NuDesign MultiProtocol (SNMP v1/v2c/V3) Service (15.x) for Desktop/Server Linux Installation and Usage Notes

Table of Contents

Document History

Date	Rev.	Remarks
2015-01-16	A000	- Original release.
2015-02-24	A001	- Update dynamic configurations - Add general XNV documentation
2015-12-04	A002	- expand the compatibility, requirements and installation sections. Add updates for the evaluation version.
2015-12-10	A003	- update for new install layout with libs being installed under the install directory root. - other minor edits.
2016-02-11	A004	- add documentation for V1 trap “up convert” to an inform option.
2016-12-16	A005	- updates for v12.x - updates to dynamic configuration section,, particularly relating to new auth. and priv. protocols.
2017-11-15	A006	- minor updates
2018-05-18	A007	- add FIPS mode documentation.
2018-07-20	A008	- add “Bind Addresses” section.

Introduction

This document describes installation and usage of this NuDesign Technolgies Inc. SNMPv3 master agent/service/daemon for desktop and server Linux.

The document discusses these main components of this release:

- Compatibility
- Requirements
- Installation
- Executables

Compatibility

The product has been extensively tested using a variety of desktop and server Linux distributions, including various versions of Fedora, RHEL 5, 6, & 7, CentOS 6.x & 7.x and recent versions of openSUSE(42.1), and Ubuntu(15.10), inboth 32 & 64 bits when both are supported by the distribution.

Additionally, the service is compatible with the following RFCs

- RFC 1157, Simple Network Management Protocol
- RFC 1213, Management Information Base for Network Management of TCP/IP-based internets: MIB-II
- RFC 2741, Agent Extensibility (AgentX) Protocol Version 1
- RFC 2790, Host Resources MIB
- RFC 2863, The Interfaces Group MIB (IF)
- RFC 3410, Introduction and Applicability Statements for Internet Standard Management Framework
- RFC 3411, An Architecture for Describing SNMP Management Frameworks
- RFC 3412, Message Processing and Dispatching
- RFC 3413, SNMP Applications
- RFC 3414, User-based Security Model
- RFC 3415, View-based Access Control Model
- RFC 3416, Version 2 of SNMP Protocol Operations
- RFC 3417, Transport Mappings
- RFC 3584, Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework
- RFC 3826, The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model
- RFC 4113, Management Information Base for the User Datagram Protocol (UDP)
- RFC 4292, IP Forwarding Table MIB (IP-FORWARD)
- RFC 4293, Management Information Base for the Internet Protocol (IP)
- RFC 7860 HMAC-SHA-2 Authentication Protocols in User-Based Security Model (USM) for SNMPv3

Additionally, the product adopts several industry conventions based on “internet drafts”, to provide enhanced levels of privacy. These are:

- <draft-reeder-snmpv3-usm-3desede-00> Extension to the User-Based Security Model (USM) to Support Triple-DES EDE in "Outside" CBC Mode.

- <draft-blumenthal-aes-usm-04> The Aes Cipher Algorithm in the SNMP's User-based Security Model. This adds CFB128-AES-192 and CFB128-AES-256. These are in addition CFB128-AES-128 provided for in RFC 3826.

Requirements

Several specific files, generally part of or available to a distribution are required. The following is a specific list of requirements to execute the service.

- OpenSSL (libcrypto.so) that includes AES[1] and optionally libssl.so[2], if your license includes web server and https is desired.
- Std c++ library 6.x (libstdc++.so.6)
- libdl.so.2
- libpthread.so.0
- libm.so.6
- libgcc_s.so.1
- several IP ports need to be allowed through the system firewall, particularly **UDP** 161, but possibly others, depending on licensing, configuration and the options that are enabled.

[1] The SNMP agent expects to find the **OpenSsl** crypto library named as '**`libcrypto.so`**' on the library path. If your system provides the library with another name you might consider creating a symbolic link from it to '**`libcrypto.so`**' as a solution to satisfy the requirement. This should be a viable solution on most systems.

E.g. '`ln -sf libcrypto.so.0.9.8 libcrypto.so`'

[2] Similarly, the library '**`libssl.so`**' is expected to be available if **`https`** is desired for the daemon's (optional) embedded web server. The product has been tested with various versions of OpenSsl, including '**`0.9.8`**' and '**`1.0`**'

License File

This SNMP service **requires** a ***license file to run***. The license file is provided at the time of the purchase of the product. The license file is a text file with the name root **"NuDesignSNMPServiceLicenseKey"**, plus decimal digits conveying the number of licenses associated with the key and a licensee number, plus the extension **".txt"**.

E.g.

NuDesignSNMPServiceLicenseKey500-00.txt

The file must be copied into one of two places. The first is the **/etc** directory, the second is the directory in which the executable for the service resides. Since this is typically `/usr/bin`, the former directory is recommended.

> **Note**: the evaluation version does not require a license file, but will cease to operate after 30 days.

Installation

This release comes in the form of a Linux executable named with a "`.bin`" file extension. There is a 32 bit and 64 bit version of the installer, though both include both 32 & 64 bit executable files. The installation determines whether the distribution is a 32 bit or a 64 bit system at run time and will install the appropriate files. The installation executables themselves only differ by the installer's executable bitness.

> **Note**: If you try to use the 32 bit installer on a 64 bit system, it will only succeed if your distribution supports both 32 bit and 64 bit executables and the system being installed on current has the 32 bit support installed.

By default files that are adjuncts to the execution of the service are installed under **`/usr/local/NuDesign`**. You may choose to install these elsewhere when prompted to provide an installation directory.

The service executables are installed under the '`/usr/bin`' directory. The associated shared object libraries are installed in the '`lib`' directory under the root of the installation directory. As part of the install, this '`lib`' directory is added to the library load path.

> **Note**: If, when starting the service, you receive an error indication that these libraries were not found, then the operation of adding, the libraries to the library load path may not have been successful. To correct this, you can:
>
> - add the '`lib`' directory to the library load path yourself (see **`man ldconfig`**)
> - copy (cp) or link (ln) the files installed under the install '`lib`' directory to either the /usr/lib or /usr/lib64 directory, depending on the requirements of your distribution.
> - Ensure **LD_LIBRARY_PATH** is exported into the environment of the service prior to the service's execution.
>
> **E.g.**
>
> ```
> export LD_LIBRARY_PATH=/usr/local/NuDesign/libs
> ```

Also note that the installation software needs to be run at an elevated privilege level (**`root`**) to correctly install all files.

Executables

The two executables included with the product are described in this section.

NDAgD[3]

Synopsis: **NDAgD** [options]

Description:

This is the NuDesign Multi Protocol SNMP/FCGI/CLI (master) agent service. The master agent is IPv4/IPv6 compatible. If IPv6 is determined to be running on the host, then the SNMP agent is set to use IPv4 & IPv6 concurrently.

In addition to SNMP, the executable includes[4] a fastCGI client for presenting SNMP data via an external web server.

The executable resides in the `/usr/bin`[5] directory.

Note: Since the default UDP service port in usually considered a privileged port, you typically need to execute this service with elevated privilege. You may also need to alter the firewall configuration to gain access to the port from a non local location.

NDAgD has the following dependencies on the following NDT specific shared object support libraries:

- `libNDWebs.so.0`[6]
- `libNDWebsLinux.so.0`[7]
- `libNDSubAg`[8]`.so.0`
- `libNDAgentX`[9]`.so.0`
- `libNDSProxy.so.0`

[3] The master service's name is **NDAgDHttp** with the IPv4 HTTP/HTTPS option or **NDAgDV2** or **NDAgDV3** for versions without HTTP or fastCGI. The eval's name is **NDAgDE**. It is referred to as **NDAgD** in any documentation.
[4] Only on versions including fcgi Web support.
[5] This the typical the path for the 32 bit version. For the 64 bit version, typically the path to the executables is `/usr/bin64`.
[6] Included only with the IPv4 HTTP option
[7] "
[8] Optional (use) NDT SubAgent management library
[9] Optional (use) AgentX conforming management library

- libNDMibHX.so.0
- libNDVacm.so.0
- libNDNVol.so.0
- libNDSAgent.so.0
- libNDSUdp.so.0
- libNDSocket.so.0
- libNDSocketLinux.so.0
- libNDSnmp.so.0
- libNDSCryp.so.0
- libNDSsi.so.0[10]
- libNDMibH.so.0[11]
- libNDSys.so.0
- libNDSysLinux.so.0
- libNDFCGI.so.0[12]
- libfcgi.so.0[13]

Additionally, the following system shared object library is required. These libraries should be available with the Linux distribution. If not, it is available in source for download from the internet.

- libcrypto.so
- libssl.so (optional)

Lastly the sub agent libraries for MIB-2 and Host Resources are available to load with the master agent.

- libNDMIB2.so.1[14]
- libNDMIB2-6.so.0[15]
- libNDHost.so.1

can be placed here as well, depending on the configuration of `ndagdext.xnv` (see below).

[10] only provided if your license includes either the HTTP options or the HTTP/fastCGI option.
[11] **`libNDMibHE.so.0`** on the evaluation version.
[12] libNDFCGI.so.0 is the NuDesign fastCGI interface and only exists in the product when your license includes the HTTP/fastCGI interface.
[13] libfcgi.so.0 is the fastCGI library required by libNDFCGI.so.0 and is only required when your license includes the HTTP/fastCGI interface.
[14] libNDMIB2.so.0 is an implementation based on RFC1213.
[15] libNDMIB2-6.so.0 is an implementation based on newer RFCs and also includes Ipv6 specific information. It is recommended the master agent load only this one or libNDMIB2.so at once.

NDAgD Command Line Options:

When executed without command line options, it displays a short help and exits. The following command line options are recognized.

-c, run the agent in the console. (default option)

-d, run the agent as a background process/service (daemon).

-p, available only when used in conjunction with the '–d', enables a communications pipe to the agent console that be accessed using '`NDAgClient`'.

-s, stops the agent running in the background.

-l <name>, "loads" the named sub agent (as given in `ndagdext.xnv`) with the agent running in the background.

-u <name>, "unloads" the named sub agent (as given in `ndagdext.xnv`) from the agent running in the background.

-f <name>, freshens ("unloads" and "loads") the named sub agent (as given in `ndagdext.xnv`) from the agent running in the background.

-r, causes the agent running in the background to re evaluate the **`ndagdext.xnv`** file.

Using fastCGI with Web Servers Supporting fastCGI[16]

FastCGI is a third party technology that is freely available (see the site link below for the details of licensing). It is intended to be an efficient CGI like mechanism for extending web services of a web server. The technology is stable.

This section briefly documents how to configure **NDAgD** to enable fastCGI with several common web servers. Primary documentation for this functionality will be found with the particular web server being documented. Supplemental information may also be found at the fastCGI web site at http://www.fastcgi.com.

The examples provided below enable using your existing web application, if you have used previous versions of the product and also work with the web pages generated by xAgentBuilder for your project. It is important to note that all processing in the generated web application that require processing has a URL prefixed with "`/ndmib?filename=`"[17].

Additionally, fastCGI can be set up a number of different ways, but all examples below assume that the selected web server and the master agent process are resident on the same computer and communicate via a TCP port at 9000. Access to the same file system is assumed. It is conceivable to configure such that the web server is on a different host. In that case, care is required to ensure that the necessary files for you web application are available in the file space of the master agent as well.

For all web servers supporting fastCGI, the first step is to ensure that any fastCGI module(s) are loaded and functional. The below examples are excerpts associated with the configuration files for each.

Apache

In the 'Main' server configuration section add the following line...

```
FastCGIExternalServer   /var/www/html/ndmib -host 127.0.0.1:9000
```

The above assumes

```
DocumentRoot "/var/www/html"
```

[16] only applicable if your license includes the HTTP/fastCGI option.

[17] This fact is used in the configuration examples. It is usually possible to configure fastCGI to be based on a particular file extension. While this mechanism is possible from the standpoint of processing the data in the request, it will break compatibility with the generated pages from xAgentBuilder and any web applications based on previous versions of this product.

NGINX

Within the `server` definition block, create a `location` definition block as follows...

```
location ~ /ndmib*$ {
      root              html;
      fastcgi_pass      127.0.0.1:9000;
      include           fastcgi_params;
}
```

Below is a possible configuration for causing processing based on the file extension '`.ndfcgi`'. See the caveat in the previous section.

```
Location ~ \.ndfcgi$ {
      root              html;
      fastcgi_pass      127.0.0.1:9000;
      include           fastcgi_params;
}
```

Note that this would pass files with this extension to the same process as the example above. This is permissible.

Lighttpd

You should have a section in your configuration file pertaining to fastCGI something like the following…

```
include "conf.d/fastcgi.conf"
fastcgi.server = (
                  ".php" => ((
                     "bin-path" => "/usr/bin/php-cgi",
                     "socket" => "/tmp/php.socket"
                  ))
)
```

To this, add the section below…

```
                  "/ndmib" => ((
                       "host" => "127.0.0.1",
                       "port" => 9000,
                       "check-local" => "disable"
                   ))
```

The entire section would then look as follows….

```
include "conf.d/fastcgi.conf"
fastcgi.server = (
                  ".php" => ((
                     "bin-path" => "/usr/bin/php-cgi",
```

```
                    "socket" => "/tmp/php.socket"
                )),
                "/ndmib" => ((
                    "host" => "127.0.0.1",
                    "port" => 9000,
                    "check-local" => "disable"
                ))
    )
```

This causes `lighttpd` to pass all URLs starting with "`/ndmib`" to 127.0.0.1:9000 for processing. The last line suppresses `lighttpd` from checking that the URL exists before handing a request off to the fastCGI processor in the mast agent.

Below is a possible configuration for causing processing based on the file extension '`.ndfcgi`'. See the caveat in the previous section.

```
                ".ndfcgi" => ((
                    "host" => "127.0.0.1",
                    "port" => 9000,
                    "check-local" => "disable"
                ))
```

Note that this would pass files with this extension to the same process as the example above. This is permissible.

Using fastCGI with Web Servers Supporting Only CGI[18]

Some web servers that do not support fastCGI, but support CGI can be used with the application 'cgi-fastcgi' (included a convenience with the product, but is also available to build in the fastCGI SDK available from the fastCGI site). While this will enable processing of data requests in the same fashion as with direct fastCGI, however there are a couple implications with using this method.

The first issue is that this essentially makes the request a CGI request with the same drawback one would expect with CGI, namely performance. This is because a new process is started with each request, so this tends to be a time and system resource intensive operation.

The second issue is that the web pages generated by embedded xAgentBuilder, nor any existing web application based on the same, will not function directly. This is due to the method of invocation. It is important to note that all processing in the generated web application that reliant on processing the URL prefix

[18] only applicable if your license includes either the HTTP options or the HTTP/fastCGI option.

"/ndmib?filename=", as is that expected with pages generated by embedded xAgentBuilder, will not work correctly.

As with the examples above for web servers directly supporting fastCGI, the example below assumes that the selected web server and the master agent process are resident on the same computer and communicate via the TCP port 9000.

Assuming the platform specific executable 'cfg-fcgi' is in the web accessible directory 'cgi-bin', then a typical invocation line would look like the following…

```
http://127.0.0.1/cgi-bin/cgi-fcgi?-bind+-connect :9000+Index.htm
```

Where:

- '?' Is the separator between the executable name and the parameter list.

- '+' Is the parameter separator.

- '-bind' and '-connect' are parameters to cfg-fastcgi. Consult fastcgi documentation for more information. Brielfy though, " :9000" specifies communicating on TCP 9000 on the local computer. This is also equivalent to " 127.0.0.1:9000".

- 'Index.htm' is a file to process.

Enabling SSL / HTTPS

If your license includes the embedded GoAhead IPv4 web server, then there are a few requirements before the web server will enable secure web services.

First, the system specific OpenSSL library **`libssl.so`** must be on the library load path.

Secondly, the following three certificate files must exist and be in the directory where the daemon is executing from:

- `server.pem`
- `certs/cacert.pem`
- `certs/cakey.pem`

As a convenience, we provide these files as “self signed” certificates. You **SHOULD NOT** release your product with these certificates. They are provided **FOR TESTING ONLY**.

> **Note**: When using the self signed certificates, most web browsers will warn you about the certificate and you must accept the certificate to continue to use HTTPS.

There are a variety of resources on the Internet as to how to “self sign” and certificates in general. The following link http://www.openssl.org/docs/HOWTO/certificates.txt and http://www.openssl.org/docs/HOWTO/keys.txt are OpenSSL documents relating to certificates and keys.

Lastly, the web server must be configured to enable the secure port. This is the default configuration. (See **`SecureMode`**, documented below)

Enabling Web Support for Remote Sub Agents

If your license includes a web server and if any of the sub agents that are going to be managed by the master agent are remote and derived from a xAgentBuilder project, then an additional step is required to enable web based access to these agents. An auxiliary object definitions file must be located in the master agent's executable directory. xAgentBuilder creates the necessary file for each sub agent project when the project is generated. This will be a file with the file extension **`.xmi`**. The file is placed in the xAgentBuilder project `/web/<project name>` directory and will have the name, under the `/<project name>.xmi`. Just copy this file to the master agent's executable directory and restart the master agent.

If you have multiple remote sub agents that implement multiple MIBs, then it is permissible to consolidate the files if you wish. The file name isn't important as long as the file extension remains **`.xmi`**.

Due to a slightly different registration process for AgentX remote sub agents, a modification step is required to the corresponding XMI files.

For every file XMI file, the user must add the following section:

[Module]
Name=ND-GARAGE-MIB
Subtree=1.3.6.1.4.1.4761.99.11.1

Where:

`[Module]`	identifies the section associated with the follow info.
`Name=`	is the MIB module name that is associated with the information in the XMI file.
Subtree=	A dot notational OID of the registration point for the remote sub agent. Generally this is the "root" node of the sub agent and usually has the same value as the shortest OID in the XMI file.

A XMI modified for AgentX is still compatible with the format of XMI files required by the NDT Sub Agent handler.

> Note: when supporting AgentX remote agents, you may not consolidate XMI files into a single file.

Configuration:

NDAgD[19] utilizes two files for nonvolatile storage. These are **ndagd.xnv** and **ndagdext.xnv**. The former controls global configuration items and the latter is used to manage loadable sub agents at start up.

Interpreting XNV Configuration files

XNV files are simple text files that adhere to a couple simple rules, described as follows.

; header comment
; header comment
; . . .

[<sectionName1>]
; comment for section
<itemKey11> = <itemValue11>
<itemKey12> = <itemValue12>
. . .

[<sectionName2>]
; comment for section
<itemKey21> = <itemValue21>
<itemKey22> = <itemValue22>
. . .

where

; = comment line, ignored
<sectionName> = string
<itemKey> = string
<itemValue> = <primitiveValue> | <compositeValue>
<compositeValue> = <primitiveValue> [<primitiveValue> [. . .]]
<primitiveValue> = string | integer | string“(“integer“)”

Note that you can have as many comment lines in the file header as you wish. Section comments are restricted to one line only. While you can specify multiline comments in the section, only the first one will be saved.

[19] The master agent name is **NDAgDHttp** with the HTTP option or **NDAgDV2** or **NDAgDV3** for versi ons without HTTP or fastCGI.

ndagd.xnv Options

(Installed under the `'/etc'` directory). Note, some of these sections do not apply to all versions of the master agent. The one that do, depend on licensing.

The file, `ndagd.xnv` has a wide range of configurable options. The configuration XNV for the master agent can be be broken into two broad categories. These are options

- that control individual master agent/server modules/components. The examples were taken from a sample XNV file.
- for configuring the SNMP v3 engine (or v1/v2 engine if the engine has been configured to be constrained to operate in SNMPv1/v2c only)

Module Options

[SNMP Agent]

```
;Version=SNMPv1(0)
Version=SNMPv3(3)
Port=161
NumWorkerThreads=3
TransportBlockInterval=100000
NotificationReceiverPort=0
```

Where:

`[Snmp Agent]` identifies the section associated with the SNMP component.

`Port=` is the UDP port number on which the SNMP agent responds to requests.

`Version=` is the maximum SNMP version number the SNMP agent will respond to. Default is `SNMPv3(3). SNMPv1(1)` or `SNMPv2c(2)` may also be used.

`NumWorkerThreads=` The number of threads available to process requests.

`TransportBlockingInterval=` Value in microseconds that the request processing thread blocks to wait requests.

NotificationReceiverPort= The UDP port on which the master agent listens for SNMP notifications. If the value is zero (0), no port is opened.

[SNMP Agent/Bind Addresses]

```
Default=4
0=192.168.0.1:161
1=192.168.0.2:161
2=127.0.0.1:161
4=[::1]:161
```

An optional section for specifying specific interfaces to bind to[20].

Default= Specifies the number of interface expressions to process.

n= Specifies a particular interface expression, where 0<=n<Default. && n<=32. Each expression specifies an IPv4 or IPv6 address and port number.

[20] You may use this facility to restrict the interfaces the service is visible on. It also could be used to overcome the phenomenon that occurs when there are multiple interfaces on a single segment and you wish to ensure responses come from the addressed interface.

[FCGI Agent]

This section controls options that pertain to the FCGI web interface.

```
E.g.Socket=:9000
Root=/usr/local/NuDesign/web
Enable=1
```

Where:

[FCGI Agent][21] identifies the section associated with the fastCFGI server component. (Optional component)

Socket= is the specification of the socket on which the fastCGI server responds to fastCFG requests. If the string is colon (":") prefixed, the it specified a TCP port number.

E.g.

Socket= :9000

Otherwise, the it specifies a unix domain socket.

E.g.

Socket= /tmp/fastcgi-nd

Root= is the path to the root directory for the server's HTML pages.

Enable= controls whether the fastCGI server is enabled on start up or not. This is optional and if it does not exist, the default is enabled. If it has a value of '0' then the fastCGI server is not enabled.

[FCGI VACM]

This section control options that pertain to how the FCGI Agent accesses SNMP variables in the SNMP engine.

E.g.

```
SecurityName=private
SecurityModel=secSNMPv2c(2)
SecurityLevel=noAuthNoPriv(1)
ContextName=""
```

[21] Only necessary when your license includes the FCGI web support server.

Where:

[FCGI VACM][22] identifies the VACM section associated with the fastCGI server. This specifies the MIB "view" available to object presented via the fastCGI server. Objects requested in pages that are "in view" will return results. Those outside will not. Changing this configuration should generally not be necessary unless the default configuration provided is invalidated by administrative changes to the V3 configuration of the agent.(Optional component)

SecurityName= is the V3 security name the web server uses when accessing MIB objects. This may be any `usmUserSecurityName` currently in the `usmUserTable`. By default `'private'` is specified. An entry in the `usmUserTable` for this name is also default.

SecurityModel= is the V3 security model the fastCGI server uses when accessing MIB objects. This may be one of `secSNMPv1(1), secSNMPv2c(2) or secSNMPv3(3).`

SecurityLevel= is the V3 security level the fastCGI server uses when accessing MIB objects. Default is `noAuthNoPriv(1).` It could also take the values `authNoPriv(2)` or `authPriv(3)`

ContextName= is the V3 context name the fastCGI server uses when accessing MIB objects. Default is "".

[22] Only necessary when your license includes the FCGI web support server.

[Web Agent]

This section controls options that pertain to the embedded Web Agent.

```
E.g.
            Port=8080
            Root=/usr/local/NuDesign/web
            HomePage=Home.htm
            EnableLocalSecurity=1
            SecurePort=443
            SecureMode=3
```

Where:

[Web Agent][23] identifies the section associated with the embedded IPv4 HTTP/HTTPS server.(Optional component)

Port= is the TCP port number on which the embedded HTTP server responds to requests.

Root= is the path to the root directory for the server's HTML pages.

HomePage= identifies the specific default HTML page.

SecurePort= specifies the port secure web services (https) are provided on. If not provided, the default is 443.

Secure Mode= specifies the web service ports to open. When the value is:

1: implies open only the web service port controlled by "Port=".
2: implies open only the secure web service port controlled by "SecurePort=".
3: implies open both ports.

If not provided, the default is '3'.

EnableLocalSecurity= when set to '1' enables a user name and password challenge to be issued on the local host if user management is enabled. Default is no entry, meaning local challenges are not issued.

Enable= controls whether the web server is enabled on start up or not. This is optional and if it does not exist, the default is

[23] Only necessary when your license includes the embedded IPv4 HTTP server.

enabled. If it has a value of zero (0) then the fastCGI server is not enabled.

[Web VACM]

This section controls options that pertain to how the Web Agent accesses SNMP variables in the SNMP engine.

E.g.

```
SecurityName=private
SecurityModel=secSNMPv2c(2)
SecurityLevel=noAuthNoPriv(1)
ContextName=""
```

Where:

`[Web VACM]`[24] identifies the VACM section associated with the optional embedded HTTP server. This specifies the MIB "view" available to web server. Objects requested in pages that are "in view" will return results. Those outside will not. Changing this configuration should generally not be necessary unless the default configuration provided is invalidated by administrative changes to the V3 configuration of the agent.(Optional component)

`SecurityName=` is the V3 security name the web server uses when accessing MIB objects. This may be any `usmUserSecurityName` currently in the `usmUserTable`. By default `'private'` is specified. An entry in the `usmUserTable` for this name is also default.

`SecurityModel=` is the V3 security model the web server uses when accessing MIB objects. This may be one of `secSNMPv1(1), secSNMPv2c(2) or secSNMPv3(3).`

`SecurityLevel=` is the V3 security level the web server uses when accessing MIB objects. Default is `noAuthNoPriv(1).` It could also take the values `authNoPriv(2)` or `authPriv(3)`

`ContextName=` is the V3 context name the web server uses when accessing MIB objects. Default is "".

[24] Only necessary when your license includes the embedded IPv4 HTTP server.

[CLI VACM]

This section controls options that pertain to how the CLI accesses SNMP variables in the SNMP engine.

E.g.

```
SecurityName=private
SecurityModel=secSNMPv2c(2)
SecurityLevel=noAuthNoPriv(1)
ContextName=""
```

Where:

[CLI VACM] identifies the VACM section associated with the CLI interface. This specifies the MIB "view" available to web server. Objects requested in pages that are "in view" will return results. Those outside will not. Changing this configuration should generally not be necessary unless the default configuration provided is invalided by administrative changes to the V3 configuration of the agent.

(the following items apply to both sections)

SecurityName= is the V3 security name the web server uses when accessing MIB objects. This may be any `usmUserSecurityName` currently in the `usmUserTable`. By default `'private'` is specified. An entry in the `usmUserTable` for this name is also default.

SecurityModel= is the V3 security model the web/cli server uses when accessing MIB objects. This may be one of `secSNMPv1(1), secSNMPv2c(2) or secSNMPv3(3).`

SecurityLevel= is the V3 security level the web/cli server uses when accessing MIB objects. Default is `noAuthNoPriv(1).` It could also take the values `authNoPriv(2)` or `authPriv(3)`

ContextName= is the V3 context name the web/cli server uses when accessing MIB objects. Default is "".

[XCLI]

This section controls options for the xCLI command line.

```
E.g.
            ExePathAndName=cli
            ConfigPath=/ndxcli
            Timeout=2
```

Where:

[XCLI][25] identifies the section associated with the XCLI command line component.(Optional component)

ExePathAndName= is the path and filename of the xcli command line console that may be invoked from a SSI web sequence.

ConfigPath= is the path to the configuration repository for the xcli command line console.

Timeout= is the time in seconds that a xcli command line console is allowed to run to process a CLI command sequence.

[AXMaster]

This section controls options that pertain to the XAgent master.

```
            IP=4
            Port=705
```

Where:

[AXMaster] identifies the section associated with the AgentX configuration component.

Port= is the TCP port number on which the AgentX handler responds to requests. If this specification doesn't exist or is set to zero (0), then the library libNDAgentX.so is not loaded at start up.

IP= (optional) specification for IP version handling. Values may be 4 or 6, with 6 being the default when not specified. A

[25] Only necessary when your including the an xCLI command line processor.

value of 6 enables the communication socket for IPv4 and IPv6.

[SAMaster]

This section controls options that pertain to the NDT master agent.

```
E.g.
            Port=10001
            IP=4
```

Where:

[SAMaster]	identifies the section associated with the optional NDT Sub Agent configuration component.
Port=	is the UDP port number on which the NDT Sub Agent handler responds to requests. If this specification doesn't exist or is set to zero (0), then the library libNDSubAg.so is not loaded at start up.
IP=	(optional) specification for IP version handling. Values may be 4 or 6, with 6 being the default when not specified. A value of 6 enables the communication socket for IPv4 and IPv6.

[General]

Where:

```
            CPU=0
```

[General]	identifies the non-specific section "General"
CPU=	(optional) on platforms with multiple processors (sockets or cores), specifies the CPU affinity for the application. When no CPU statement is specified, CPU affinity is not set. If CPU has a value of –1 then the affinity is set to the last processor found. Acceptable settings are from 0 to n-1, where 'n' is the total number of processors available to the OS on the system. E.g. `CPU=1` would associate the application with CPU1. You may also turn off affinity assignment by specifying `CPU=-2`.

ForceMIPSMode= This option alters the behavior of the service, when running in **Version=SNMPv3(3)** (see the **SNMP Agent** section above), so that when enabled (set to '1'), restricts the authentication and protocols used by the service, to be consistent with those compliant with FIPS 140-2. See the "**FIPS Mode Operation"** section for more information.

[ProxyForwarder]

```
Enable=1
```

Enable= Controls whether the proxy forwarder is enabled or not. By default it is enabled. When set to zero(0), it disables it. See **v3ProxyTable** documentation.

SNMP Engine Options

SNMP MIB Options

[mib-2.system]

Where:

SysDescr= A textual description of the node. This is either a user configured value or computer generated. See **CreateSysDescr**.

SysObjectID A configurable authoritative identification of the master agent. Default: `1.3.6.1.4.1.4761`

sysLocation A configurable textual description for the location of the node.

sysContact A configurable textual description of the contact person for the node.

SysName The assigned name of the node. This is either a user configured value or the host name of the node. See **SetSysNameFromHostName**.

SysServices An integer value relating to the primary services associated with this node. See RFC3418 for more information about this setting.

CreateSysDescr When set to '1', the master agent will generate a string for SysDescr describing hardware and software.

E.g.

```
"Hardware: Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz Processors 12 -
Software: Linux 3.18.7-100.fc20.x86_64 #1 SMP Wed Feb 11 19:01:50
UTC 2015"
```

The value is not saved so the same value of `SysDescr` is retained at the next restart.

SetSysNameFromHostName When set to '1' causes the master agent to resolve the host name of the node. The value is not saved

so the determined value of **`sysName`** is retained for the next restart.

[mib-2.snmp]

Where:

`SnmpEnableAuthenTraps` When set to '1' causes authentication failure notifications to be sent. This setting directly affects the value of **`snmpEnableAuthenTraps`** at start up.

NotificationOriginator Options

Where:

`UpConvertV1ToInform` When set to '1' causes V1 traps to be up converted to an inform. The default is not to up convert.

SNMP v3 Options

The content described in this section affects the operation of the SNMP engine when the SNMP agent is set to SNMPv3 (`SNMPv3(3)`) mode. See the '`Version=`' expression in the `[SNMP Agent]` section.

[v3SnmpEngine]

```
EngineBoots=3055
EngineID=00:00:12:99:7f:00:00:01:ab:cd:ef:bb
TimeSyncMode=0
```

Where:

`EngineBoots` The number of times that the SNMP engine has (re-)initialized itself since snmpEngineID was last configured.

`EngineID` An SNMP engine's administratively-unique identifier.

`TimeSyncMode` When zero (0) (default) or when it doesn't exist in the configuration, the internal engine ID table is indexed by engineID and network address pair. This mode allows for supporting multiple engines with the same engineID. When set to any other value, the table is maintained by engineID only, requiring network administration to ensure unique engined IDs.

`SendAuthFailWithVBL` When set to one(1) will enable appending a varbind list of two additional.variable to aid in diagnosing the cause of the authentication failure. Note: this variable affects operation in v1 or v2c modes as well.

[v3SnmpTargetAddrTable]

This is the non volatile representation of the **`snmpTargetAddrTable`** and **`snmpTargetAddrExtTable`**. Each row is organized as follows:

```
rowNumber=taName taDomain taAddress taTimeout taRetryCount taTagList taParams
              taStorageType taRowStatus taExtTMask taExtMMS
```

Eg.

```
1=t1 udp(1) 192.168.0.1:162 15 0 tag1 p1 nonVolatile(3) active(1) "" 484
```

Note, the final two entries in the row correspond to entries from the **`snmpTargetAddrExtTable`**. See RFC 3413 for more information about the contents of this table.

[v3SnmpTargetParamsTable]

This is the non volatile representation of the **snmpTargetParamsTable**. Each row is organized as follows:

```
rowNumber=tpName tpMPModel tpSecurityModel tpSecurityName tpSecurityLevel
             tpStorageType tpRowStatus
```

Eg.

```
1=p1 mpSNMPv2c(1) secSNMPv2c(2) public noAuthNoPriv(1) nonVolatile(3) active(1)
```

See RFC 3413 for more information about the contents of this table.

[v3SnmpNotifyTable]

This is the non volatile representation of the **snmpNotifyTable**. Each row is organized as follows:

```
rowNumber=tnotifyName(ix) notifyTag notifyType storageType rowStatus
```

Eg.

```
      1=n1 tag1 trap(1) nonVolatile(3) active(1)
```

See RFC 3413 for more information about the contents of this table.

[v3SnmpNotifyFilterProfileTable]

This is the non volatile representation of the **snmpNotifyFilterProfileTable**. Each row is organized as follows:

```
rowNumber=targetParamsName notifProfileName storageType rowStatus
```

Eg.

```
1=p1 nfpn1 nonVolatile(3) active(1)
```

Note the external index `snmpTargetParamsName` is explicitly a part of the row. See RFC 3413 for more information about the contents of this table.

[v3SnmpNotifyFilterTable]

This is the non volatile representation of the **snmpNotifyFilterTable**. Each row is organized as follows:

```
rowNumber=profileName filtSubtree filtMask filtType storageType rowStatus
```

Note the external index **snmpNotifyFilterProfileName** is explicitly a part of the row. See RFC 3413 for more information about the contents of this table.

[v3ProxyTable]

This is the non volatile representation of the **snmpProxyTable**. Each row is organized as follows:

```
rowNumber=name type engineID ctxName tgtParamsIn sngTargetIn mltTargetOut
          StorageType RowStatus
```

Eg.

1=proxy1 read(1) 80:00:12:99:04:6e:64:74 "" p2 t1 "" nonVolatile(3) active(1)

See RFC 3413 for more information about the contents of this table.

[v3SnmpCommunityTable]

This is the non volatile representation of the **snmpCommunityTable**. Each row is organized as follows:

```
rowNumber=index community securityName osCtxEngineID ctxName transportTag
            eStorageType  eRowStatus
```

Eg.

```
1=c1 public public 00:00:12:99:7f:00:00:01:ab:cd:ef:bb "" "" nonVolatile(3)
         active(1)
```

See RFC 2576 for more information about the contents of this table.

[v3UsmUserTable]

This is the non volatile representation of the **snmpUSMUserTable**. Each row is organized as follows:

```
rowNumber=index community securityName osCtxEngineID ctxName transportTag
            eStorageType  eRowStatus
```

Eg.

```
1=EngineID UserName AuthProtocol PrivProtocol StorageType RowStatus
         LocalizedAuthKey LocalizedPrivKey
```

See RFC 3414 for more information about the contents of this table.

[v3VacmContextTable]

This is the non volatile representation of the **vacmContextTable**. Each row is organized as follows:

```
rowNumber=contextName
```

Eg.

```
1=""
```

The above depicts the default context. See RFC 3415 for more information about the contents of this table.

[v3VacmSecurityToGroupTable]

This is the non volatile representation of the **vacmSecurityToGroupTable**. Each row is organized as follows:

```
rowNumber=secModel secName groupName storageType rowStatus
```

Eg.

```
1=secSNMPv2c(2) public grpReadOnly nonVolatile(3) active(1)
2=secUSM(3) shaaes grpAll nonVolatile(3) active(1)
```

See RFC 3415 for more information about the contents of this table.

[v3VacmAccessTable]

This is the non volatile representation of the **vacmAccessTable**. Each row is organized as follows:

```
rowNumber=secModel secName groupName storageType rowStatus
```

Eg.

```
1=grpReadOnly "" secSNMPv2c(2) noAuthNoPriv(1) prefix(2) all all all
          nonVolatile(3) active(1)
2=grpAll "" secUSM(3) noAuthNoPriv(1) prefix(2) all all all nonVolatile(3)
          active(1)
```

See RFC 3415 for more information about the contents of this table.

[v3VacmViewTreeFamilyTable]

This is the non volatile representation of the **`vacmViewTreeFamilyTable`**. Each row is organized as follows:

```
rowNumber=viewName subtree familyMask familyType storageType rowStatus
```

Eg.

1=all 1.2 ff included(1) nonVolatile(3) active(1)
2=all 1.3 ff included(1) nonVolatile(3) active(1)

See RFC 3415 for more information about the contents of this table.

SNMP v1/v2c Options

The content described in this section affects the operation of the SNMP engine when the SNMP Agent is set to SNMPv1 or SNMPv2 mode. See the '`Version=`' expression in the `[SNMP Agent]` section.

Note, this section **does not** apply to SNMPv1 or SNMPv2 communications handled by the master agent when it is in SNMPv3 mode.

[v1v2cCommunityTbl]

This is the non volatile storage for managing v1/v2c community names. Each row has the following format:

```
rowNumber=viewName subtree familyMask familyType storageType rowStatus
```

Eg.

```
1=public readOnly(2)
2=private readCreate(4)
```

`CommunityName` is the name of the community being defined for the master agent.

`maxAccess` defines the maximum access granted for the community name. Values may be one of:

```
notify(1)
readOnly(2)
readWrite(3)
readCreate.(4)
```

[v1v2cTargetAddrTbl]

This is the non volatile storage for managing v1/v2c targets. Each row has the following format:

```
rowNumber=name address timeout retryCount tagList community version
```

E.g.

```
1=t1 192.168.0.1:162 0 0 tag1 public 1
2=t2 10.0.0.1:162 0 0 tag2 public 2
```

Where:

`version` is the SNMP version (1 or 2) to send the trap with.

[v1v2cNotifyTbl]

This is the non volatile storage for managing v1/v2c notifications. Each row has the following format:

```
rowNumber=notifyTag notifyType
```

E.g.

```
1=tag1 trap(1)
2=tag2 inform(2)
```

ndagdext.xnv Options

(Installed under the `'/etc'` directory)

This file controls the configuration of dynamically loadable sub agents.

As a means of illustrating the information in the `ndagdext.xnv` file, following is the taken from the default content of the file.

```
[ExtensionAgents]
1=NDMIB2
2=NDHost
3=NDMIB2-6

[NDMIB2]
Load=0
Path=libNDMIB2.so.0[26]
Web=$(NDWEBROOT)/NDMIB2

[NDHost]
Load=1
Path=libNDHost.so.1
Web=$(NDWEBROOT)/NDHost

[NDMIB2-6]
Load=1
Path=libNDMIB2-6.so.0
Web=$(NDWEBROOT)/NDMIB2-2
```

The section identified by `[ExtensionAgents]` is used to enumerate the sub agents that are available to load or unload. In this case, three are defined; two for "MIB2", `NDMib2` & `NDMib2-6` and for host resources, `NDHost`.

The right side of the expression defines a section name in which sub agent specific information is found. In each such named section are three expressions defined as follows:

`Load=` takes a value of '`0`' or '`1` '. When the value is '`1`', it indicates that the sub agent is to be loaded the next time the file is evaluated, otherwise it is unloaded. (See the –r command line option for the (daemon) agent)

`Path=` specifies the path to the sub agent shared object library.

[26] Library paths may be either specific or rely on the library being on the library load path (as depicted). Alternately...

E.g. **`Path=/usr/local/NuDesign/libs/libNDMIB2.so.0`**

Web= specifies to path to the HTML files associated with this sub agent[27]. The macro **$(NDWEBROOT)** may be used in the path to specify a path relative to the "**Root**" web path specified in **ndagd.xnv**.

(Note: the Web path examples above will **likely** need to be set appropriate for the target configuration. Also **note** the above examples happen to be for the typical installation location of the xAgentBuilder SDKs for this platform.

[27] Only used when your license includes the embedded IPv4 HTTP/HTTPS web server.

FIPS Mode Operation

Selecting **`ForceFIPSMode=1`** mode of operation in the **`[General]`** section, forces the use of only FIPS 140.2 compliant protocols and changes a number of behaviors in the service, in addition to not allowing non FIPS 140.2 compliant protocols, commonly used in SNMPv3:

1. Selecting FIPS 140.2 does not change the current service configuration. This mode operates by altering certain behaviors of typical SNMPv3 operations, so that switching between FIPS and non FIPS mode does not force changes to the current configuration.

2. When enabled, the agent only responds to SnmpV3 **`authPriv`** requests. Moreover, while in this mode when using **`authPriv`**, it does not produce a Response message when using MD5 authentication or DES privacy.

3. If other request types are submitted to the agent, while in this mode, then the following rules apply:

   - There will be no response returned to V1 or V2 requests.
   - If a SNMPv3 **`noAuthNoPriv`** is submitted, an "***`unsupported security level`***" report is sent.
   - If **MD5** is used for authentication, in either a **`authNoPriv`** or **`authPriv`** request, a "***`wrong digest`***" report is sent.
   - If in an **`authNoPriv`** request, not using **MD5** for authentication, then an "***`unsupported security level`***" report is sent.
   - If **DES** is used in an **`authPriv`** request, a "***`decryption error`***" report is returned.

4. When in this mode, the service also does not emit V1, V2 traps or informs, nor V3 traps or informs that do not meet item 2 above.

5. The only "in the clear" response that is emitted while in this mode, is the response to an SNMPv3 "discovery" request.

Dynamic V3 SNMP Engine ID and USM User Creation

This pre-configuration phase is conducted in response to the agent finding the file "**`NDMPDynConfig.xnv`**" in the same directory as the executable at startup.

Since the contents of this file usually contains sensitive information, after processing, the file is erased, so any backup copy you've created becomes the only record of this information.

The intention of this facility is to assist an administrator in the initial deployment of one or more master agents.

A sample file's contents can be viewed at this link.

This processing has three components: **EngineID** generation and **USM UserTable** creation as well as couple miscellaneous engine behavior options.

V3 SNMP EngineID Generation

EngineID generation is based on the value of '**`EngineIdGenMode`**' in **`NDMPDynConfig.xnv`** and has seven different values, implying different modes of creation:

Gen Mode	Function
0	(default) do not change current engine id.
1	Use the current IPv4 address of the first non-loop back interface found. E.g. `80:00:12:99:01:C0:A8:00:01` (given an IP address of 192.168.0.1)
2	Use the current IPv6 address of the first non-loop back interface found.
3	Use the current MAC address of the first non-loop back interface found. E.g. `80:00:12:99:03:01:02:03:04:05:06` (given a MAC address of `01:02:03:04:05:06`)
4	Use the current text provided. E.g. `80:00:12:99:04:12:99:04:6E:64:74:2D:69:6E:63:2E:63:6F:6D` (given **EngineIdUseText** below)
5	Use the current octets provided. See **EngineIdUseOctets** below E.g. `80:00:12:99:05:7f:00:00:01:ab:cd:ef:bc` (given **EngineIdUseOctets** below)
6	Use the current octets provided, but do not insert the method into the resulting engine id. See **EngineIdUseOctets** below

	E.g. `00:00:12:99:7f:00:00:01:ab:cd:ef:bc` (given **EngineIdUseOctets** below)

Note: With methods 1 to 5, the method number is inserted into the **EngineId** after the **`EngineIdPrefix`**

An example copy of **`NDMPDynConfig.xnv`** can be found in the **`doc`** directory of the product.

USM Table Initialization

The USM User Table initialization allows an initial USM table to be put in place and any adjustments to related tables, such as the VACM tables, will also be made.

There are only two modes of operation of this function. To do nothing, leaving all tables as is. Alternately, it will replace the USM table with the information supplied in this XNV file, making any necessary changes to other tables, such as the VACM tables. You will note the data supplied for the USM User Table will look like:

```
4=shades SHA(3) DES(2) "shadesauth" "shadespriv" "grpAll"
```

includes an optional field ("**grpAll**" in this case) which is used while setting up the VACM tables.

It is important to note that if you elect to install a new USM User Table, any previous table information will be lost.

When configuring this section, you may use the following authentication protocols:

- none(1)
- MD5(2)
- SHA(3)
- SHA224(4)
- SHA256(5)
- SHA384(6)
- SHA512(7)

You may also use the following privacy protocols:

- none(1)
- DES(2)

- AES128(4)
- AES192(5)
- AES256(6)
- 3DES(7)

Contents of Sample NDMPDynConfig.xnv

```
; *******
;
; Configuration file to dynamically create engineid and
; corresponding USM table.
;;
; The v3CommunityTable is modified if any entries are found to contain the
; the old engine id.   Entries are updated with the new engine id.
;;
; This file is erased on completion.  Make a backup if you need it.
;;
;
; *******
;;
; EngineIdGenMode
; ===============
;;
; integer value.  Valid values: 0 - 6.
;;
; Where:
;;
; 0 - (default) do not change current engine id.
;;
; 1 - Use the current IPv4 address of the first non-loop back interface found.
;;
; E.g. 80:00:12:99:01:C0:A8:00:01 (given an IP address of 192.168.0.1)
;;
; 2 - Use the current IPv6 address of the first non-loop back interface found.
;; ( not implemented at this time.)
;;
; 3 - Use the current MAC address of the first non-loop back interface found.
;;
; E.g. 80:00:12:99:03:01:02:03:04:05:06
;             (given a MAC address of 01:02:03:04:05:06)
;;
; 4 - Use the current text provided.
;;
; E.g. 80:00:12:99:04:12:99:04:6E:64:74:2D:69:6E:63:2E:63:6F:6D
;             (given EngineIdUseText below)
;;
; 5 - Use the current octets provided.  See EngineIdUseOctets below
;;
; E.g. 80:00:12:99:05:7f:00:00:01:ab:cd:ef:bc (given EngineIdUseOctets below)
;;
; 6 - Use the current octets provided, but do not insert the method into the
;     resulting engine id.  See EngineIdUseOctets below
;;
; E.g. 00:00:12:99:12:7f:00:00:01:ab:cd:ef:bc (given EngineIdUseOctets below)
;;
; Note: With methods 1 to 5, the method number is inserted into the engine id
; after the EngineIdPrefix.
;;
```

```
; EngineIdPrefix
; ==============
;;
; 4 octet string, specifying first four octets of the engine id.
;;
; The first bit of the first octet will be set to '1' if the generation mode
; has a value 1 - 5.
;;
; default prefix: "00:00:12:99", implying the default after first bit set:
; "80:00:12:99" for methods 1 - 5.
;;
; USMGenMode
; ==========
;;
; integer value.  Valid values: 0 - 1.
;;
; Where:
;;
; 0 - do not change USM table
;;
; 1 - (default) create table from provided USM table.  Note:
; this option will delete existing entries in the table.
;;

; SysNameMode
; ===========
;;
;      Optional: integer value.  Valid value: 1
;;
;      When set to '1' the service will be configured to use the current
;      host name as the value for sysName.0.   Any other value or the absence
;      of the parameter will leave the setting of this mode and hence
;      sysName.0 to the current (or default) mechanisms.
;;

; SysDescrMode
; ============
;;
;      Optional: integer value.  Valid value: 1
;;
;      When set to '1' the service will be configured to create a string
;      from computer info as the value for sysDescr.0.   Any other value
;      or the absence of the parameter will leave the setting of this mode
       and hence, sysDescr.0 to the current (or default) mechanisms.
;
;;
[DynConfig]
EngineIdGenMode=3
EngineIdPrefix=80:00:12:99
EngineIdUseOctets=7f:00:00:01:ab:cd:ef:bc
EngineIdUseText="ndt-inc.com"
USMGenMode=1
SysNameMode=1
SysDescrMode=1
;;
; The last column in the table below is used to create a row in the
; vacmSecurityToGroupTable for each USM entry.
;;
;   Acceptable values for authentication:
;;
;   none(1)
```

```
;   MD5(2)
;   SHA(3)
;   SHA224(4)
;   SHA256(5)
;   SHA384(6)
;   SHA512(7)
;;
;   Acceptable values for privacy:
;;
;   none(1)
;   DES(2)
;   AES128(4)
;   AES192(5)
;   AES256(6)
;   3DES(7)

[DynUsmUserTable]
;
;index=UserName AuthProtocol PrivProtocol AuthPassword PrivPassword [GroupName:
default=grpReadOnly]
1=md5 MD5(2) none(1) "md5auth" "" "grpAll"
2=sha SHA(3) none(1) "shaauth" "" "grpAll"
3=md5des MD5(2) DES(2) "md5desauth" "md5despriv" "grpAll"
4=shades SHA(3) DES(2) "shadesauth" "shadespriv" "grpAll"
5=public none(1) none(1) "" "" "grpReadOnly"
6=shaaes SHA(3) AES128(4) "shaaesauth" "shaaespriv"
7=md5aes MD5(2) AES128(4) "md5aesauth" "md5aespriv"
8=md5nopriv MD5(2) none(1) "md5noprivauth" "" "grpAll"
9=shanopriv SHA(3) none(1) "shanoprivauth" "" "grpAll"
10=sha256aes256 SHA256(5) AES256(6) "sha256aes256auth" "sha256aes256priv"
"grpAll"
11=sha3des SHA(3) 3DES(7) "sha3desauth" "sha3despriv" "grpAll"
12=sha224aes SHA224(4) AES128(4) "sha224aesauth" "sha224aespriv" "grpAll"
13=sha256aes SHA256(5) AES128(4) "sha256aesauth" "sha256aespriv" "grpAll"
14=sha256des SHA256(5) DES(2) "sha256desauth" "sha256despriv" "grpAll"
15=shaaes192 SHA(3) AES192(5) "shaaes192auth" "shaaes192priv" "grpAll"
16=sha224aes192 SHA224(4) AES192(5) "sha224aes192auth" "sha224aes192priv"
"grpAll"
17=sha512aes256 SHA512(7) AES256(6) "sha512aes256auth" "sha512aes256priv"
"grpAll"
```

NDT Default V3 SNMP Community/USM Configuration

By default, the following configurations are available for accessing the agent. Note: the configuration provided is the default configuration for all NuDesign SNMP products and as a result is more or less public information. As such it should only be used for a test and or development deployment. Deploying this configuration in a production environment is **NOT** recommended.

public	V1/V2c read-only community string with access to all implemented MIB objects.
private	V1/V2c read-write community string with access to all implemented MIB objects.
public	**noAuthPriv**, V3 read-only USM User with access to all implemented MIB objects. No authorization or privacy passwords.
private	**authPriv** (MD5, DES), V3 read-write USM User with access to all implemented MIB objects. Auth password: **privateauth**, privacy password: **privatepriv**.
md5	**authNoPriv** (MD5), V3 read-write USM User with access to all implemented MIB objects. Auth password: **md5auth**, privacy password: none. (Status: deprecated)
sha	**authNoPriv** (SHA), V3 read-write USM User with access to all implemented MIB objects. Auth password: **shaauth**, privacy password: none. (Status: deprecated)
md5des	**authPriv** (MD5, DES), V3 read-write USM User with access to all implemented MIB objects. Auth password: **md5desauth**, privacy password: **md5despriv**.
shades	**authPriv** (SHA, DES), V3 read-write USM User with access to all implemented MIB objects. Auth password: **shadesauth**, privacy password: **shadespriv**.
md5aes	**authPriv** (MD5, AES128), V3 read-write USM User with access to all implemented MIB objects. Auth password: **md5aesauth**, privacy password: **md5aespriv**.
shaaes	**authPriv** (SHA, AES128), V3 read-write USM User with access to all implemented MIB objects. Auth password: **shaaesauth**, privacy password: **shaaespriv**.
md5nopriv	**authNoPriv** (MD5), V3 read-write USM User with access to all implemented MIB objects. Auth password: **md5noprivauth**, privacy password: none.
shanopriv	**authNoPriv** (SHA), V3 read-write USM User with access to all implemented MIB objects. Auth password: **shanoprivauth**, privacy password: **none**.

NDAgClient

Synopsis: `NDAgClient`

Description:

This application provides a console interface to a daemon that is currently running in the background with the -p option. In addition to the console interface documented in the section **'**Console Interface**'**, one additional console command is available. This is the **'exit'** command that causes this client application to exit.

There are two modes of operation for this application. When the application is started with no additional command line information, the application presents an interactive console interface similar to that which is available when the agent is run as a console application.

When the application is started with additional information on the command line, the additional information is presented to the daemon and executed. When a response is received from the daemon, the application outputs the daemon's response to `stdout` and exits.

Example 1:

```
ndagclient get sysDecr.0
```

The above causes the '`get sysDescr.0`' to be executed by the daemon. The response should be something like:

```
sysDescr = NuDesign Multiprotocol Agent for Linux
```

Redirecting `stdout` from this is a little "tricky", since the rest of the command line is presented to the daemon to execute. However the solution is simple, have the client command line execute in it's own shell then redirect the output of that shell however you want.

Example 2:

```
(ndagclient get sysDecr.0) > tmp.out
```

The above still causes the '`get sysDescr.0`' to be executed by the daemon, however using this syntax, it is run in a child shell, with the end of the command line being the '0'. The (shell) response is redirected to `tmp.out`. The contents of `tmp.out` should be something like:

```
sysDescr = NuDesign Multiprotocol Agent (SnapGear 3.3 Linux
on a Coyote board)
```

In this mode of operation, you can use the client from a script to perform some set of operations repetitively. Also note that in this mode the '**exit**' command is implied, so this is no need to use it.

> **Note**: There is a limitation of one active client application running at one time. If another client is running and another is started, the user will see a message indicating this. This will also happen when the agent is being run from the console. This application (and agent in console mode) uses a file "lock" on a temporary file to accomplish this. The file used is "`/tmp/NDAgD-client-lock`". If for any reason this client application is terminated abnormally (i.e. without invoking the '`exit`' command in the application) then the user may be required to remove the lock file if it was not removed by the terminated session.

`NDAgClient` should be run with `root` privileges. If the agent service daemon is running with `root` privileges, then **`NDAgClient`** MUST run with `root` privileges.

Console Interface

The following are the commands available from both the agent console (CLI) and `NDAgClient`. Note: details about parameter use are available via the '`?`' and '`help`' commands.

Command	Function
`q`	Stops the agent.
`?`	Displays a list of command options.
`help`	Displays help on a specific command.
`clivacm`	Displays or modifies the command line's VACM configuration.
`AgentXPort`	Shows the current TCP port on which AgentX requests are being processed.
`agents`	Displays, suspends or resumes the SNMP, FCGI or HTTP agents.
`agparams`	Displays current agent operations parameters.
`get`	Performs a get on an object.
`getnext`	Performs a get-next on an object.
`gget`	Performs a group get of a scalar group of objects. E.g. gget SNMPv2-MIB.system.
`mib`	Displays the list of MIB objects currently available in the agent.
`rget`	Performs a get of a row of a table.
`rgetnext`	Performs a get next of a row of a table.
`samport`	Show the current sub agent registration port.
`set`	Performs a set on an object.
`setti`	Performs a set and increment on an object. (For "Spinlock" variables.)
`snmp`	Display or manage SNMP trace facililty.
`tget`	Performs a get of a table. E.g. `tget RFC1213-MIB.ifTable.`
`vacm`	Display or manage the view access control table.
`version`	Display daemon version.
`walk`	Walk some or all objects in the agent.
`xdll`	Display load status of, load or unload sub agents.

Additionally, there is a circular command history buffer of the last 10 commands executed, available by using the up or down cursor keys.

When using **`NDAgClient`**, there are two additional operations available. These are:

Command	Function
`exit`	Exits NDAgClient, but leaves the daemon executing.
`!<command>`	Executes the provided `command` in the daemon's context. E.g. !ls –l There is a limitation in this mechanism in that the output from the requested application must go to stdout.