169000000a1
localEngineBoots=0

userEntry=localEngineID,snmpV3user,snmpV3user,usmHMACMD5AuthProtocol,snmpV3password,usmDESPrivProtocol,snmpV3passwordforprivacy
```

> **WARNING**: when enabling privacy, be sure to use **authPriv** in the jdmk.uacl configuration file!
> The privacy key *snmpV3passwordforprivacy* can and should be different from the authentication password *snmpV3password* (the longer, the better).

You can add users by duplicating the **userEntry**, and additionalize authentication/privacy for each user:

```
localEngineID=0x80008c21010a211169000000a1
localEngineBoots=0

userEntry=localEngineID,snmpV3user,snmpV3user,usmHMACMD5AuthProtocol,snmpV3password
userEntry=localEngineID,snmpV3otheruser,snmpV3otheruser,usmHMACSHAAuthProtocol,snmpV3otherpassword,usmDESPrivProtocol,snmpV3otherpasswordforprivac
```

More documentation about the jdmk.security file format: http://docs.oracle.com/cd/E19206-01/816-4178/snmpsecurity-131/index.html

2.2.4. snmp.properties (V2 and V3)

This files allows the additionalization of several SNMP specific parameters (agent port, trap port, buffer size, ...).
Most of them should remain to their default values, however, some must or can be changed::

2.2.4.1. snmpEnabled (mandatory for V2 and V3)

The default value is false to enforce the SNMP configuration to be performed before activating SNMP.

```
# The SNMP activation flag (true/false)
# Please uncomment and set to true to enable SNMP
#
#snmpEnabled=false
```

Uncomment this entry and set it to **true** to enable SNMP.

```
# The SNMP activation flag (true/false)
# Please uncomment and set to true to enable SNMP
#
snmpEnabled=true
```

2.2.4.2. password

Used to associated the ONMSi password of SNMP users.

```
# Configure SNMP user passwords (for V2 community support, or V3 password override)
# Please uncomment the following entry and configure it to your needs as explain in the documentation
#
#password.snmpuser=snmppassword
```

Because SNMP V2 communities don't have password, it is mandatory to uncomment and duplicate this entry for each SNMP V2 community, then configure it with you ONMSi user login (=SNMP V2 community) and ONMSi user password:

```
# Configure SNMP user passwords (for V2 user/community support, or V3 password override)
# Please uncomment the following entry and configure it to your needs as explain in the documentation
#
password.snmpV2user=snmpV2password
password.snmpV2otheruser=snmpV2otherpassword
```

Because SNMP V3 users already have passwords defined in **jdmk.security**, their authentication passwords will be used as ONMSi user password. If you want the SNMP V3 authentication password to be different from the ONMSi user password, you can override the password here, as described above for SNMP V2:

```
# Configure SNMP user passwords (for V2 user/community support, or V3 password override)
# Please uncomment the following entry and configure it to your needs as explain in the documentation
#
password.snmpV3user=snmpV3password
```

2.2.5. Multiple manager support

In order to allow for multiuser access, each SNMP V2 community/SNMP V3 user has its own instance of the MIB.
This allows secure use of functions (see below), and sequenced alarm traps.

2.3. Open SNMP ports in the firewall

ONMSi uses by default the standard SNMP ports:

- 161 for the agent (locally on the agent for the data access)
- 162 for managers (remotely on managers for trap reception).

These ports can be changed in the *snmp.properties* configuration file.

3. ONMSi MIB

3.1. Files

The ONMSi SNMP interface is defined in the JDSU-ONMSI-MIB.mib file.
This MIB depends on some other static MIBs:

- JDSU-SMI-MIB.mib: core MIB for JDSU products, defining the ONMSi root OID
- IANA-ITU-ALARM-TC-MIB.mib: IANA MIB for alarm standard conformity
- SNMPv2-SMI: structure of management information for SNMP v2
- SNMPv2-TC: the textual conventions for SMIv2

A zipped archive of all these MIBS is available for download.

3.2. Main nodes

```
+ jdsuOnmsiMib
 \
  +- jdsuOnmsiProduct
  |
  +- jdsuOnmsiAdministration
  |
  +- jdsuOnmsiServices
  |
  +- jdsuOnmsiEvents
  |
  +- jdsuOnmsiConf
```

The root of the MIB contains 5 children nodes:

- *jdsuOnmsiProduct*: contains informations about the product
- *jdsuOnmsiAdministration*: allows the ONMSi SNMP agent configuration (reload configuration, *I'm alive* trap configuration)
- *jdsuOnmsiServices*: allows resource management and operations
- *jdsuOnmsiEvents*: contains trap definitions
- *jdsuOnmsiConf*: contains conformance information

3.3. The service concept

```
+ jdsuOnmsiServices
 \
  +- jdsuOnmsiHomeService
  |
  +- jdsuOnmsiPonService
  |
  +- jdsuOnmsiPeakService
  |
  +- jdsuOnmsiLinkService
  |
  +- jdsuOnmsiMonitoringTestService
  |
  +- jdsuOnmsiAlarmService
  |
  +- jdsuOnmsiOtuService
  |
  +- jdsuOnmsiCentralOfficeService
```

Managing resources with the MIB is done through services dedicated to each object type:

- *jdsuOnmsiHomeService*: for homes
- *jdsuOnmsiPonService*: for PONs
- *jdsuOnmsiPeakService*: for PON peaks
- *jdsuOnmsiLinkService*: for links
- *jdsuOnmsiMonitoringTestService*: for monitoring tests
- *jdsuOnmsiAlarmService*: for alarms
- *jdsuOnmsiOtuService*: for OTUs
- *jdsuOnmsiCentralOfficeService*: for central offices

3.3.1. Data

Services are split into two nodes. The first node is the data node, where tables represent objects.
For objects supporting additional attributes, those are displayed in another table.

Example:

```
+ jdsuOnmsiHomeService
 \
  +- jdsuOnmsiHomeData
   \
    +- jdsuOnmsiHomeTable
    |\
    | +- jdsuOnmsiHomeEntry
    |  \
    |   +- jdsuOnmsiHomeEntryInternalKey
    |   |
    |  ...
    |   |
```

```
    |   +- jdsuOnmsiHomeEntryLatestPeakStatus
    |
    +- jdsuOnmsiHomeAdditionalAttributeTable
    |\
    | +- jdsuOnmsiHomeAdditionalAttributeEntry
    |  \
    |   +- jdsuOnmsiHomeAdditionalAttributeEntryInternalKey
    |   |
    |   +- jdsuOnmsiHomeAdditionalAttributeEntryName
    |   |
    |   +- jdsuOnmsiHomeAdditionalAttributeEntryValue
    |
    +- jdsuOnmsiHomeTerminationTypeTable
    |\
    | +- jdsuOnmsiHomeTerminationTypeEntry
    |  \
    |   +- jdsuOnmsiHomeTerminationTypeEntryInternalKey
    |   |
    |   +- jdsuOnmsiHomeTerminationTypeEntryName
    |   |
    |   +- jdsuOnmsiHomeTerminationTypeEntryDescription
    |
    +- jdsuOnmsiHomeAttributeTable
     \
      +- jdsuOnmsiHomeAttributeEntry
       \
        +- jdsuOnmsiHomeAttributeEntryName
        |
        +- jdsuOnmsiHomeAttributeEntryAdditional
        |
        +- jdsuOnmsiHomeAttributeEntryFindable
        |
        +- jdsuOnmsiHomeAttributeEntryUpdatable
```

The home data node has 2 main tables: *jdsuOnmsiHomeTable* representing home objects, and *jdsuOnmsiHomeTerminationTypeTable* representing home termination types.
The additional attributes of home objects are displayed in the *jdsuOnmsiHomeAdditionalAttributeTable* table, that combines home internal keys with home additional attribute names and values.
The noticeable attributes of home objects (such as those that are additional, can be updated or user in find functions) are displayed in the *jdsuOnmsiHomeAttributeTable* table.

You can get a more details description of SNMP tables in the ONMSi SNMP Tables page.

3.3.2. Functions

The second node of services is the functions node. Each sub-node represents a function.

Each function has 2 mandatory children nodes:

- the execute node: setting the node to 1 (integer) will execute the function
- the error node: when the function execution fails, the node contains information on the error that occurred. When the execution is successful, the node is empty.

Functions may also have optional children nodes for defining:

- parameters
- results

Example: the function used to find homes

```
+ jdsuOnmsiHomeService
 \
  +- jdsuOnmsiHomeFunctions
   \
    +- jdsuOnmsiHomeFind
     \
      +- jdsuOnmsiHomeFindParamAttribute
      |
      +- jdsuOnmsiHomeFindParamValue
      |
      +- jdsuOnmsiHomeFindExecute
      |
      +- jdsuOnmsiHomeFindError
```

There are two parameters:

- *jdsuOnmsiHomeFindParamAttribute* for the attribute on which the find is performed
- *jdsuOnmsiHomeFindParamValue* for the value to find

Because the result of the find function is a list of homes, there is no result node.
This function will clear *jdsuOnmsiHomeTable* and *jdsuOnmsiHomeAdditionalAttributeTable* to fill them with found homes.

3.4. *I'm alive* trap

In order to inform the managers that the agent is up and running, *jdsuOnmsiImAliveTrap* are sent periodically.

The period of the trap and text contained in the trap can be additionalized in *jdsuOnmsiImAlive* (globally, not per SNMP user).
This trap also contains the latest alarm event sequence, in order to help manager alarm synchronization (see below)

3.5. Alarm event synchronization

3.5.1. Alarm event sequence number

In order to prevent losing of alarm event traps, *jdsuOnmsiAlarmEventTrap* contains a *jdsuOnmsiAlarmEventEntrySequence* attribute.
That attribute holds a sequence starting from 1 and increasing after every alarm event trap.

Because every SNMP user (community) has its own notification filter, there is one sequence for each SNMP user (community).

3.5.2. Alarm event trap loss detection

The detection of lost traps can be done upon alarm event trap reception, by checking that its sequence number strictly follows the previous alarm event trap sequence number.

Should a long duration elapse between subsequent alarm event traps, another way to detect lost traps would be to check the sequence number contained in *jdsuOnmsiImAliveTrap*: it should be equal to the sequence number held in the latest alarm event trap.

3.5.3. Re-sending lost alarm event traps

When a manager detects a trap loss, it can ask the agent to re-send missing traps through the *jdsuOnmsiAlarmResendEvents* function:
Set as parameter the sequence number of the first lost alarm event trap, then execute the function.
The agent will re-send all alarm event traps with a sequence number equal to or greater than the parameter.

3.5.4. Full alarm event re-synchronization

If the manager has been shut down for a long duration, it should clear its alarm event cache, and ask for a re-synchronization through the *jdsuOnmsiAlarmResynchronize* function.

The agent will then reset the SNMP user alarm event sequence, and send alarm event traps for all events of all currently active alarms.

4. Cook book

4.1. Running a PON test

This recipe demonstrates how we can find a PON by its name, run a test on that PON and receive the result as a trap.

4.1.1. Finding a PON

We look for the PON with the name *PON 1234* through the *jdsuOnmsiPonFind* function:

```
+ jdsuOnmsiPonService
 \
  +- jdsuOnmsiPonFunctions
   \
    +- jdsuOnmsiPonFind
     \
      +- jdsuOnmsiPonFindParamAttribute <= set name
      |
      +- jdsuOnmsiPonFindParamValue <= set PON 1234
      |
      +- jdsuOnmsiPonFindExecute <= set 1
      |
      +- jdsuOnmsiPonFindError => check the attribute is empty
```

Results can be found in the *jdsuOnmsiPonTable*, and the data we need is the PON internal key.

4.1.2. Starting a PON test

Let's suppose the internal key of the PON was *123456*.
Starting the PON test will be done by using the *jdsuOnmsiPonStartTest* function:

```
+ jdsuOnmsiPonService
 \
  +- jdsuOnmsiPonFunctions
   \
    +- jdsuOnmsiPonStartTest
     \
      +- jdsuOnmsiPonStartTestParamInternalKey <= set 123456
      |
      +- jdsuOnmsiPonStartTestExecute <= set 1
      |
      +- jdsuOnmsiPonStartTestError => check the attribute is empty
```

4.1.3. Receiving the PON test result

If the PON test was successfully started, the result will be asynchronously sent through a *jdsuOnmsiPonTestResultTrap*.
Because the operation is processed asynchronously, the *jdsuOnmsiPonStartTestError* attribute of the trap should also be checked upon trap reception, to verify that the test did not fail or time out.

4.2. Running a test on demand on a link

This recipe demonstrates how we can find a link by its *_externalKey* additional attribute, find the monitoring tests on this link, start a monitoring test and receive the result as a trap.

4.2.1. Finding a link

We look for a link where the *_externalKey* additional attribute value is *LINK_1234*. This is done through the *jdsuOnmsiLinkFind* function:

```
+ jdsuOnmsiLinkService
 \
  +- jdsuOnmsiLinkFunctions
   \
```

```
    +- jdsuOnmsiLinkFind
     \
      +- jdsuOnmsiLinkFindParamAttribute <= set _externalKey
      |
      +- jdsuOnmsiLinkFindParamValue <= set LINK_1234
      |
      +- jdsuOnmsiLinkFindExecute <= set 1
      |
      +- jdsuOnmsiLinkFindError => check the attribute is empty
```

Result can be found in the *jdsuOnmsiLinkTable*, and the data we need is the link internal key.

4.2.2. Finding a monitoring test on the link

Let's suppose the internal key of the link was *123456*.
We now look for a monitoring test where the *linkInternalKey* attribute value is *123456*. This is done through the *jdsuOnmsiMonitoringTestFind* function:

```
+ jdsuOnmsiMonitoringTestService
 \
  +- jdsuOnmsiMonitoringTestFunctions
   \
    +- jdsuOnmsiMonitoringTestFind
     \
      +- jdsuOnmsiMonitoringTestFindParamAttribute <= set linkIternalKey
      |
      +- jdsuOnmsiMonitoringTestFindParamValue <= set 123456
      |
      +- jdsuOnmsiMonitoringTestFindExecute <= set 1
      |
      +- jdsuOnmsiMonitoringTestFindError => check the attribute is empty
```

Result can be found in the *jdsuOnmsiMonitoringTestTable*, and the data we need is the link internal key.

4.2.3. Starting a monitoring test

Let's suppose the internal key of the monitoring test was *1234567890*.
Starting the monitoring test is done through the *jdsuOnmsiMonitoringTestStartTest* function:

```
+ jdsuOnmsiMonitoringTestService
 \
  +- jdsuOnmsiMonitoringTestFunctions
   \
    +- jdsuOnmsiMonitoringTestStartTest
     \
      +- jdsuOnmsiMonitoringTestStartTestParamInternalKey <= set 1234567890
      |
      +- jdsuOnmsiMonitoringTestStartTestExecute <= set 1
      |
      +- jdsuOnmsiMonitoringTestStartTestError => check the attribute is empty
```

If the monitoring test was successfully started, the result will be asynchronously sent through a *jdsuOnmsiMonitoringTestResultTrap*.
Because the operation is processed asynchronously, the *jdsuOnmsiMonitoringTestStartTestError* attribute of the trap should also be checked upon trap reception, to be sure that the test did not fail or time out.

4.3. Alarm event synchronization

This recipe demonstrates how we can detect synchronization problems and fix them.

4.3.1. Synchronization problem detection

Let's suppose the last alarm event trap received had a sequence number of *123*.
If either the next alarm event trap, or the next *I'm Alive* trap, holds an alarm event sequence number higher than *124*, then some traps have been lost.

4.3.2. Synchronization fix

There are 2 ways of restoring the alarm event synchronization:

- re-send missed traps: the agent will send alarm event traps starting from the sequence number given as parameter, in chronological order.

```
+ jdsuOnmsiAlarmService
 \
  +- jdsuOnmsiAlarmFunctions
   \
    +- jdsuOnmsiAlarmResendEvents
     \
      +- jdsuOnmsiAlarmResendEventsParamSequence <= set 124 (the first missed trap sequence)
      |
      +- jdsuOnmsiAlarmResendEventsExecute <= set 1
      |
      +- jdsuOnmsiAlarmResendEventsError => check the attribute is empty
```

- re-synchronize alarms: the agent will **reset the alarm event trap sequence**, and send all alarm event trap of all currently active alarms, in chronological order.

```
+ jdsuOnmsiAlarmService
 \
  +- jdsuOnmsiAlarmFunctions
```

```
 \
  +- jdsuOnmsiAlarmResynchronize
   \
    +- jdsuOnmsiAlarmResynchronizeExecute <= set 1
    |
    +- jdsuOnmsiAlarmResynchronizeError => check the attribute is empty
```

5. SNMP testing

This last chapter will explain how to check that SNMP has been properly configured and managers will properly receive traps.

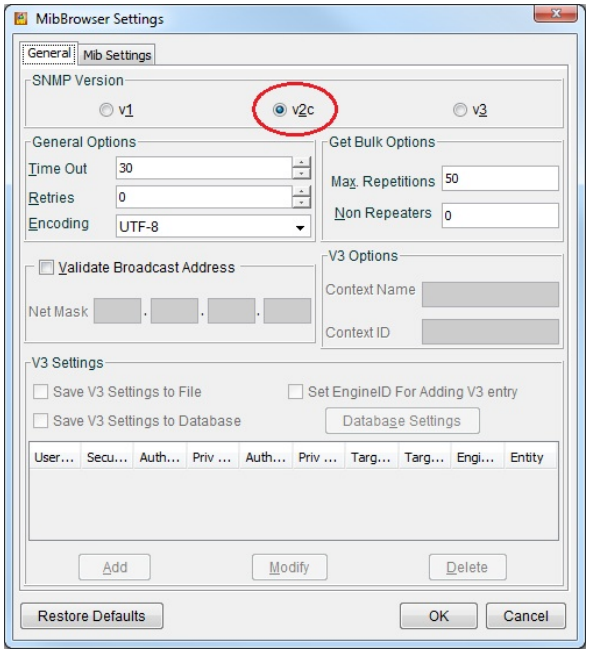
5.1. Testing tool setup

Many tools can be used to check the SNMP interface. This chapter will feature "ManageEngine MibBrowser Free Tool v5".

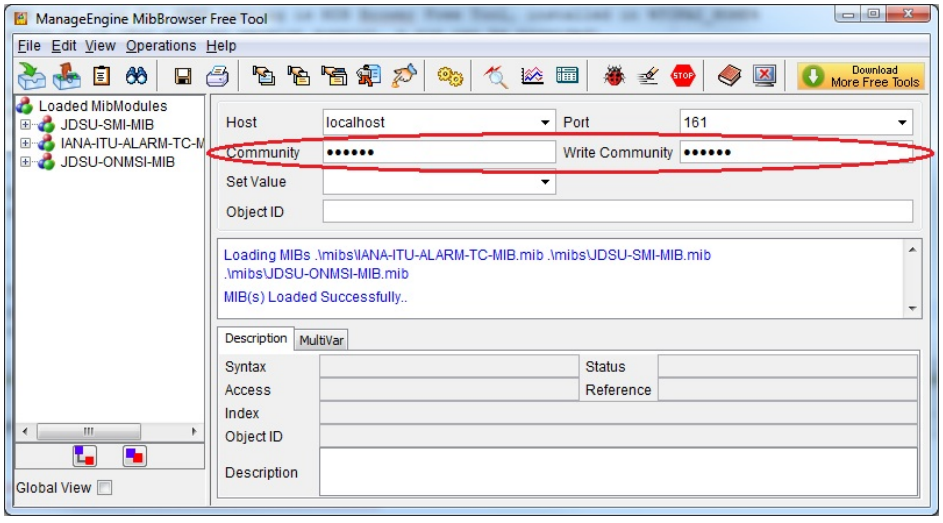
Before starting the tool, download and unzip the ONMSi MIBs into "MibBrowser Free Tool\mibs\".
Then run "MibBrowser Free Tool\MibBrowser.bat" (you may have to allow the firewall to let the tool listen traps on the port 162)

5.1.1. SNMP v2

Upon startup, the tool should already be configured for SNMP v2c. You can check it in the settings (menu Edit>Settings)

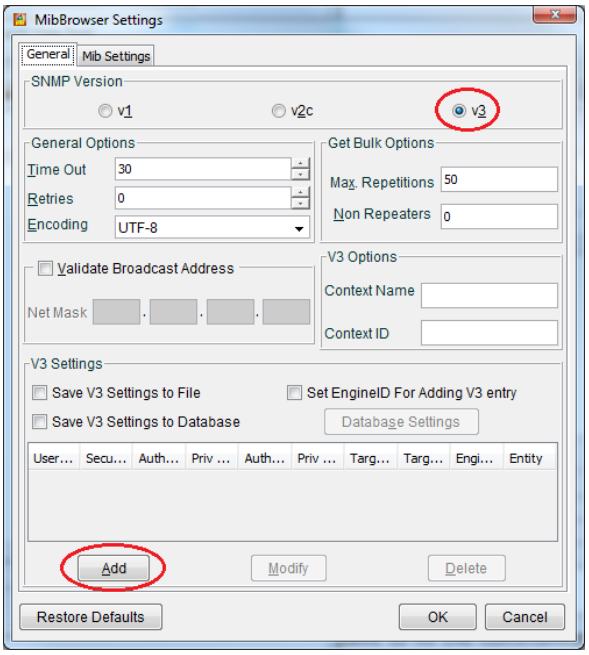


The *connection host* and *port* should already be set to "localhost" and 161. You will have to fill the *community* and *write community* attributes with the ONMSi user login (=SNMP community).



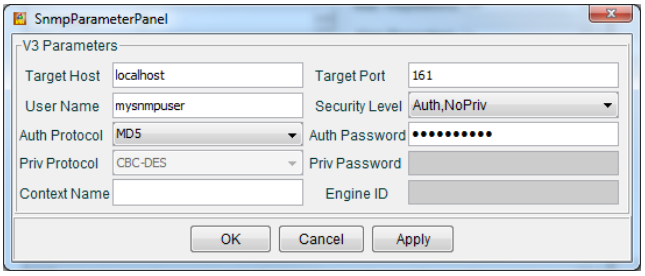
5.1.2. SNMP v3

Upon startup, the tool should be configured for SNMP v2c. You will have to configure it for SNMP v3, in the settings (menu Edit>Settings), then add an SNMP user.



WARNING: because MIB Browser Free Tool checks the user configuration with the SNMP agent, ONMSi SNMP should be up and running. You won't be allowed to add an SNMP user in MIB Free Tool if the entered configuration does not match the jdmk.security and jdmk.uacl configuration files!

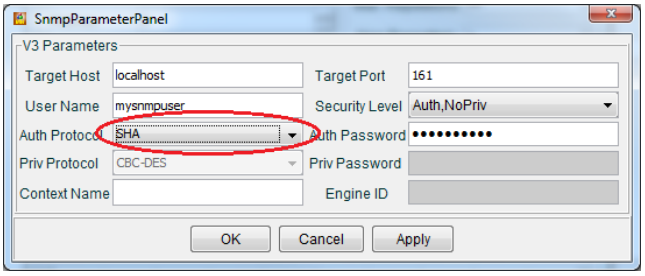
The following screenshot is for an MD5 authenticated user without privacy:



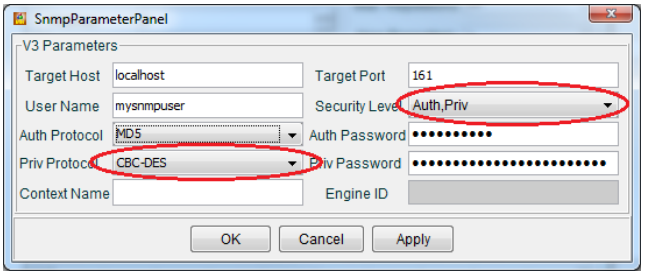
You can change *mysnmpuser* to your ONMSi user login (=SNMP user name), and *localhost* to you manager hostname/IP address.

WARNING: the context name must always be empty!

The following screenshot is for an SHA authenticated user without privacy:



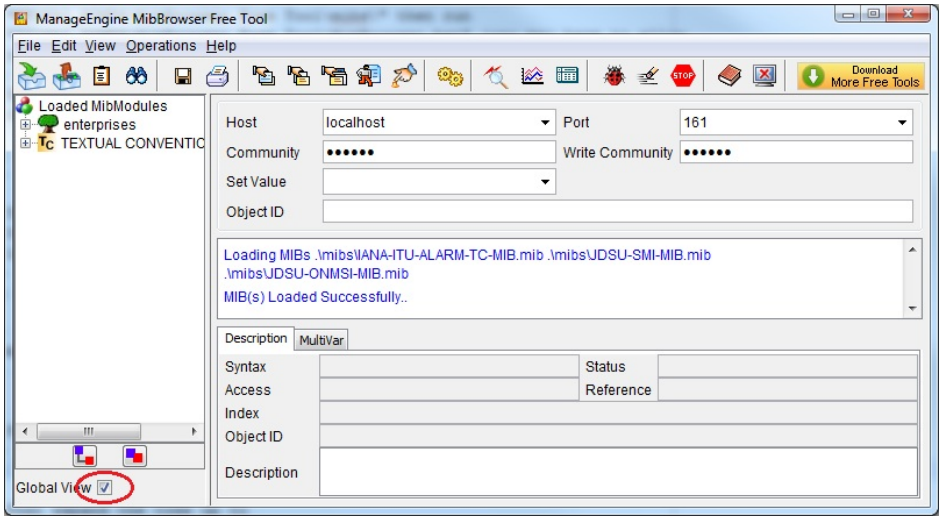
The following screenshot is for an MD5 authenticated user with DES privacy:



WARNING: **AES** algorithm for privacy is not supported.

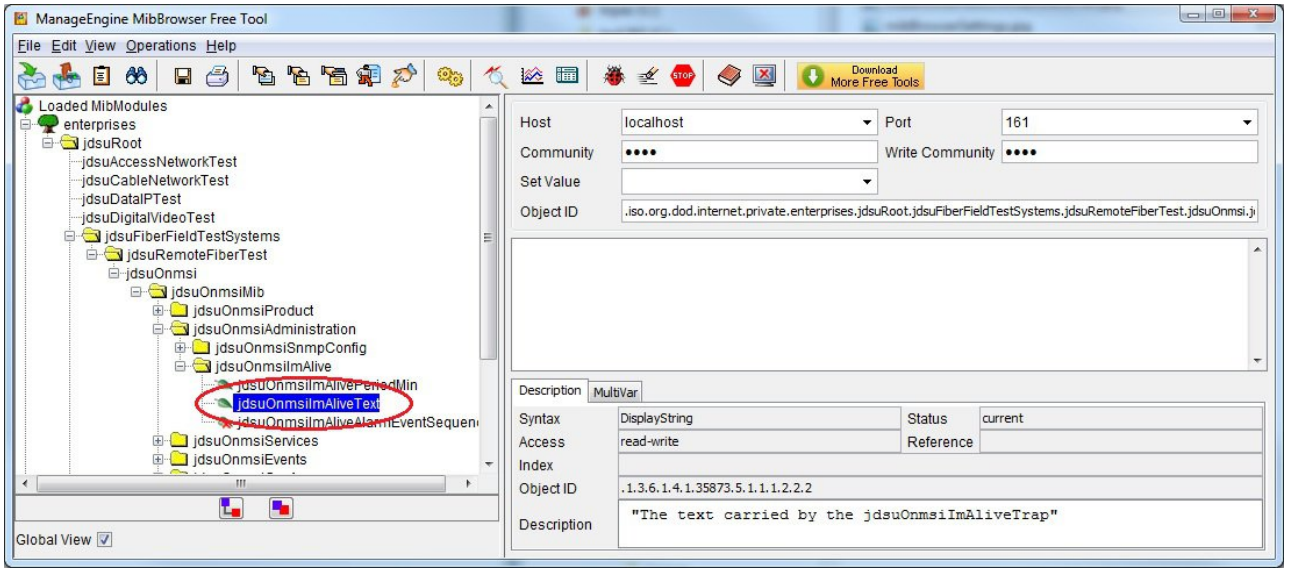
5.2. Working with the MIB

First, check the global view box, in order to see merged MIBs:

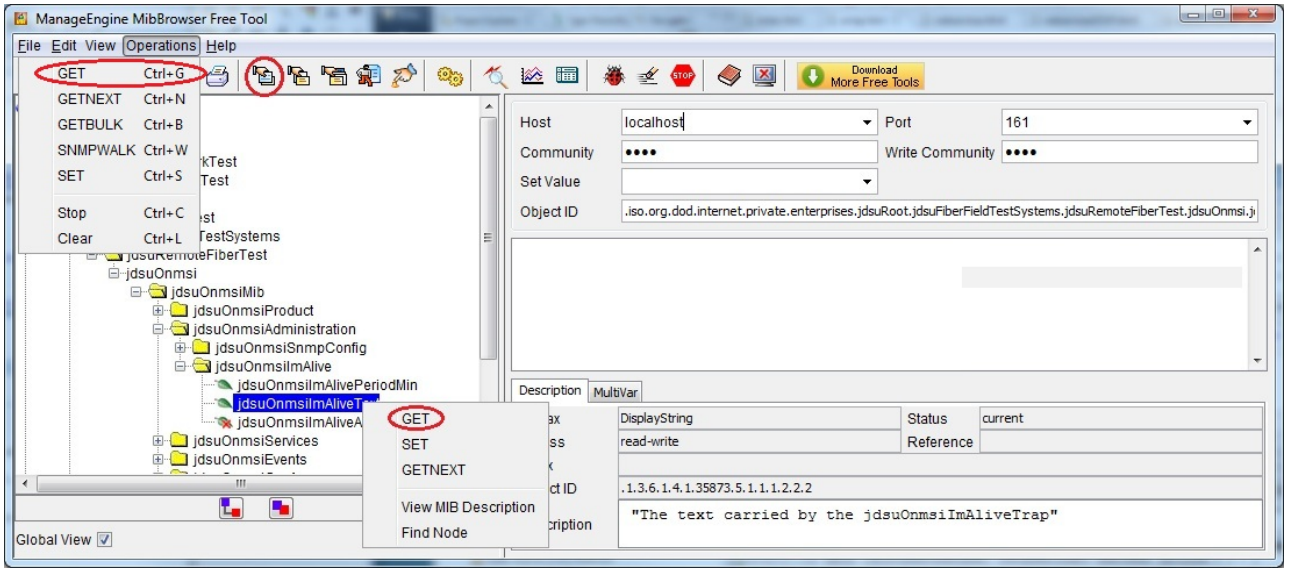


5.2.1. Get operation

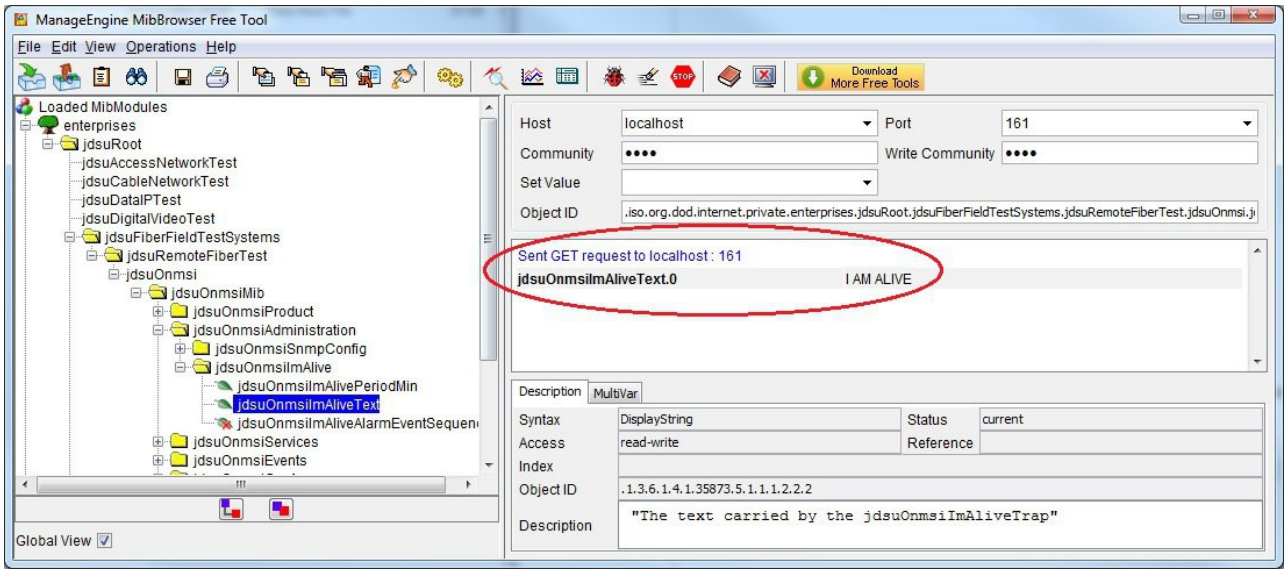
Expand the tree up to the "jdsuOnmsiImAliveText" node:



Then get the node value with the menu (Operations>GET), the toolbar button or the context menu:

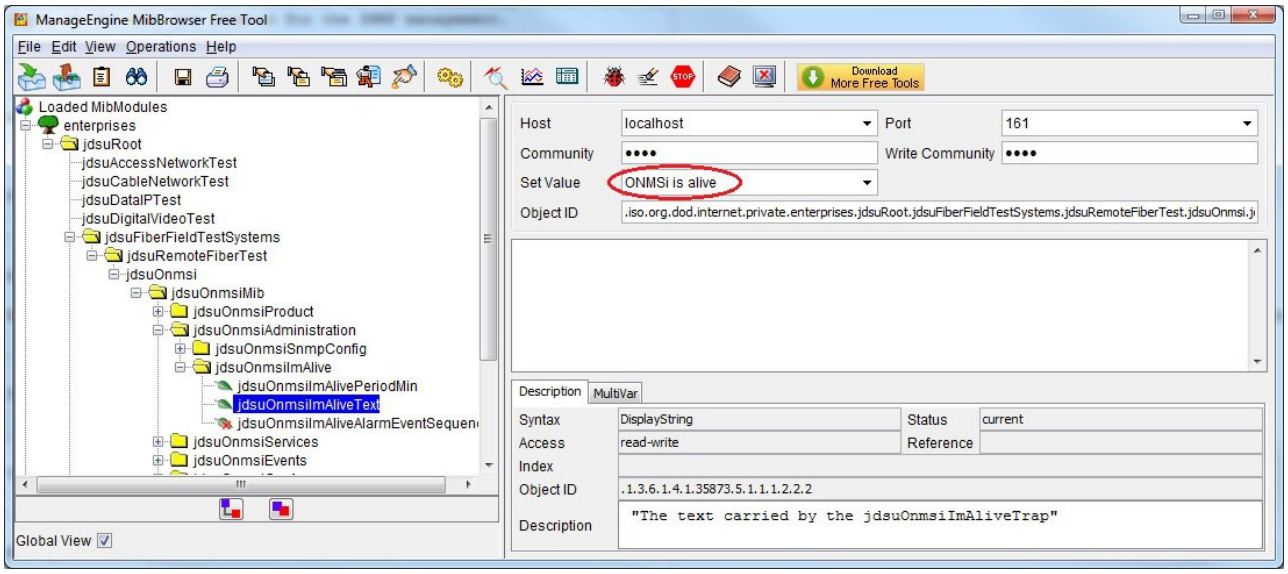


You will see the result in the result panel:

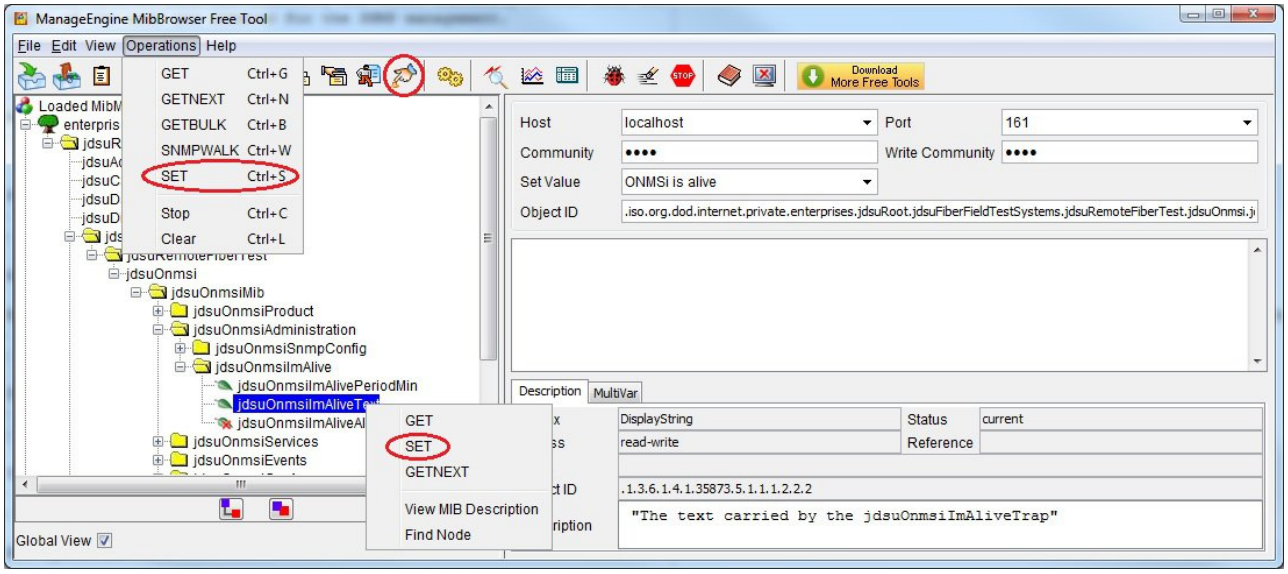


5.2.2. Set operation

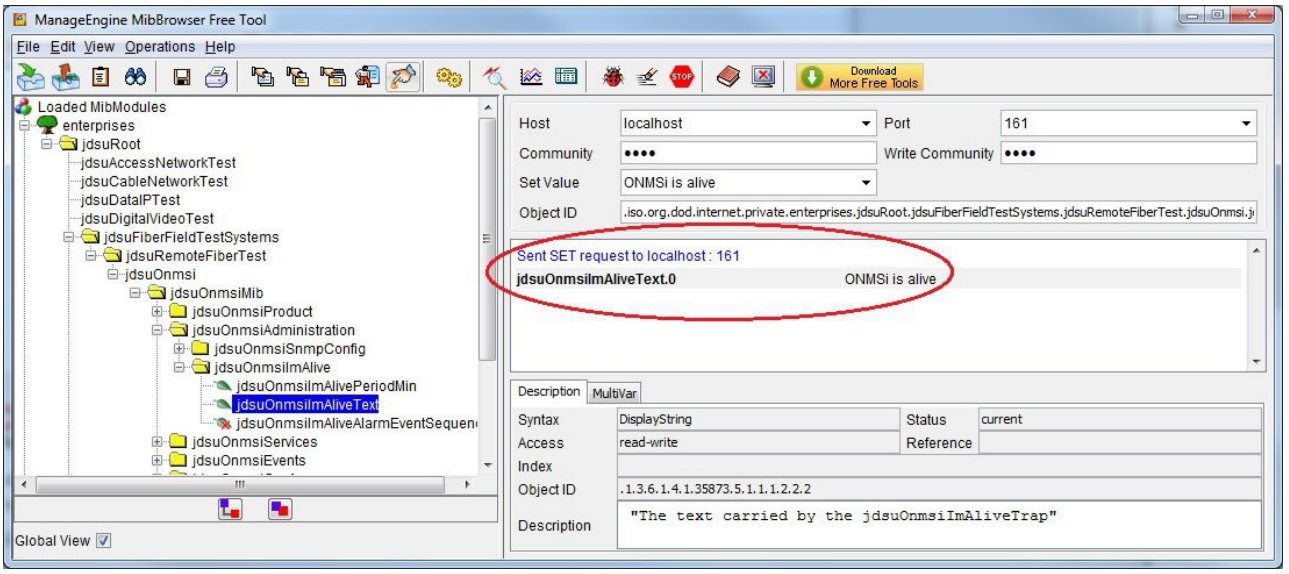
Fill the new node value in the "Set Value" attribute:



Then set the node value with the menu (Operations>SET), the toolbar button or the context menu:

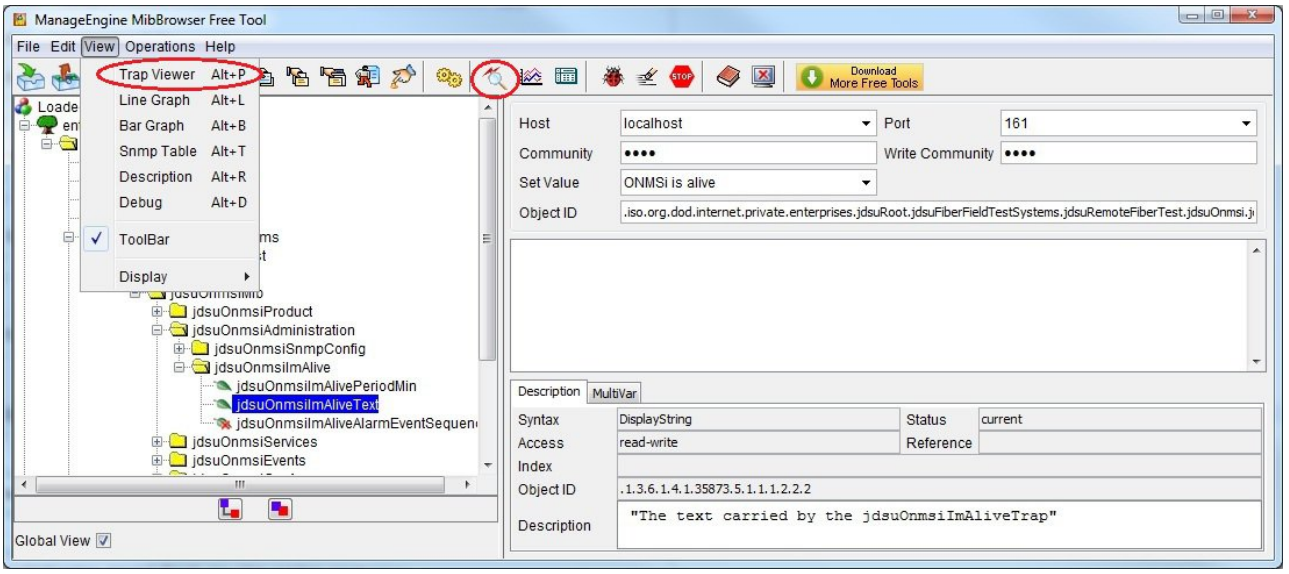


You will see the result in the result panel:



5.3. Receiving Traps

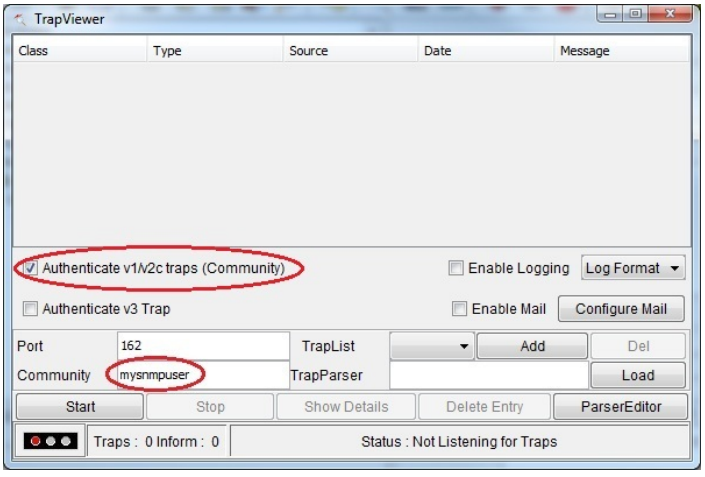
Open the trap viewer window with the menu (View>Trap Viewer) or the toolbar button:



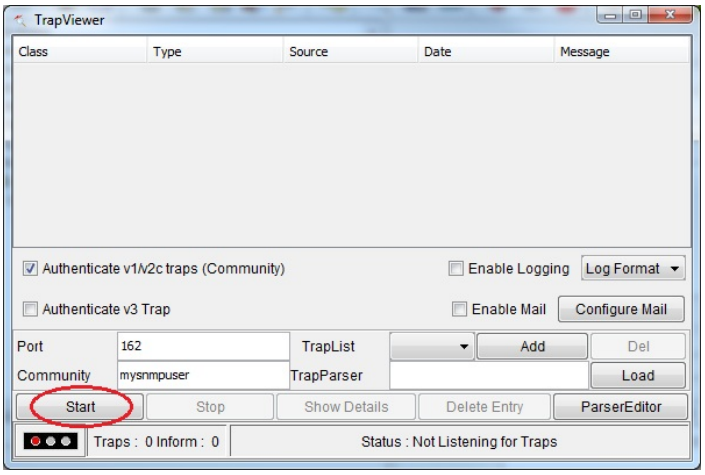
5.3.1. SNMP v2

5.3.1.1. Trap viewer setup

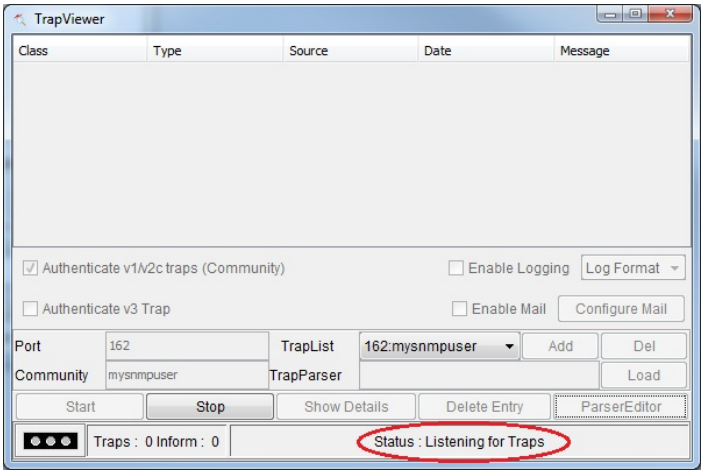
Check the *Authenticate v1/v2 traps* box and fill the *community* attribute with the ONMSi user login (=SNMP community).
Change the trap port if you modified it in snmp.properties.



Start the trap reception:

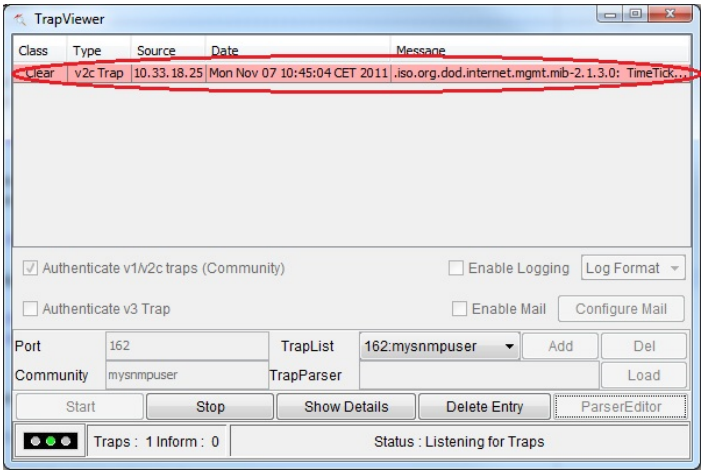


The trap viewer is now listening for traps:

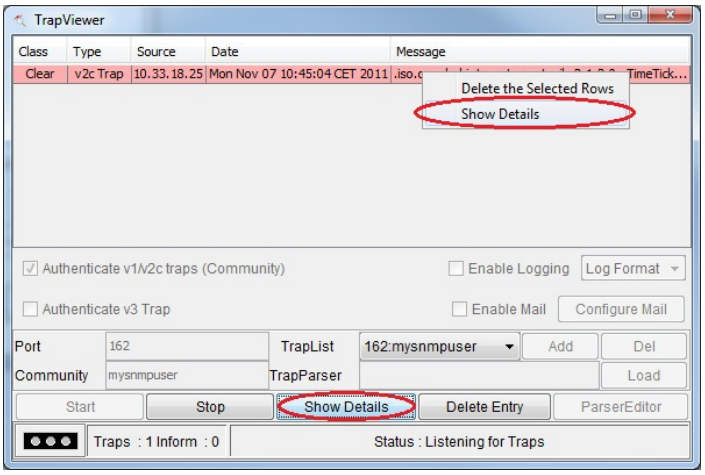


5.3.1.2. Trap reception

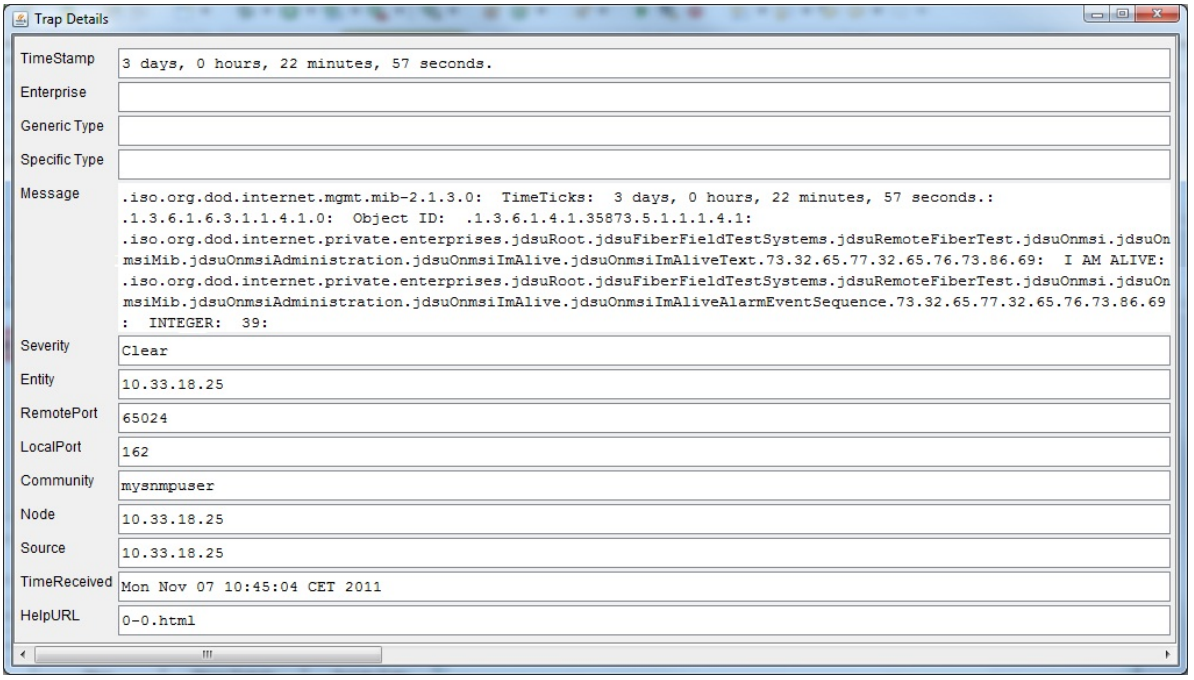
When a trap is sent by the server, such as the *I'm alive* trap, it appears in the trap viewer list:



You can get more details on the trap with the *Show Details* button:



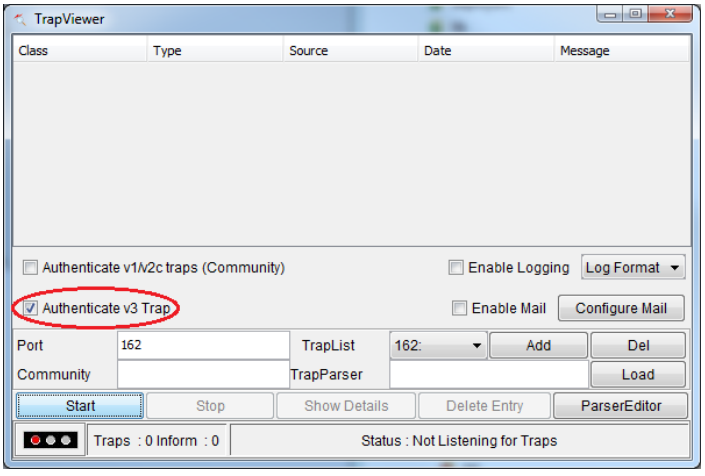
This opens a trap detail windows:



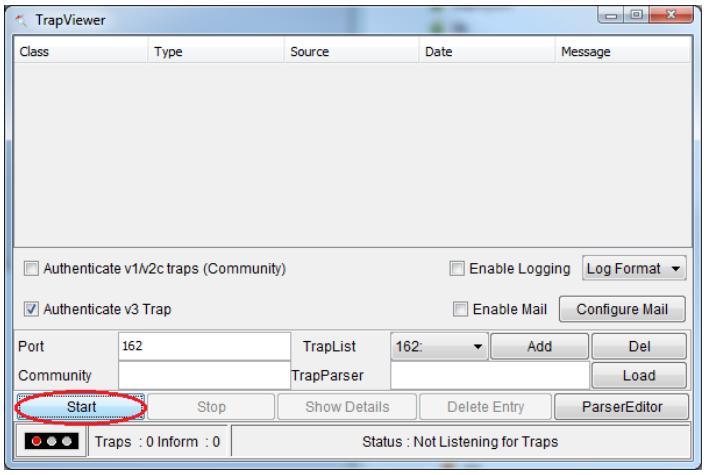
5.3.2. SNMP v3

5.3.2.1. Trap viewer setup

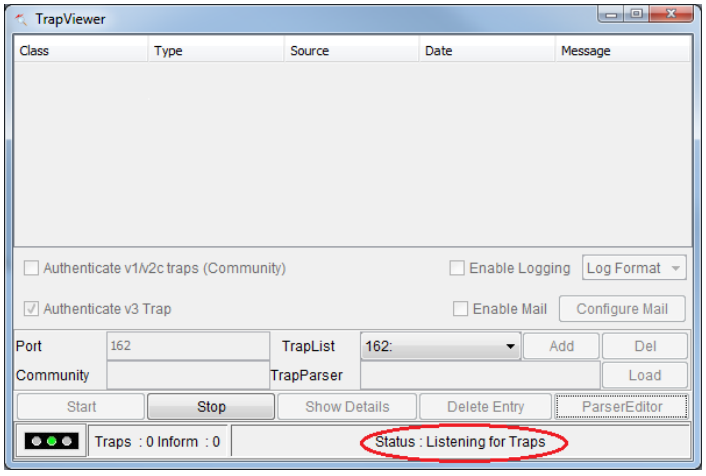
Check the *Authenticate v3 traps* box and change the trap port if you modified it in snmp.properties.



Start the trap reception:

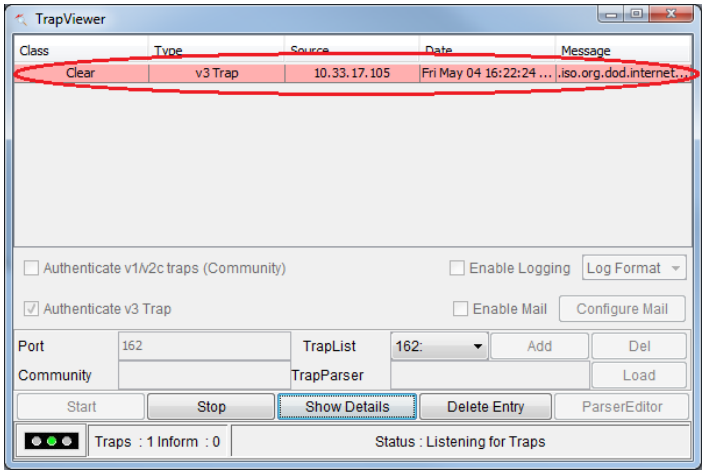


The trap viewer is now listening for traps:

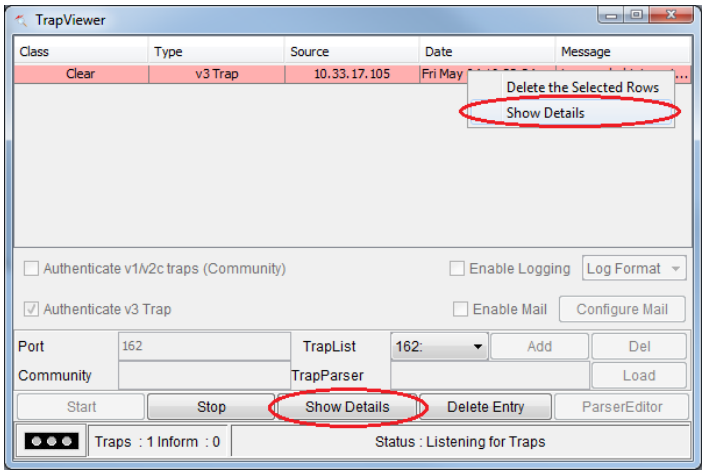


5.3.2.2. Trap reception

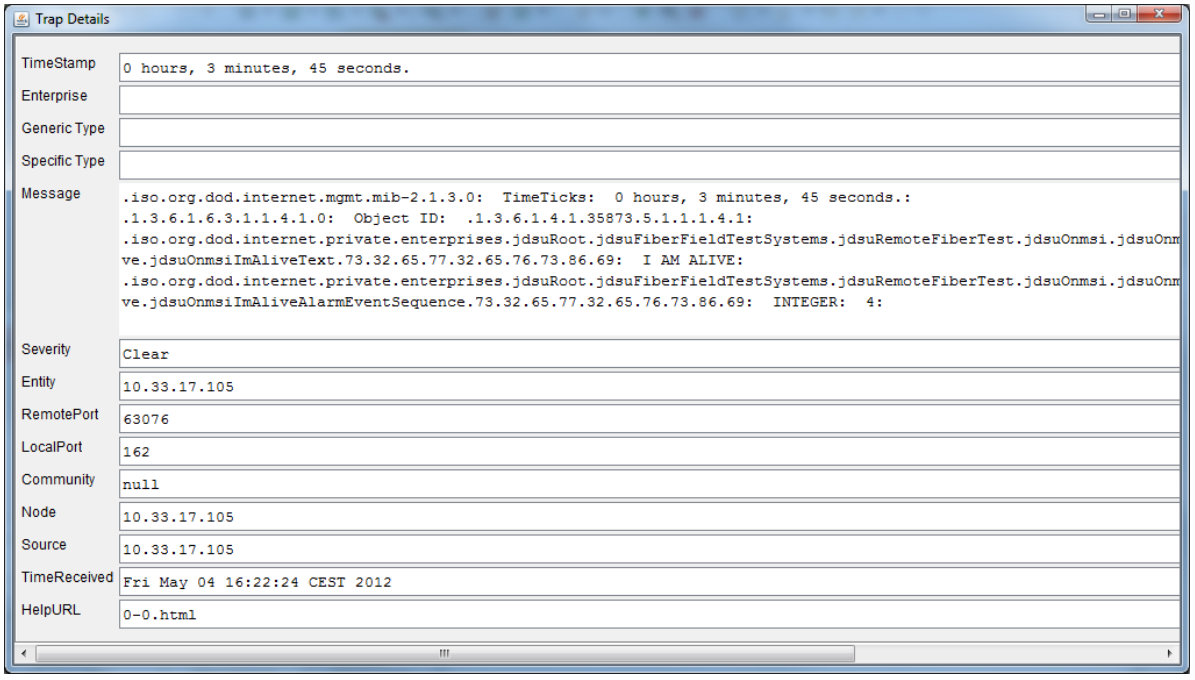
When a trap is sent by the server, such as the *I'm alive* trap, it appears in the trap viewer list:



You can get more details on the trap with the *Show Details* button:



This opens a trap detail windows:



5.3.3. Tips

Note: To help testing trap reception, you can have the server generate an *I'm alive* trap by setting a value to the *jdsuOnmsiImAlivePeriodMin* node.
Because this operation resets the *I'm alive* trap timer, you may safely set it several times to the same value (default is 5 minutes), in order to generate several *I'm alive* traps and validate your SNMP configuration.

Warning: Being able to receive *I'm alive* traps is mandatory for receiving alarm traps, but not sufficient.
If the user set up for SNMP management is not registered as an API notified user with a well configured notification rule, or has not been given proper roles on the system, you may miss important alarm traps.
Make sure to test the alarm trap reception by generating alarms on the system.

Table of content

Back to the index page