BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Network Applications and Network Administration

SNMP agent extension

October 19, 2020 Mikhail Abramov (xabram00)

Contents

Introduction

Project task

Implement the MIB module and the dynamically loadable SNMP object for net-snmp.

Project parts

The project is divided into two parts (the tasks of individual parts are presented in short form, more detailed specification of it, is possible to find in the sections corresponding to the tasks):

- **MIB module** – MIB should be register under OID .1.3.6.1.3 (iso.org.dod.internet.experimental), with its own number 22. The MIB should contain the following three SNMP objects:
  - Read-only string with xabram00 login with the OID **.1.3.6.1.3.22.1**;
  - Read-only string that returns the current time formatted according to RFC 3339 with the OID **.1.3.6.1.3.22.2**;
  - Read / write Int32 with OID **.1.3.6.1.3.22.3**;
  - Read-only string that contains information about the operating system release with the OID **.1.3.6.1.3.22.4**.
- **Dynamically loadable object** – extension for Net-SNMP as shared objects, binary files that can be loaded by the SNMPd daemon directly and are executed as part of the daemon. The extension resides in its own binary file and is loaded by the agent at runtime.

1 NET-SNMP

For the successful administration of the network, it is necessary to know the state of its elements with the ability to change the parameters of its functioning. Typically, network consists of devices from different manufacturers and it would be difficult to manage it if each of the network devices understood only its own command set. Therefore, it became necessary to create a single network resource management protocol that would be understandable by all devices, and what, therefore, would be used by all network management packages to interact with specific devices.

SNMP - Simple Network Management Protocol became it. it has become generally accepted standard for network management systems and is supported by the majority of network equipment manufacturers in their products. By virtue of its name – Simple Network Management Protocol – the main task of its development was to achieve the maximum simplicity. As a result, we have the protocol that includes a minimal set of commands, but allows us to perform almost the entire range of tasks for managing network devices – from obtaining information about the temperature of a specific device to the ability to test it.

The basic concept of the protocol is that all the information necessary to manage a device is stored on the device itself - be it a server, modem or router. Information stored in the so-called Management Information Base (MIB). MIB is a set of variables that characterize the state of a control object. These variables can reflect such parameters as the number of packets processed by the device, the state of its interfaces, the operating time of the device and etc.

Therefore, SNMP, provides small set of commands for working with MIB variables.

This set includes the following operations:

- **get** – used to request one or more MIB parameters;
- **get-next** – used to read values sequentially. Typically used to read values from tables. After requesting the first row with get-request, get-next-request is used to read the remaining rows of the table;
- **set** – used to set the value of one or more MIB variables;
- **get-response** – returns the response to the incoming request;
- **trap** – notification message about events such as cold or warm restart or a ”drop” of some connection;
- **get-bulk (SNMPv2 and SNMPv3)** – which allows a management application to retrieve a large section of a table at once;
- **notification (SNMPv2 and SNMPv3)** – standardized format of the trap;
- **inform (SNMPv2 and SNMPv3)** – tool for manager-to-manager communication;
- **report (SNMPv3)** – allow SNMP engines to communicate with each other;

How is addressing in the MIB to some of its variables? Each element has a numeric and symbolic identifier. The variable name includes the full path to it from the root element. For example, the operating time of the device since the reboot is stored in a variable located in the system section at number 3 and called sysUpTime. Accordingly, the variable name will include the entire path: iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).system(1).sysUpTime(3) or in the numbers: 1.3.6.1.2.1.1.3. It should be noted that the nodes of the tree are separated by dots. There is a standard MIB branch related to the mgmt management section that is generally supported by all network devices.

In addition, each manufacturer or organization can develop their own set of variables and put into the MIB tree. However, this is done only in a strictly defined place. If an organization develops its own MIB, then

at the stage of experiments, variables can be placed in the experimental(3) section. However, for the official registration of the data structure of some organization, it needs to get its own number in the private-enterprises section. All variables that are addressed down the branch of a given organization refer to products from that manufacturer only.

The project OID is predefined as iso(1).org(3).dod(6).internet(1).experimental(3).nstAgentPlugins(22) or .1.3.6.1.3.22. The structure of the MIB is the tree shown on the Figure (1).

Figure 1: XABRAM00-MIB OID tree

A special module called the Management Agent is used to process requests from the management station in the form of SNMP packets.

The agent receives SNMP packets and performs the corresponding actions, i.e. sends the value of the requested variable, sets the value of the variables, periodically updates the MIB information, and performs some operations in response to setting the corresponding variables.

Part 4 will reveal in more detail the principle of communication between a client and an agent using specific examples of this project.

2 Application design and Implementation

2.1 MIB module

My first aim was the design and implementation of the MIB module. After examination of the provided examples, I decided to use the following elements of the MIB module:

- **DEFINITIONS** – definition of the MIB name and MIB frame;
- **IMPORTS** – import of main objects of the module: OBJECT-TYPE, experimental – OID tree to connect our module to experiment branch, Integer32 – integer32 datatype;
- **MODULE-IDENTITY** – A brief description and update information about this mib, also here I added agent plugin number – 22;
- **OBJECT-TYPE** – main parts with objects definitions.

First of all I decided to connect MIB file to OID tree, the structure of the MIB is the tree shown on the Figure (1). To make it in the MODULE-IDENTITY part I mentioned experimental part of the branch and plugin number

```
nstAgentPlugins MODULE-IDENTITY
  LAST-UPDATED "202010150000Z"
  ORGANIZATION "BUT FIT - Mikhail Abramov"
  CONTACT-INFO "email: xabram00@stud.fit.vutbr.cz"
  DESCRIPTION "A simple MIB for ISA purposes."
  ::= { experimental 22 }
```

Next, it was necessary to define four variables:

- **nstAgentPluginLogin** – this variable should contain information about my login (xabram00) for that reason I decided to use OCTET STRING type because it specified octets of binary or textual information and supported from snmp v1. This variable has read-only access. It responses to OID: .1.3.6.1.3.22.1;

```
nstAgentPluginLogin OBJECT-TYPE
    SYNTAX OCTET STRING
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Student login"
    ::= { nstAgentPlugins 1 }
```

- **nstAgentPluginTimeInRFC3339** – this variable should contain information about the current time in the RFC3339 standard for that reason I decided to use OCTET STRING. This variable has read-only access. It responses to OID: .1.3.6.1.3.22.2;

```
nstAgentPluginTimeInRFC3339 OBJECT-TYPE
    SYNTAX OCTET STRING
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Time in RFC3339"
    ::= { nstAgentPlugins 2 }
```

- **nstAgentPluginInt32** – this variable should contain information about the int32 for that reason I decided to use Integer32 type. This variable has read-write access. It responses to OID: .1.3.6.1.3.22.3;

nstAgentPluginInt32 OBJECT-TYPE
SYNTAX Integer32 (-2147483648..2147483647)
MAX-ACCESS read-write
STATUS current
DESCRIPTION
”Read-write Int32”
::= { nstAgentPlugins 3 }

- **nstAgentPluginReleaseVersion** – the last objects responses to OID: .1.3.6.1.3.22.4 and provide information about OS Release version and it is the same as first and second objects; .

nstAgentPluginReleaseVersion OBJECT-TYPE
SYNTAX OCTET STRING
MAX-ACCESS read-only
STATUS current
DESCRIPTION
”Information about OS release version”
::= { nstAgentPlugins 4 }

2.2 Dynamically loadable object

The design and implementation of the loadable plugin is extremely simple. Main functions can be divided into four types:

- **Initialization functions** – init_”objectname” functions to upload plugin binaries to running agent. The implementation example shown on the Figure (2);

```
/*
 * Function: init_nstAgentPluginLogin
 * ----------------------
 * Initialization function to start plugin object, automatically called by the agent
 * Function should be called init_MODULE_NAME,
 * Create a scalar handler calling netsnmp_create_handler to handle incoming signals
 */
void init_nstAgentPluginLogin(void){
    DEBUGMSGTL(("nstAgentPluginLogin",
                "Initializing module\n"));
    DEBUGMSGTL(("nstAgentPluginLogin",
                "Initalizing scalar string. Default value = %s\n",
                                                    nstAgentPluginLogin_object));
    netsnmp_register_scalar(
        netsnmp_create_handler_registration("nstAgentPluginLogin",
                                            handle_nstAgentPluginLogin,
                                            nstAgentPluginLogin_oid,
                                            OID_LENGTH(nstAgentPluginLogin_oid),
                                            HANDLER_CAN_RONLY));
    DEBUGMSGTL(("nstAgentPluginLogin",
                "Done initalizing module\n"));
}
```

Figure 2: Initialization

- **Deinitialization functions** – deinit_"objectname" functions to turn off plugin from running agent. The implementation example shown on the Figure (3);

```c
/*
 * Function: deinit_nstAgentPluginTimeInRFC3339
 * --------------------
 * Unregisters a module registered against a given OID at the default priority.
 */
void deinit_nstAgentPluginTimeInRFC3339(void){
    DEBUGMSGTL(("nstAgentPluginTimeInRFC3339",
                "Deinitializing\n"));
    unregister_mib(nstAgentPluginTimeInRFC3339_oid,
                   OID_LENGTH(nstAgentPluginTimeInRFC3339_oid));
    DEBUGMSGTL(("nstAgentPluginTimeInRFC3339",
                "Done deinitalizing module\n"));
}
```

Figure 3: Deinitialization

- **Handlers functions** – handle_"objectname" functions to handle snmp requests (get or set dependes on object). For objects with read-only access, handlers for the get method are used, for only one read-write method used handler with get and set methods with all needed steps like additional value check or creation of the reserve copy. The implementation example shown on the Figure (4);

```c
/*
 * Function:  handle_nstAgentPluginTimeInRFC3339
 * --------------------
 *  Net-SNMP Agent handler
 *  http://www.net-snmp.org/dev/agent/group__handler.html#details
 *
 *  @param netsnmp_mib_handler:
 *  @param netsnmp_handler_registration:
 *  @param netsnmp_agent_request_info:
 *  @param netsnmp_request_info:
 *  @return int: ERROR number or 0
 */
int handle_nstAgentPluginTimeInRFC3339(netsnmp_mib_handler *handler,
                                       netsnmp_handler_registration *reginfo,
                                       netsnmp_agent_request_info *reqinfo,
                                       netsnmp_request_info *requests){
    returnTimeInRFC3339();
    switch (reqinfo->mode)    {
        case MODE_GET:
            DEBUGMSGTL(("nstAgentPluginTimeInRFC3339",
                        "MODE_GET: string value = %s\n",
                        nstAgentPluginTimeInRFC3339_object));
            snmp_set_var_typed_value(requests->requestvb,
                                     ASN_OCTET_STR,
                                     (u_char *)&nstAgentPluginTimeInRFC3339_object,
                                     strlen(nstAgentPluginTimeInRFC3339_object));
            break;
        default:
            DEBUGMSGTL(("nstAgentPluginTimeInRFC3339",
                        "Unknown mode (%d)\n",
                        reqinfo->mode));
            snmp_log(LOG_ERR,
                     "Unknown mode (%d)\n",
                     reqinfo->mode);
            return SNMP_ERR_GENERR;
    }
    return SNMP_ERR_NOERROR;
}
```

Figure 4: GET handler

- **Additionals functions** – two additional functions first – to generate timestamp in RFC3339 and second – to generate information about OS Release. To get OC Release version **sys/utsname.h** library was used. The implementation example shown on the Figure (5).

```
/*
 * Function:  returnSystemInformation
 * --------------------
 * Get information about OS release version (use sys/utsname.h library):
 *    example output: "3.10.0-1127.19.1.el7.x86_64"
 *
 *  @return void -> updates static output value
 */
void returnSystemInformation(){
    struct utsname detect;
    uname(&detect);
    strcpy(nstAgentPluginReleaseVersion_object, detect.release);
}
```

Figure 5: Function to get Release version

3 Instructions for Use

3.1 MIB module

To import MIB module use command **make mib** or copy MIB file into the **/usr/share/snmp/mibs/** folder. Example for use on the Figure (6).

```
[root@localhost project]# make mib
cp /home/user/VUT-ISA/project/mibs/XABRAM00-MIB.txt /usr/share/snmp/mibs/XABRAM00-MIB.txt
snmptranslate -M+. -mXABRAM00-MIB -Tp
+--iso(1)
   |
   +--org(3)
      |
      +--dod(6)
         |
         +--internet(1)
            |
            +--directory(1)
            |
            +--mgmt(2)
            |  |
            |  +--mib-2(1)
            |     |
            |     +--transmission(10)
            |
            +--experimental(3)
            |  |
            |  +--nstAgentPlugins(22)
            |     +-- -R-- String    nstAgentPluginLogin(1)
            |     +-- -R-- String    nstAgentPluginTimeInRFC3339(2)
            |     +-- -RW- Integer32 nstAgentPluginInt32(3)
            |     |        Range: -2147483648..2147483647
            |     +-- -R-- String    nstAgentPluginReleaseVersion(4)
            |
            +--private(4)
            |  |
            |  +--enterprises(1)
            |
            +--security(5)
            |
            +--snmpV2(6)
               |
               +--snmpDomains(1)
               |
               +--snmpProxys(2)
               |
               +--snmpModules(3)
```

Figure 6: MIB import

3.2 SNMP Agent

To start snmp agent with debug mode for defined downloadable plugin use the **make server** command under the root user or the direct command **snmpd -f -L -DnstAgentPluginLogin -DnstAgentPluginTimeInRFC3339 -DnstAgentPluginInt32 -DnstAgentPluginReleaseVersion,dlmod**. Example for use on the Figure (7).

```
[root@localhost project]# make server
snmpd -f -L -DnstAgentPluginLogin -DnstAgentPluginTimeInRFC3339 -DnstAgentPluginInt32
-DnstAgentPluginReleaseVersion,dlmod
registered debug token nstAgentPluginLogin, 1
registered debug token nstAgentPluginTimeInRFC3339, 1
registered debug token nstAgentPluginInt32, 1
registered debug token nstAgentPluginReleaseVersion, 1
registered debug token dlmod, 1
dlmod: register mib
dlmod: dlmod_path: /usr/lib64/snmp/dlmod
NET-SNMP version 5.7.2
```

Figure 7: Started agent

3.3 Dynamically loadable object

Now we can upload extension to running agent and control logs not only on the client side but also on the agent side. First off all we need to build objects – to do it use command **make** or **make build**. Example for use on the Figure (8).

```
[user@localhost project]$ make
gcc -I. `net-snmp-config --cflags` -fPIC -shared -c -o AgentPluginObject.o AgentPluginObject.c
gcc -I. `net-snmp-config --cflags` -fPIC -shared -o AgentPluginObject.so AgentPluginObject.o
[user@localhost project]$ ll
total 176
-rw-rw-r--. 1 user user 20306 Oct 17 15:27 AgentPluginObject.c
-rw-rw-r--. 1 user user  3695 Oct 17 15:27 AgentPluginObject.h
-rw-rw-r--. 1 user user 86784 Oct 17 17:47 AgentPluginObject.o
-rwxrwxr-x. 1 user user 50968 Oct 17 17:47 AgentPluginObject.so
-rw-rw-r--. 1 user user  5215 Oct 17 15:27 Makefile
drwxrwxr-x. 2 user user    30 Oct 17 15:27 mibs
-rw-rw-r--. 1 user user  1633 Oct 17 15:27 README.md
```

Figure 8: Build logs

Now we have binary file of downloadable object. To put it into the running agent use command **make deploy** or directly sequence of commands (example for nstAgentPluginLogin into the /home/user/VUT-ISA/project location:

- snmpset localhost UCD-DLMOD-MIB::dlmodStatus.1 i create
- snmpset localhost UCD-DLMOD-MIB::dlmodName.1 s ”nstAgentPluginLogin” UCD-DLMOD-MIB::dlmodPath.1 s ”/home/user/VUT-ISA/project/AgentPluginObject.so”
- snmpset localhost UCD-DLMOD-MIB::dlmodStatus.1 i load

On the agent terminal we can see logs output with debug information.

3.4 Communication examples

For communication examples I prepared two types of manual tests one for local test from user terminal you can start it with the command **make test** and one for remote test with the command **make remote-test**. Or directly through get or set commands. Example of the local test with OIDs you can find on the Figure (9).

Or you can use command with MIB name and the Plugin name example is on the Figure (10).

```
[user@localhost project]$ snmpget localhost .1.3.6.1.3.22.1.0 .1.3.6.1.3.22.2.0
.1.3.6.1.3.22.3.0 .1.3.6.1.3.22.4.0
SNMPv2-SMI::experimental.22.1.0 = STRING: "xabram00"
SNMPv2-SMI::experimental.22.2.0 = STRING: "2020-10-17T18:11:29-04:00"
SNMPv2-SMI::experimental.22.3.0 = INTEGER: 0
SNMPv2-SMI::experimental.22.4.0 = STRING: "3.10.0-1127.19.1.el7.x86_64"
[user@localhost project]$ snmpset localhost .1.3.6.1.3.22.3.0 i 666
SNMPv2-SMI::experimental.22.3.0 = INTEGER: 666
```

Figure 9: SNMP requests through OID

```
[user@localhost project]$ snmpget localhost XABRAM00-MIB::nstAgentPluginLogin.0 XABRAM
00-MIB::nstAgentPluginTimeInRFC3339.0 XABRAM00-MIB::nstAgentPluginInt32.0 XABRAM00-MIB
::nstAgentPluginReleaseVersion.0
XABRAM00-MIB::nstAgentPluginLogin.0 = STRING: "xabram00"
XABRAM00-MIB::nstAgentPluginTimeInRFC3339.0 = STRING: "2020-10-17T18:12:39-04:00"
XABRAM00-MIB::nstAgentPluginInt32.0 = INTEGER: 666
XABRAM00-MIB::nstAgentPluginReleaseVersion.0 = STRING: "3.10.0-1127.19.1.el7.x86_64"
[user@localhost project]$ snmpset localhost XABRAM00-MIB::nstAgentPluginInt32.0 i 1234
XABRAM00-MIB::nstAgentPluginInt32.0 = INTEGER: 1234
```

Figure 10: SNMP requests through MIB

Additionaly I tested both types of connection (oid/mib) from the remote manager with turned off firewall. Examples of communication between client and agent provided on Figures (11) and (12).

```
snmpget -v 2c -c public 192.168.56.4 .1.3.6.1.3.22.1.0 .1.3.6.1.3.22.2.0 .1.3.6.1.3.22.3.0 .1.3.6.1.3.22.4.0
SNMPv2-SMI::experimental.22.1.0 = STRING: "xabram00"
SNMPv2-SMI::experimental.22.2.0 = STRING: "2020-10-17T18:28:49-04:00"
SNMPv2-SMI::experimental.22.3.0 = INTEGER: 1234
SNMPv2-SMI::experimental.22.4.0 = STRING: "3.10.0-1127.19.1.el7.x86_64"

snmpset -v 2c -c public 192.168.56.4 .1.3.6.1.3.22.3.0 i 666
SNMPv2-SMI::experimental.22.3.0 = INTEGER: 666

snmpget -v 2c -c public 192.168.56.4 .1.3.6.1.3.22.3.0
SNMPv2-SMI::experimental.22.3.0 = INTEGER: 666
```

Figure 11: Requests and responses on client side

```
Connection from UDP: [192.168.56.1]:34566->[192.168.56.4]:161
nstAgentPluginLogin: MODE_GET: string value = xabram00
nstAgentPluginTimeInRFC3339: MODE_GET: string value = 2020-10-17T18:28:49-04:00
nstAgentPluginInt32: MODE_GET: int32 value = 1234
nstAgentPluginReleaseVersion: MODE_GET: string value = 3.10.0-1127.19.1.el7.x86_64
Connection from UDP: [192.168.56.1]:58314->[192.168.56.4]:161
nstAgentPluginInt32: MODE_SET_RESERVE1: check value type
nstAgentPluginInt32: MODE_SET_RESERVE2: reserve value = 1234
nstAgentPluginInt32: MODE_SET_ACTION: new int32 value = 666
nstAgentPluginInt32: MODE_SET_ACTION: new int32 value = 666
nstAgentPluginInt32: MODE_SET_COMMIT: skip
nstAgentPluginInt32: MODE_SET_FREE: skip
Connection from UDP: [192.168.56.1]:36360->[192.168.56.4]:161
nstAgentPluginInt32: MODE_GET: int32 value = 666
```

Figure 12: Debug on the agent side

References

[1] MAURO, D. R. a K. J. SCHMIDT. Essential SNMP. Cambridge: O Reilly, 2001. ISBN 9780596000202.

[2] Klyne, G.; Newman, C.: Date and Time on the Internet: Timestamps. July 2002, [online].

`http://www.ietf.org/rfc/rfc3339.txt`

[3] Agent Architecture - Net-SNMP Wiki. Net-SNMP [online]. Available on:

`http://www.net-snmp.org/wiki/index.php/Agent_Architecture`

[4] Net-SNMP. Net-SNMP [online]. Available on:

`http://www.net-snmp.org/tutorial/tutorial-5/toolkit/mib_module/`

[5] TUT:Writing a Dynamically Loadable Object - Net-SNMP Wiki. Net-SNMP [online]. Available on:

`http://www.net-snmp.org/wiki/index.php/TUT:Writing_a_Dynamically_Loadable_Object`