Technical Report

NuDesign SNMPv3 Agent Service for Windows

Sending Notifications from SNMP Extension Agent DLLs

NuDesign Technologies, Inc.

Table of Contents

1. Sending Notifications from SNMP Extension Agent DLLs

Microsoft's SNMP Agent Service provides a mechanism for sending SNMPv1 traps on behalf of the SNMP Extension Agent DLLs. The mechanism is simple. Extension Agent DLLs that generate traps must create a Win32 Event object to communicate occurrence of traps to the Agent Service. The event handle is given to the Agent when the Agent is initialized, and it should be NULL for traps not to be generated.

Now, when Extension Agent DLL decides to send traps, it notifies Agent by raising Event whose handle is passed to the Agent during initialization. On receiving such event, Agent will repeatedly call **SnmpExtensionTrap** method until FALSE is returned, indicating that all outstanding traps have been processed. Parameter list for the SnmpExtensionTrap provides for the info required for sending v1 traps.

NuDesign's SNMPv3 Agent Service for MS Windows provides mechanism that enables Extension Agent DLLs to send other types of notification PDUs: v2 traps and informs. It does not just simply send inform requests on behalf of the Extension Agent DLLs, it also provides a callback mechanism for passing information on results of the inform request: response, timeout or error.

Note that the SNMP agent in **NuDesign SNMPv3 Agent Service (`NDMPAgent`)** could run in SNMPv1, SNMPv2 or SNMPv3 mode.

When running in SNMPv1 mode, `NDMPAgent` responds to SNMPv1 requests only and it will send only v1 traps.

In SNMPv2 mode, `NDMPAGENT` responds to SNMPv1 and SNMPv2 requests, and sends v1Traps, v2 traps, and informs (depending on the notification destination).

And finally, in SNMPv3 mode, `NDMPAgent` processes SNMPv1, SNMPv2 and SNMPv3 requests, and sends v1Traps, v2 traps and informs (depending on the notification destination).

To take advantage of sending traps and informs from the Extension Agent DLL, developer should implement and export from the DLL all methods prototyped in the "`NDSnmpExtension.h`" header file. Note that NuDesign extension function for notifications will be called only if `NDMPAgent` is started in SNMPv2c or SNMPv3 mode.

In other words, NuDesign extension for sending v2traps and informs does not work in SNMPv1, which makes sense since SNMPv1 does not support these PDUs.

If those exports are missing (i.e. not implemented) NDMPAgent will still send notifications on behalf of the DLL, but in this case only traps. The info about trap will be retrieved by repeatedly calling SnmpExtensionTrap method until FALSE is returned. Note that even though info for the v1Trap is provided by extension DLL in this case, NDMPAgent will convert the trap to v2trap if agent is running in SNMPv2 or SNMPv3 mode. No informs will be sent, i.e. notification that are configured as informs will be sent as traps.

The mechanism for communicating occurrence of traps/informs to the NDMPAgent is similar to the one used for traps in Microsoft's agent. It is Win32 Event object whose handle is passed to the agent in NDSnmpExNotifInit. NDSnmpExNotifInit is the first function that will be called by NDMPAGENT and extension DLL should supply the handle of the Win32 Event object that will be used later for announcing notifications.

```
BOOL WINAPI NDSnmpExNotifInit( HANDLE* phSaNotifEvent )
```

From now on, we'll assume that NDSnmpExtension methods are implemented in Extension Agent DLL. When Extension Agent DLL decides to send notifications, it notifies NDMPAgent by raising Event whose handle is passed to the NDMPAgent during initialization. On receiving such event, NDMPAgent will repeatedly call **NDSnmpExNotification** method until FALSE is returned, indicating that all outstanding notifications have been processed. Here is the prototype for the NDSnmpExNotification:

```
BOOL WINAPI NDSnmpExNotification(
        AsnObjectIdentifier* pTrapOid,
            SnmpVarBindList* pVarBindList,
        ENDSnmpExInformInfo* eInformInfoRequested
);
```

All parameters are supposed to be filled by extension DLL.

The first parameter, **pTrapOid**, is pointer to an AsnObjectIdentifier structure to receive the identifier value for the snmpTrapOID.0. The SNMP Service does not free the memory for this variable.

The second parameter, **pVarBindList**, is pointer to the variable bindings list. The Extension Agent must allocate the memory for this parameter. The NDMPAgent frees the memory with a call to the SnmpUtilVarBindListFree function.

The last parameter, **eInformInfoRequested**, is of type ENDSnmpExInformInfo:

```
enum ENDSnmpExInformInfo
{
   eNDSnmpExInformInfo_nothing    = 1,
   eNDSnmpExInformInfo_requestIds = 2,
   eNDSnmpExInformInfo_targets    = 3
};
```

This parameter tells NDMPAgent which info to provide back to the Extension Agent. Setting this parameter to **eNDSnmpExInformInfo_nothing** indicates that extension DLL is not interested in what happens with inform requests, so any response, timeout or error on inform request will simply be discarded.

If extension DLL wants to know the actual destinations where informs are sent, it should set this parameter to **eNDSnmpExInformInfo_targets**. After NDMPAgent sends informs it will call NDSnmpExInformInfoTargets method:

```
VOID WINAPI NDSnmpExInformInfoTargets(
  const NDSnmpExTarget* pArray,
           AsnInteger  lenArray)
```

The first parameter is pointer to the array of:

```
struct NDSnmpExTarget
{
   char*      pszAddress;
   AsnInteger rqId;
   char*      pszSecurityName;
};
```

The length of the array is specified as the second parameter.

pszAddress is string representation of the target address (in most cases dotted decimal inet address, for example “192.168.1.1:162”). **rqId** is request id of the pdu sent to that address. And finally, **pszSecurityName** is pointer to community string (if SNMPv2) or user name (if SNMPv3).

If extension DLL wants to know just the request Ids of the PDUs sent (note that NDMPAgent changes rqId for each pdu that it sends out) it should set the last parameter in NDSnmpExNotification to **eNDSnmpExInformInfo_requestIds**. After NDMPAgent sends informs in this case, it will call NDSnmpExInformInfoRqIds method:

```
VOID WINAPI NDSnmpExInformInfoRqIds(
   const AsnInteger* pArray,
         AsnInteger  lenArray)
```

The first parameter is pointer to integer array of length specified as the second parameter. Array contains request ids of the inform pdus sent.

Now, extension DLL simply waits for the result of the inform requests. But where are the informs/traps sent?

`NDMPAgent` maintains tables used to select destinations for the notifications. If running in SNMPv3 mode, these tables (and how they are used) are described in RFC2573.

For SNMPv1/SNMPv2c mode `NDMPAgent` uses two tables. One table contains addresses of the targets, along with the parameters to be used when sending requests (timeout, retry count, community, version). The second one simply maps tags to the notification type. Here is the example:

```
[v1v2cTargetAddrTbl]
; name address         timeout retries tagList     community version
1=t1   192.2.1.51:162   5     0       tag1         public    SNMPv1(0)
2=t2   BG070:162        5     0       "tag1 tag2" private   SNMPv2c(1)
3=t3   ACCOUNTING1:4162 5     0       office       public    SNMPv2c(1)

[v1v2cNotifyTbl]
; notifyTag notifyType
1=tag1      trap(1)
2=tag2      inform(2)
3=router    inform(2)
```

Lets show in an example how `NDMPAgent` determines where to send notifications and how it chooses parameters for the messages. We'll assume that `NDMPAgent` is started in SNMPv2c mode.

Notification originator subsystem of the agent will perform the following steps. For each row in the Notify table it'll try to select targets from the Target address table. Selection happens when tagList in the Target address row contains tag from the Notify table.

For example, the 1st row in Notify table specifies tag "tag1". Rows 1 and 2 of the target address table contain this tag in the tagList. In other words, trap will be sent to addresses 192.2.1.51:162 and BG070:162.

The version of the SNMP message to send and which community string to use will be taken from the corresponding `community` and `version` columns.

The first destination, 192.2.1.51:162, is specified as SNMPv1 so conversion has to take place: v2Trap will be converted to v1trap pdu and SNMPv1 trap pdu will be sent to it.

The second destination, BG070:162, has SNMPv2c in the version column, so SNMPv2c message containing this v2trap PDU will be sent.

The second row in Notify table contains “tag2” tag that selects the second row in the Target address table. The type of the notification is inform. Since destination is SNMPv2c an inform PDU will be sent to it.

The last row in the Notify table will not select anything in the Target address table. There is no row in the Target address table that has “router” tag in the tagList.

As, you have seen, no notification is sent to ACCOUNTING1:4162 since the tagList for that address contains “office” tag which does not appear in any row of Notify table.

Now that the agent sent notifications, it’ll call **`NDSnmpExInformInfoRqIds`** and in this case array passed will have only one element, request id o f the only inform sent. We should save this request id so we can match response with inform request.

Here is the possible outcome of the sending informs.

The first (and probably the most wanted outcome) is that the target SNMP entity replied. In this case `NDMPAgent` calls **`NDSnmpExInformOnResponse`** passing:

1. rqId - requestId of the request pdu
2. pAddress – address where the request was sent to
3. ctxEngId - context engine id (NULL for SNMPv2)
4. ctxName - context name (NULL for SNMPv2)

If there is no response from the target `NDMPAgent` calls **`NDSnmpExInformOnTimeout`** passing:

1. rqId - requestId of the request pdu
2. pAddress – address where the request was sent to

If there is an error sending request (not enough memory, USM user not found, etc.) `NDMPAgent` calls **`NDSnmpExInformOnError`** passing:

1. rqId - requestId of the request pdu
2. errNum – error number

Finally if the target entity returns Report pdu (in SNMPv3 mode, report pdus are not used in SNMPv2c), `NDMPAgent` calls **`NDSnmpExInformOnReport`** passing:

1. rqId - requestId of the request pdu
2. pAddress – address where the request was sent to
3. pReportOid – oid of the report counter
4. cnt - reported counter value
5. ctxEngId - context engine id (NULL for SNMPv2)
6. ctxName - context name (NULL for SNMPv2)

For an example of the implementation of NuDesign's extension for sending notifications from the Extension Agent DLLs please take a look at **NotifyDll** Visual C++ project.

This DLL implements the first two object defined in `ND-GARAGE-MIB` mib. On every third request it raises Event notifying agent that it has notifications to send. In `NDSnmpExNotification` it simply populates the parameters (leaving varbind list empty). Note that the varbind list in v2Trap and inform pdus has the following pairs of object names and values:

```
sysUpTime.0
snmpTrapOid.0
additional varbinds . . .
```

`pVarBindList` in argument list for the **`NDSnmpExNotification`** function should contain only **"additional varbinds".** The first varbind (sysUpTime.0) will be supplied by the `NDMPAgent`. The second varbind (snmpTrapOid.0) requires only value, the one passed in pTrapOid parameter.

NotifyDll.DLL logs activity in the "NotifyDll.log" file. Here is the sample of how it may look after a few informs:

```
NDSnmpExNotification TRUE
NDSnmpExInformInfoTargets
      [0] 192.168.1.51:162,private,4
      [1] 209.146.195.200:162,private,5
NDSnmpExNotification  FALSE
NDSnmpExInformOnResponse  address=192.168.1.51:162, rqId=4
NDSnmpExNotification TRUE
NDSnmpExInformInfoTargets
      [0] 192.168.1.51:162,private,13
      [1] 209.146.195.200:162,private,14
NDSnmpExNotification  FALSE
NDSnmpExInformOnResponse  address=192.168.1.51:162, rqId=13
NDSnmpExNotification TRUE
NDSnmpExInformInfoTargets
      [0] 192.168.1.51:162,private,16
```

```
       [1] 209.146.195.200:162,private,17
NDSnmpExNotification  FALSE
NDSnmpExInformOnResponse  address=192.168.1.51:162, rqId=16
NDSnmpExInformOnTimeout  address=209.146.195.200:162, rqId=5
NDSnmpExNotification TRUE
NDSnmpExInformInfoTargets
      [0] 192.168.1.51:162,private,19
      [1] 209.146.195.200:162,private,20
NDSnmpExNotification  FALSE
NDSnmpExInformOnResponse  address=192.168.1.51:162, rqId=19
NDSnmpExNotification TRUE
NDSnmpExInformInfoTargets
      [0] 192.168.1.51:162,private,22
      [1] 209.146.195.200:162,private,23
NDSnmpExNotification  FALSE
NDSnmpExInformOnResponse  address=192.168.1.51:162, rqId=22
NDSnmpExInformOnTimeout  address=209.146.195.200:162, rqId=14
NDSnmpExInformOnTimeout  address=209.146.195.200:162, rqId=17
NDSnmpExInformOnTimeout  address=209.146.195.200:162, rqId=20
NDSnmpExInformOnTimeout  address=209.146.195.200:162, rqId=23
```

2. About NuDesign SNMPv3 Agent Service for Windows

NuDesign SNMPv3 Agent Service for Windows is a seamless upgrade of Microsoft SNMP Agent Service. The product comes with a redistributable SNMPv3 Agent Configuration Applet. All extension DLLs that run with Microsoft SNMP Agent Service will run with NuDesign's SNMPv3 Agent Service for Windows as well.

NuDesign SNMPv3 Agent Service operates under Windows XP, Windows Server 2003, 2008 & 2012, Vista, Windows 7, Windows 8/8.1 and Windows 10; 32bit and 64 bit versions. Benefits of using NuDesign SNMPv3 Agent Service for Windows:

- it supports SNMP v1/v2c/v3 access to MIB management objects via Microsoft's Extension Agent DLL architecture. A full SNMPv3 support is offered, v3 security (CBC-DES or CFB128-AES-128 privacy and HMAC-MD5 or HMAC-SHA authentication) and v3 Administration Framework for Notifications (implements SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB)
- support for either the 32bit legacy extension DLL's in both 32/64bit Windows environments or new 64bit extension DLL's in 64bit Windows environments.
- it provides full support for "SNMP over IPv6" as well as "SNMP over IPv4" networks.
- it allows extension DLLs to be started or stopped dynamically, without interrupting the Service, with the help of NuDesign SNMPv3 Service Configuration Applet. Microsoft SNMP Service needs to be stopped before a new extension DLL is added or an existing DLL is removed. There is no such restriction with NuDesign's SNMP Service.
- it supports the use of remote subagents, either the lightweight subagents based on UDP communications & SNMP BER encoding, or the TCP/IP based subagents using the IETF AgentX, rfc2741 standard. The NuDesign Service can be deployed as a Master controling a network of desktop & embedded devices equipped with AgentX remote subagents.
- it provides full support for all SNMP notification types (v1/v2 traps & informs including a callback mechanism that provides extension agent DLL with status information).
- it fully implements SNMP Proxy Forwarder Application as per rfc3413.
- it is a multithreading service. One thread is always listening to incoming requests and other threads are responsible for satisfying the requests. With this architecture, SNMPv3 Agent Service is never blocked.
- it has been built using Standard C/C++ libraries, can therefore be readily ported to numerous embedded environments.

This product in conjunction with NuDesign's Visual xAgentBuilder for C++ provides a comprehensive SNMPv3 Agent solution for Windows OS'es.

3. About NuDesign Technologies

NuDesign provides network monitoring and management solutions to networking manufacturers and service providers worldwide. The company also provides professional services to customers developing custom solutions.

The products and services that NuDesign's customers develop enhance the visibility and control of networking infrastructures, applications, services and facilitate configuring, managing, and controlling of products in the LAN, broadband access, and telecom industries.

NuDesign's products / services include:

- the Management Agents for embedded targets, implementing standard and custom YANG / MIB data models with secure, integrated NETConf / SNMPv3 / CLI and / or RESTConf access.
- the secure SNMPv3 and CLI management solutions for multiple desktop and embedded targets.
- the secure monitoring of Windows and Linux Servers using extensible SNMPv3 Agent Service for Windows or Linux with standard and / or custom extension subagents, configured with the provided SNMPv3 Configuration Editor. NuDesign SNMPv3 Service for Windows is a direct, drop-in replacement of Microsoft Windows SNMP Service.
- the SNMP Browser, MIB Builder and Traffic Monitor products that are feature rich, extensible, and easy to use in development, test and production environments.
- specialized SNMPv3 Management Agents and Managers for avionics AFDX® / ARINC664 networks and SNMPv1 products for NTCIP Transportation industry.

The highly automated management Agent / Subagent / Manager development tools, code generation wizards, multiple target development libraries, samples, evals with associated tutorials and our technical support enable quick learning, fast prototyping, experimentation, development and reliable production deployment.

The code generation tools facilitate organization of design process, providing for generation of very complete and immediately compilable agent or manager user code and project files for target OS / RTOS.

For more information please visit **www.ndt-inc.com** or call 416 737 0328