Enabling SNMPv3 Security Features in NET+OS V7.X

1 Document History

Date	Initials	Change Description
7/17/08	JZW	Initial Entry
7/18/08	JZW	Fill in chapters
7/21/08	JZW	Enter typo comments
7/21/08	JZW	Continue initial entry
7/23/08	JZW	Personal review and edit session
7/23/08	JZW	Add table of contents
7/28/08	JZW	Add in community v1/v2c backward compat info
8/13/08	JZW	Roll in RM Comments

2 Table of Contents

3 Introduction

SNMPV3 added security features to the already existing SNMPv1 and SNMPv2c. Security, in this case, can be broken down in to the following three components, authentication, privacy and viewport. This white paper discusses, methods and structures that NET+OS uses for implementing these features in NET+OS's V7.x SNMP implementation.

Authentication deals with ensuring that the user who is viewing an agents MIBs is allowed to access that MIB. It even ensures that the user is authorized to access this agent. Users who are not so authorized, should be denied access to those MIBS.

Privacy deals with ensuring that no third party can view the data passed (over the internet) between the management software/user and the agent. Privacy is usually implemented through encryption.

Lastly, what I am calling viewport, is the ability of the agent to restrict the breadth of the MIB(s) to which the user has access. This is differentiated from authentication as follows: The user may have access to the agent but only be able to view MIB2 or his company's private MIB, instead of the entire MIB.

3.1 Problem Solved

The methods and structures used in SNMPv3 for implementing security features have a wide berth and are highly flexible. Unfortunately, in flexibility comes complexity. SNMPv3 is no exception to this. For example, there are 5 RFCs that are needed to explain SNMPv3. Additionally, these do not talk about SNMP's data access, only the security features added in SNMPv3. William Stalling's book (which I believe is a great introduction to this topic) SNMP, SNMPv2, SNMPv3 and RMON1 & 2 book takes 121 pages to explain SNMPv3. So to even get the basic concepts down, there is a lot of reading to do.

What this white paper hopes to do is boil all this down into some explanations on how to get basic SNMPv3 features working, in a NET+OS environment. This includes setting up users, groups, viewports, authentication and privacy. This white paper will also explain added backward compatibility, thus allowing your SNMPv1 and V2 users to continue interacting with your device even though you have upgraded it to SNMPv3.

3.2 Audience

This white paper is written in a technical manner and is intended for a technical audience. This is intended for software engineers and other software practitioners that intend to take advantage of NET+OS's SNMPv3 feature set in their applications.

3.3 Assumptions

This white paper assumes that the reader is familiar with the following technologies:

- Digi's NET+OS development environment (V7.0 & above)
    - Green Hills IDE development environment

    - GNU development environment
        - ESP IDE development environment
        - Command line interface development environment
- TCP/IP networking
- SNMP (V1/V2c)
- NET+OS API reference guide
- Digi ARM-based chips, development boards and modules
- Developing applications using some combination of the above

Further, if you want to practice the use of the information used in this document we recommend starting with the snmpv3 sample application. Take that application and added structures and API calls as outlined in this document. We would recommend that you read though this document before attempting to make updates to the application. Build the application, down load it into your device, configure your favorite MIB browser and test out your updates.

3.4 Scope

The intention of this white paper is quite narrow, compared to what Digi's NET+OS development environment offers. This white paper's focus is adding SNMPv3 security features to NET+OS applications running with SNMPv3.

This paper does not address any of the following topics:

- Developing applications under any of Digi International's NET+OS development environments
- TCP/IP networking
- C programming
- Debugging applications
- Writing, compiling, debugging or including SNMP MIBS
- Writing stub functions for SNMP MIBs
- Porting NET+OS V6.x (V5.x, V4.x…) SNMP-related applications to NET+OS V7.x
- Using any SNMP-related management software

3.5 Theory of Operation

There are five main structures that define SNMP v3 security in NET+OS. Four of them deal with security in general. The fifth deals with associating an existing user with an SNMPv1 or SNMPv2c community name, thus providing backward compatibility. The security-related structures will be explained. Then the APIs that allow the developer to insert entries into those structures will be explained. In a separate section, this paper explains the structure used in implementing SNMPv1 and SNMPv2c compatibility. This paper will also demonstrate a number of examples of the structures and API calls required to set up users in different configurations.

3.6 Conventions

4 SNMPv3 Security Structures in NET+OS V7.x

4.1 Introduction

There are five basic structures that define a security environment in NET+OS's snmp implementation. This section describes these data structures. The following section, describing APIs, will talk about putting structure entries and APIs together to form a user access definitions. I will use examples from the example application (snmpv3) to illustrate structure contents.

4.2 Caveats, placement of API calls

The (NET+OS) treck TCP/IP stack, sets up an initial SNMPv3 security environment, prior to control being given to applicationStart(). The presumption is that to enable SNMP, somewhere after applicationStart, naSnmpStart() will be called. If you set up additional entries, by using the APIs and data structures described below PRIOR to calling naSnmpStart(), the treck initial SNMP environment, relating to that API and entry type, will be lost. This may be what you want, as you might want FULL control of all users, groups, vtfs et al. In addition, this could disable/delete the public/private (SNMPv1/2c) community names set up in the initial treck environment. If, on the other hand you want to retain the treck-related SNMP initial SNMP environment settings, you must call the API calls described below AFTER calling naSnmpStart().

4.3 User (USM) Structure

This defines user names. The entry data structure is defined by the following type: NaSnmpUsmEntry_t. Along with the user name, a number of other fields need to be set.

The following table contains the fields that are most important:

Field Name	Function
Name	This will be the users name
SecurityName	This is generally set to the same as the Name field
authProtocol	Defines the authentication protocol.
privProtocol	Defines the privacy protocol

Generally the name field and the securityName field are set to the same string. This is the name that the MIB browser will use to gain access to the agent. The authProtocol can be set to either "none" or HMAC-MD5. privProtocol is the privacy protocol used to encrypt data. privProtocol can be set to either "none" or CDC-DES. Please see the API reference for the actual values used in these fields.

4.4 Security to Group Structure

This structure establishes a link between users and groups. The entry data structure is defined by the following data structure: NaSnmnpVacmSecuritytoGroupEntry_t. The

system uses the user name to get to the correct entry in this structure. Using the group and the security model (v1, v2c, v3), from this entry, the system can then gain access to the user's access rights.

The following table contains the fields that are most important:

Name	Function
Security model	V1, V2c, V3
SecurityName	Users name
groupName	The group name associated with this user and securityModel

Please refer to the API reference guide for additional details on this subject.

4.5 Access Structure

A combination of group name, context prefix, security model and security level are used to access an entry in this field. The result are accesses to view tree (I called them viewports) entries. View tree entries define the length and breadth of access to portions (or all) of a MIB. If your user successfully logs into the agent but your MIB view is empty, ensure that you called the APIs that add these structures to the system. Also ensure that the correct MIB view trees are referenced in these structures. Last ensure that the view trees are defined (exist).

The following table contains the fields that are most important:

Name	Function
Security model	V1, v2c, v3
Read view	Depth and breadth of the MIB from which I can read
Write view	Depth and breadth of the MIB to which I can write
Notify view	Depth and breadth of the MIB for which I can send traps

4.6 View tree Structure

The view tree table defines the length and breadth of an OID to which the user has access. References to these are included in an access structure for defining the read, write and notification access. If your user successfully logs into the agent but you can not access either the entire MIB or a portion that is important to you, check to ensure that these structures are correct.

The following are the most important fields of this structure:

Subtree	The OID or part of OID to which you want access
Mask	Combined with the subtree to define access. See the explanation of mask below.
View tree type	Will this subtree be included or excluded from view by the user

Tree type	Included or excluded

The mask field needs a further explanation, thus I quote from Appendix 17A of Stallings book SNMP, SNMPv2, SNMPv3 and RMON I and II: "If the mask bit is 1, the sub identifier is used; If the bit is 0, then the corresponding sub identifier is treated as a 'wildcard' in which any value may appear."

4.7 Community Structure

This table makes backward compatibility to V1 and V2c community names. Please see the section entitled V1/V2c community Compatibility for information about this.

Community index	Must be unique. Generally a string. Will be used as an index into this table
Community name	The name of the community. "public" and "private" are the most common but any other valid string is acceptable
Community security name	The V1 or V2 user name that the agent will reference when a community string is used in accessing the agent

5 SNMPv3 Security Structure-related APIs

5.1 Introduction

In this section, the question "how do I put the structures and APIs together to set up particular user configurations?", is explored and hopefully provides you with answers. This section deals exclusively with SNMPv3 users. In the section entitled V1/V2 Community Compatibility, setting up communities in V1 and V2c is explained. I will use structures and code from the NET+OS sample application nasnmpv3 to demonstrate what is required. In the structure definitions, I am including the "bare" strings, in fields where strings are used. In your use, I would expect (as is done in example application nasnmpv3) to use manifest constants, such as
#define DIGI_USER "digi_user". I use "bare" strings here to ensure clarity.

Please note the following: Across these examples certain entries are inserted more than once. In reality, for example, allMibs needs to be inserted only once. For documentation purposes and to allow each example to stand alone, I repeat the entry insert API calls.

5.2 User with authentication and encryption

5.2.1 Introduction

The following set of structure initializers might be used to set up a user, with full access to the set of MIBs. In addition, this user will be authenticated and all communications are encrypted.

5.2.2 Data structures

```
NaSnmpUsmEntry_t secureUser =
{
   “secure_user_auth_and_priv”,
   (sizeof “secure_user_auth_and_priv”) - 1,
   “secure_user_auth_and_priv”,
   (sizeof “secure_user_auth_and_priv”) - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_HMACMD5_AUTH_PROTOCOL,
   “the_auth_password”,
   (sizeof “the_auth_password”) - 1,
   "",
   0,
   NA_SNMP_DES_PRIV_PROTOCOL,
   “the_privacy_password”,
   (sizeof “the_privacy_password”) - 1,
   "",
   0,
   “public”,
   (sizeof “public”) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

For simplicity, user name and secure user name are the same string. We are using HMAC for authentication and DES for encryption. Both will need passwords. Further we have chosen not to utilize keys along with the passwords for authentication and encryption. This is clearly a decision that the implementer makes.

We now need to define a group. This will be used later. This defines access and can be shared by multiple users, if needed.

```
NaSnmpVacmAccessEntry_t accessForAuth_and_priv_User =
{
   “auth_and_priv_group”,
   (sizeof “auth_and_priv_group”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_USM,
```

```
   NA_SNMP_VACM_SECURITY_LEVEL_AUTH_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   “allMib”,
   (sizeof “allMib”) - 1,
   “allMib”,
   (sizeof “allMib”) - 1,
   “allMib”,
   (sizeof “allMib”) - 1,
   NA_SNMP_PERMANENT
};
```

NA_SNMP_VACM_SECURITY_MODEL_USM means that this is defined using the snmpV3 security model. We’ll use v1 and v2 later. Do not worry about the string “allMib, right now. We will address it later. Take it as this user has access to the entire MIB. Also you’ll notice three initializers using the view “allMib”. The first defines read access. The second defines write access and the third defines notification access.

```
NaSnmpVacmViewTreeFamilyEntry_t theAllMibVTF =
{
   "allMib",
    (sizeof "allMib") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

This is the definition of allMib as referenced above. It gives full and unbridled access to the entire MIB. Clearly this is only for an authenticated user or a user with read rights only. NA_SNMP_VACM_INCLUDED states that the entire OID presented in the structure is included in what the user can access.

```
NaSnmpVacmSecurityToGroupEntry_t s2gForSecureUser =
{
   NA_SNMP_VACM_SECURITY_MODEL_USM,
   "secure_user_auth_and_priv",
   (sizeof "secure_user_auth_and_priv") - 1,
   "auth_and_priv_group",
   (sizeof "auth_and_priv_group") - 1,
   NA_SNMP_PERMANENT
};
```

This structure links the user to the group.
NA_SNMP_VACM_SECURITY_MODEL_USM states that this is an SNMP v3 security model. So when this entry is inserted, user "secure_user_auth_and_priv" will be associated with group "auth_and_priv_group".

5.2.3 API calls

The following API calls using the data structures mentioned above, would be used to set up this user environment.

```
naSnmpInsertUserEntry(&secureUser);
naSnmpInsertS2GEntry(&s2gForSecureUser);
naSnmpInsertVacmAccessEntry(&accessForAuth_and_priv_User);
naSnmpInsertVtfEntry(&theAllMibVTF);
```

5.3 User with authentication only

5.3.1 Introduction

In this example, the user is authenticated by the communications are not encrypted. As in the last example, the user has full access to the MIB.

5.3.2 Data structures

```
NaSnmpUsmEntry_t secureUserAuthOnly =
{
   "secure_user_auth_only",
   (sizeof "secure_user_auth_only") - 1,
   secure_"user_auth_only",
   (sizeof "secure_user_auth_only") - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_HMACMD5_AUTH_PROTOCOL,
   "the_auth_password",
```

```
   (sizeof “the_auth_password”) - 1,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   “”,
   0,
   "",
   0,
   “public”,
   (sizeof “public”) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

For simplicity, user name and secure user name are the same string. We are using HMAC for authentication. Also you will notice that no encryption is used and thus no encryption password is required or included. Further we have chosen not to utilize keys along with the passwords for authentication and encryption. This is clearly a decision that the implementer makes.

We now need to define a group. This will be used later. This defines access and can be shared by multiple users, if needed.

```
NaSnmpVacmAccessEntry_t accessForAuthOnly =
{
   “auth_only_group”,
   (sizeof  “auth_only_group”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_USM,
   NA_SNMP_VACM_SECURITY_LEVEL_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   “allMib”,
   (sizeof “allMib”) - 1,
   “allMib”,
   (sizeof “allMib”) - 1,
   “allMib”,
   (sizeof “allMib”) - 1,
   NA_SNMP_PERMANENT
};
```

NA_SNMP_VACM_SECURITY_MODEL_USM means that this is defined using the snmpV3 security model. We’ll use v1 and v2 later. Do not worry about the string “allMib, right now. We will address it later. Take it as this user has access to the entire MIB. You’ll also notice three initilizers that contain the view “allMib”. The first refers to read access. The second to write access and the third to notification access.

```
NaSnmpVacmViewTreeFamilyEntry_t theAllMibVTF =
{
   "allMib",
    (sizeof "allMib") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
          1,
          {0x00},
          1,
          NA_SNMP_VACM_INCLUDED,
          NA_SNMP_PERMANENT
};
```

This is the definition of allMib as referenced above. It gives full and unbridled access to the entire MIB. Clearly this is only for an authenticated user or a user with read rights only. NA_SNMP_VACM_INCLUDED states that the entire OID presented in the structure is included in what the user can access.

```
NaSnmpVacmSecurityToGroupEntry_t s2gForSecureUserAuthOnly =
{
   NA_SNMP_VACM_SECURITY_MODEL_USM,
   "secure_user_auth_only",
   (sizeof "secure_user_auth_only") - 1,
   "auth_only_group",
   (sizeof "auth_only_group") - 1,
   NA_SNMP_PERMANENT
};
```

This structure links the user to the group.
NA_SNMP_VACM_SECURITY_MODEL_USM states that this is an snmp v3 security model. So when this entry is inserted, user "secure_user_auth_only" will be associated with group "auth_only_group".

5.3.3 API calls

The following API calls using the data structures mentioned above, would be used to set up this user environment.

```
naSnmpInsertUserEntry(&secureUserAuthOnly);
naSnmpInsertS2GEntry(&s2gForSecureUserAuthOnly);
naSnmpInsertVacmAccessEntry(&accessForAuthOnly);
naSnmpInsertVtfEntry(&theAllMibVTF);
```

5.4 User with neither authentication no encryption (open)

5.4.1 Introduction

In this example, the user is not authenticated and any data moved between agent and management software is not encrypted. As in the last example, the user has full access to the MIB.

5.4.2 Data Structures

```
NaSnmpUsmEntry_t insecureUser =
{
   "insecure_user",
   (sizeof "insecure_user") - 1,
   "insecure_user",
   (sizeof "insecure_user") - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
```

```
   NA_SNMP_NO_AUTH_PROTOCOL,
   “”,
   0,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   “”,
   0,
   "",
   0,
   “public”,
   (sizeof “public”) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

For simplicity, user name and secure user name are the same string. We are employing NO authentication protocol. Also you will notice that no encryption is used and thus no encryption password is required or included. Further we have chosen not to utilize keys along with the passwords for authentication and encryption. This is clearly a decision that the implementer makes.

We now need to define a group. This will be used later. This defines access and can be shared by multiple users, if needed.

```
NaSnmpVacmAccessEntry_t accessForOpen =
{
   “open_group”,
   (sizeof  “open_group”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_USM,
   NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   “allMib”,
   (sizeof “allMib”) - 1,
   “allMib”,
   (sizeof “allMib”) - 1,
   “allMib”,
   (sizeof “allMib”) - 1,
   NA_SNMP_PERMANENT
};
```

NA_SNMP_VACM_SECURITY_MODEL_USM means that this is defined using the snmpV3 security model. We’ll use v1 and v2 later. Do not worry about the string “allMib, right now. We will address it later. Take it as this user has access to the entire MIB.

You'll also notice three initilizers that contain the view "allMib". The first refers to read access. The second to write access and the third to notification access. Also notice that NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV is used. This means that this group represents users with neither authentication needs nor encryptions needs.

```
NaSnmpVacmViewTreeFamilyEntry_t theAllMibVTF =
{
   "allMib",
    (sizeof "allMib") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

This is the definition of allMib as referenced above. It gives full and unbridled access to the entire MIB. Clearly this is only for an authenticated user or a user with read rights only. NA_SNMP_VACM_INCLUDED states that the entire OID presented in the structure is included in what the user can access.

```
NaSnmpVacmSecurityToGroupEntry_t s2gForInSecureUser =
{
   NA_SNMP_VACM_SECURITY_MODEL_USM,
    "insecureUser",
   (sizeof "insecureUser") - 1,
   "open_group",
   (sizeof "open_group") - 1,
   NA_SNMP_PERMANENT
};
```

This structure links the user to the group.
NA_SNMP_VACM_SECURITY_MODEL_USM states that this is an snmp v3 security model. So when this entry is inserted, user "insecureUser" will be associated with group "open_group".

5.4.3 APi Calls

The following API calls using the data structures mentioned above, would be used to set up this user environment.

```
naSnmpInsertUserEntry(&insecureUser);
naSnmpInsertS2GEntry(&s2gForInSecureUser);
naSnmpInsertVacmAccessEntry(&accessForOpen);
naSnmpInsertVtfEntry(&theAllMibVTF);
```

5.5 User with neither authentication nor encryption but restricted view

5.5.1 Introduction

In this example, the user is not authenticated and any data moved between agent and management software is not encrypted. But, as opposed to prior users, this user will have very limited read access and no write access to the MIB.

5.5.2 Data Structures

```
NaSnmpUsmEntry_t insecureUserRestrictAccess =
{
   "insecure_user_restrict_access",
   (sizeof "insecure_user_restrict_access") - 1,
   "insecure_user_restrict_access",
```

```
   (sizeof "insecure_user_restrict_access") - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_NO_AUTH_PROTOCOL,
   "",
   0,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   "",
   0,
   "",
   0,
   "public",
   (sizeof "public") - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

For simplicity, user name and secure user name are the same string. We are employing NO authentication protocol. Also you will notice that no encryption is used and thus no encryption password is required or included. Further we have chosen not to utilize keys along with the passwords for authentication and encryption. This is clearly a decision that the implementer makes.

We now need to define a group. This will be used later. This defines access and can be shared by multiple users, if needed.

```
NaSnmpVacmAccessEntry_t accessForOpen_with_restrict =
{
   "open_group_with_restrict",
   (sizeof  "open_group_with_restrict") - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_USM,
   NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   "myMib",
   (sizeof "myMib") - 1,
   "none",
   (sizeof "none") - 1,
   "allMib",
   (sizeof "allMib") - 1,
   NA_SNMP_PERMANENT
};
```

NA_SNMP_VACM_SECURITY_MODEL_USM means that this is defined using the snmpV3 security model. We'll use v1 and v2 later. Do not worry about the string "myMib, right now. We will address it later. Take it as this user has access to a very limited viewport into the MIB. You'll also notice three initializers containing "myMib, "none" and allMib, respectively. The first refers to read access. The second to write access and the third to notification access. Thus, this user has no write access. Also notice that NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV is used. This means that this group represents users with neither authentication needs nor encryptions needs.

```
NaSnmpVacmViewTreeFamilyEntry_t theAllMibVTF =
{
   "allMib",
    (sizeof "allMib") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

This is the definition of allMib as referenced above. It gives full and unbridled access to the entire MIB. Clearly this is only for an authenticated user or a user with read rights only. NA_SNMP_VACM_INCLUDED states that the entire OID presented in the structure is included in what the user can access. For this user, this is applied to notify access only.

```
NaSnmpVacmViewTreeFamilyEntry_t theNoneVTF =
{
   "none",
    (sizeof "allMib") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_EXCLUDED,
         NA_SNMP_PERMANENT
};
```

This is the definition of the vtf "none" You will notice that it looks strikingly like allMib's definition. The difference being that allMib uses NA_SNMP_VACM_INCLUDED, while "none" uses NA_SNMP_VACM_EXCLUDED. Thus for allMib the entire is included in the view. In "none" the entire MIB is excluded from the view. This user has NO write access.

```
NaSnmpVacmViewTreeFamilyEntry_t theMyMibVTF =
{
   "myMib",
         (sizeof "myMib") - 1,
   {1,3,6,1,4,1,905,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         8,
         {0xFE},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

There are a couple of things to notice here. First, There is more than just 1 contained in the allowed OID. So we are selecting a particular part of the MIB to make part of our viewport. Second the {0xFE} is new. This tells the software to use the first 7 digits in the OID in the definition of the viewports. In essence, {0xFE} is ANDed with the OID to come up with the portion of the MIB to which this user has access. Also the use of NA_SNMP_VACM_INCLUDED says that we have access to this portion of the MIB to the exclusion of all other parts. Thus, this user has read access to a very limited portion of the MIB.

```
 NaSnmpVacmSecurityToGroupEntry_t s2gForInSecureUserRestricted =
{
   NA_SNMP_VACM_SECURITY_MODEL_USM,
    "insecure_user_restrict_access",
   (sizeof "insecure_user_restrict_access") - 1,
   "open_group_with_restrict",
   (sizeof "open_group_with_restrict") - 1,
   NA_SNMP_PERMANENT
};
```

This structure links the user to the group.
NA_SNMP_VACM_SECURITY_MODEL_USM states that this is an snmp v3 security model. So when this entry is inserted, user "insecure_user_restrict_access" will be associated with group "open_group_with_restrict".

5.5.3 API Calls

The following API calls using the data structures mentioned above, would be used to set up this user environment.

```
naSnmpInsertUserEntry(&insecureUserRestrictAccess);
naSnmpInsertS2GEntry(&s2gForInSecureUserRestricted);
naSnmpInsertVacmAccessEntry(&accessForOpen_with_restrict);
naSnmpInsertVtfEntry(&theMyMibVTF);
naSnmpInsertVtfEntry(&theNoneVTF);
naSnmpInsertVtfEntry(&theAllMibVTF);
```

6 V1/V2c Community Compatibility

6.1 Introduction

This section discusses using the community names public and private in V1 and V2c mode. It also discusses setting up your own community names. These allow backward compatibility to V1 and V2c users.

Community names work a little different then regular (v3) names. For v1 and v2c community names, there must be a user name, access and s2g set of structures set up. Then in the community entry, the community name is associated with the user name. Once this is done, the community name can access the agent. Its access rights will be the same as those of the associated user name.

6.2 V1 community with read only

6.2.1 Introduction

This user will have open read access but no write access. There will be a community name created and that community name can read access the agent though this user. We will go through all the steps that were performed for previous users but there will be an additional one, namely adding a community entry.

6.2.2 Data Structures

The following structure sets up a user named digiV1Community. The user uses no authentication and no encryption.

```
NaSnmpUsmEntry_t digiNoAuthNoEnc =
{
   “digiV1Open”,
   (sizeof “digiV1Open”) - 1,
   “digiV1Open”,
   (sizeof “digiV1Open”) - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_NO_AUTH_PROTOCOL,
   "",
   0,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   "",
   0,
   "",
   0,
   ““public””,
   (sizeof ““public””) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

digiVTF_seeAllMib is set up to give this user access to the entire MIB. This will be used for read access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeAllMib =
{
   "digiAllMib",
         (sizeof " digiAllMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
```

```
        {0x00},
        1,
        NA_SNMP_VACM_INCLUDED,
        NA_SNMP_PERMANENT
};
```

VTF_seeNone is set up to give the user access to none of the MIB. This will be used for write access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeNone =
{
   "digiNoneMib",
        (sizeof " digiNoneMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        1,
        {0x00},
        1,
        NA_SNMP_VACM_EXCLUDED,
        NA_SNMP_PERMANENT
};
```

The accessDigiV1Community is set up to give users assigned to this group read access to the entire MIB, no write access and full notify access.

```
NaSnmpVacmAccessEntry_t accessDigiV1Community =
{
 “digiV1Group”,
   (sizeof “digiV1Group”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv1,
   NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   "digiAllMib",         // read access to MIB
   (sizeof "digiAllMib") - 1,
   "digiNoneMib ",                 // write access to nothing
   (sizeof "digiNoneMib") -1,
   "digiAllMib",          // notify do anything you want
   (sizeof "digiAllMib") - 1,
   NA_SNMP_PERMANENT

};
```

This structure associates our newly made user name with a V1 group having the access models we want.

```
NaSnmpVacmSecurityToGroupEntry_t s2gForjzwV1 =
{
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv1,
   “digiV1Open”,
   (sizeof “digiV1Open”) - 1,
   “digiV1Group”,
   (sizeof “digiV1Group”) - 1,
   NA_SNMP_PERMANENT
};
```

This last step is new relative to the users we set up in the prior section. This sets up a community name and associates that community name with an existing user name. Remember that when creating v1 and v2c communities, to keep the community names the same. This is because generally you want to have v1 and v2c communities of the same name but different versions accessing the agent.

```
NaSnmpCommunityTableEntry_t digiV1CommunityEntry =
{
        “digiV1CommunityIndex”,
        “digiCommunity”,
        “digiV1Open”,
        "",
          ,
        "",
          ,
        "",
          ,
        (sizeof “digiV1CommunityIndex”) - 1,
        (sizeof “digiCommunity”) - 1,
        (sizeof “digiV1Open”) - 1,
        0,
        0,
        0,
        NA_SNMP_PERMANENT
};
```

6.2.3 API Calls

```
naSnmpInsertUserEntry(&digiNoAuthNoEnc);
naSnmpInsertVtfEntry(&digiVTF_seeAllMib);
naSnmpInsertVtfEntry(&digiVTF_seeNone);
naSnmpInsertS2GEntry(&s2gFordigiV1);
naSnmpInsertVacmAccessEntry(&accessDigiV1Community);
naSnmpInsertCommunityTableEntry(&digiV1CommunityEntry);
```

6.3 V1 community with read and write

6.3.1 Introduction

This user will have open read access but no write access. There will be a community name created and that community name can read access the agent though this user. We will go through all the steps that were performed for previous users but there will be an additional one, namely adding a community entry.

6.3.2 Data Structures

The following structure sets up a user named digiV1Community. The user uses no authentication and no encryption.

```
NaSnmpUsmEntry_t digiNoAuthNoEnc =
{
   “digiV1Open”,
   (sizeof “digiV1Open”) - 1,
   “digiV1Open”,
   (sizeof “digiV1Open”) - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_NO_AUTH_PROTOCOL,
   "",
   0,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   "",
   0,
   "",
   0,
   ““public””,
   (sizeof ““public””) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

digiVTF_seeAllMib is set up to give this user access to the entire MIB. This will be used for read access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeAllMib =
{
   "digiAllMib",
        (sizeof " digiAllMib ") - 1,
```

```
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

VTF_seeNone is set up to give the user access to none of the MIB. This will be used for write access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeNone =
{
   "digiNoneMib",
         (sizeof " digiNoneMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_EXCLUDED,
         NA_SNMP_PERMANENT
};
```

The accessDigiV1Community is set up to give users assigned to this group read access to the entire MIB, no write access and full notify access.

```
NaSnmpVacmAccessEntry_t accessDigiV1Community =
{
 “digiV1Group”,
   (sizeof “digiV1Group”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv1,
   NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   "digiAllMib",         // read access to  MIB
   (sizeof "digiAllMib") - 1,
   "digiAllMib",         // read access to MIB
   (sizeof "digiAllMib") - 1,
   "digiAllMib",          // notify do anything you want
   (sizeof "digiAllMib") - 1,
   NA_SNMP_PERMANENT

};
```

This structure associates our newly made user name with a V1 group having the access models we want.

```
NaSnmpVacmSecurityToGroupEntry_t s2gFordigiV1 =
{
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv1,
   “digiV1Open”,
   (sizeof “digiV1Open”) - 1,
   “digiV1Group”,
   (sizeof “digiV1Group”) - 1,
   NA_SNMP_PERMANENT
};
```

This last step is new relative to the users we set up in the prior section. This sets up a community name and associates that community name with an existing user name. Remember that when creating v1 and v2c communities, to keep the community names the same. This is because generally you want to have v1 and v2c communities of the same name but different versions accessing the agent. Important reminder, for each community you set up, its community index (first field in the structure) must be unique.

```
NaSnmpCommunityTableEntry_t digiV1CommunityEntry =
{
        “digiV1CommunityIndex”,
        “digiCommunity”,
        “digiV1Open”,
        "",
          ",
        "",
          ",
        "",
          ",
        (sizeof “digiV1CommunityIndex”) - 1,
        (sizeof “digiCommunity”) - 1,
        (sizeof “digiV1Open”) - 1,
        0,
        0,
        0,
        NA_SNMP_PERMANENT
};
```

6.3.3 API Calls

naSnmpInsertUserEntry(&digiNoAuthNoEnc);
naSnmpInsertVtfEntry(&digiVTF_seeAllMib);
naSnmpInsertS2GEntry(&s2gFordigiV1);
naSnmpInsertVacmAccessEntry(&accessDigiV1Community);
naSnmpInsertCommunityTableEntry(&digiV1CommunityEntry);

6.4 V2 community with read only

6.4.1 Introduction

6.4.2 Data Structures

The following structure sets up a user named digiV1Community. The user uses no authentication and no encryption.

```
NaSnmpUsmEntry_t digiV2NoAuthNoEnc =
{
   “digiV2Open”,
   (sizeof “digiV2Open”) - 1,
   “digiV2Open”,
   (sizeof “digiV2Open”) - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_NO_AUTH_PROTOCOL,
   "",
   0,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   "",
   0,
   "",
   0,
   ““public””,
   (sizeof ““public””) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

digiVTF_seeAllMib is set up to give this user access to the entire MIB. This will be used for read access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeAllMib =
{
   "digiAllMib",
   (sizeof " digiAllMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

VTF_seeNone is set up to give the user access to none of the MIB. This will be used for write access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeNone =
{
   "digiNoneMib",
         (sizeof " digiNoneMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_EXCLUDED,
         NA_SNMP_PERMANENT
};
```

The accessDigiV2Community is set up to give users assigned to this group read access to the entire MIB, no write access and full notify access.

```
NaSnmpVacmAccessEntry_t accessDigiV2Community =
{
 “digiV2Group”,
   (sizeof “digiV2Group”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv2c,
   NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   "digiAllMib",         // read access
   (sizeof "digiAllMib") - 1,
```

```
    "digiNoneMib",      // write access
    (sizeof "digiNoneMib") - 1,
    "digiAllMib",           // notify do anything you want
    (sizeof "digiAllMib") - 1,
    NA_SNMP_PERMANENT

};
```

This structure associates our newly made user name with a V2 group having the access models we want.

```
NaSnmpVacmSecurityToGroupEntry_t s2gForjzwV2 =
{
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv2c,
   “digiV2Open”,
   (sizeof “digiV2Open”) - 1,
   “digiV2Group”,
   (sizeof “digiV2Group”) - 1,
   NA_SNMP_PERMANENT
};
```

This last step is new relative to the users we set up in the prior section. This sets up a community name and associates that community name with an existing user name. Remember that when creating v1 and v2c communities, to keep the community names the same. This is because generally you want to have v1 and v2c communities of the same name but different versions accessing the agent. Important reminder, for each community you set up, its community index (first field in the structure) must be unique.

```
NaSnmpCommunityTableEntry_t digiV2CommunityEntry =
{
        “digiV2CommunityIndex”,
        “digiCommunity”,
        “digiV2Open”,
        "",
          ",
        "",
          ",
        "",
          ",
        (sizeof “digiV2CommunityIndex”) - 1,
        (sizeof “digiCommunity”) - 1,
        (sizeof “digiV2Open”) - 1,
        0,
        0,
        0,
        NA_SNMP_PERMANENT
};
```

6.4.3 API Calls

```
naSnmpInsertUserEntry(&digiV2NoAuthNoEnc);
naSnmpInsertVtfEntry(&digiVTF_seeAllMib);
naSnmpInsertVtfEntry(&digiVTF_seeNone);
naSnmpInsertS2GEntry(&s2gForjzwV2);
naSnmpInsertVacmAccessEntry(&accessDigiV2Community);
naSnmpInsertCommunityTableEntry(&digiV2CommunityEntry);
```

6.5 V2 community with read and write

6.5.1 Introduction

6.5.2 Data Structures

The following structure sets up a user named digiV2Community. The user uses no authentication and no encryption.

```
NaSnmpUsmEntry_t digiNoAuthNoEnc =
{
   “digiV2Open”,
   (sizeof “digiV2Open”) - 1,
   “digiV2Open”,
   (sizeof “digiV2Open”) - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_NO_AUTH_PROTOCOL,
   "",
   0,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   "",
   0,
   "",
   0,
   ““public””,
   (sizeof ““public””) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

digiVTF_seeAllMib is set up to give this user access to the entire MIB. This will be used for read access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeAllMib =
{
   "digiAllMib",
         (sizeof " digiAllMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

VTF_seeNone is set up to give the user access to none of the MIB. This will be used for write access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeNone =
{
   "digiNoneMib",
         (sizeof " digiNoneMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_EXCLUDED,
         NA_SNMP_PERMANENT
};
```

The accessDigiV2Community is set up to give users assigned to this group read access to the entire MIB, no write access and full notify access.

```
NaSnmpVacmAccessEntry_t accessDigiV2Community =
{
 “digiV2Group”,
   (sizeof “digiV2Group”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv2c,
   NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   "digiAllMib",         // read access to MIB
   (sizeof "digiAllMib") - 1,
```

```
    "digiAllMib",          // read access to MIB
    (sizeof "digiAllMib") - 1,
    "digiAllMib",          // notify do anything you want
    (sizeof "digiAllMib") - 1,
    NA_SNMP_PERMANENT

};
```

This structure associates our newly made user name with a V1 group having the access models we want.

```
NaSnmpVacmSecurityToGroupEntry_t s2gForjzwV2 =
{
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv2c,
   “digiV2Open”,
   (sizeof “digiV2Open”) - 1,
   “digiV2Group”,
   (sizeof “digiV2Group”) - 1,
   NA_SNMP_PERMANENT
};
```

This last step is new relative to the users we set up in the prior section. This sets up a community name and associates that community name with an existing user name. Remember that when creating v1 and v2c communities, to keep the community names the same. This is because generally you want to have v1 and v2c communities of the same name but different versions accessing the agent. Important reminder, for each community you set up, its community index (first field in the structure) must be unique.

```
NaSnmpCommunityTableEntry_t digiV2CommunityEntry =
{
        “digiV2CommunityIndex”,
        “digiCommunity”,
        “digiV2Open”,
        "",
          ",
        "",
          ",
        "",
          ",
        (sizeof “digiV2CommunityIndex”) - 1,
        (sizeof “digiCommunity”) - 1,
        (sizeof “digiV2Open”) - 1,
        0,
        0,
        0,
        NA_SNMP_PERMANENT
};
```

6.5.3 API Calls

```
naSnmpInsertUserEntry(&digiNoAuthNoEnc);
naSnmpInsertVtfEntry(&digiVTF_seeAllMib);
naSnmpInsertS2GEntry(&s2gForjzwV2);
naSnmpInsertVacmAccessEntry(&accessDigiV2Community);
naSnmpInsertCommunityTableEntry(&igiV2CommunityEntry);
```

6.6 V1 community with read only and restricted view

6.6.1 Introduction

6.6.2 Data Structures

The following structure sets up a user named digiV1Community. The user uses no authentication and no encryption.

```
NaSnmpUsmEntry_t digiNoAuthNoEnc =
{
   “digiV1OpenRest”,
   (sizeof “digiV1OpenRest”) - 1,
   “digiV1OpenRest”,
   (sizeof “digiV1OpenRest”) - 1,
   {1,3,6,1,4,1,115,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   8,
   NA_SNMP_NO_AUTH_PROTOCOL,
   "",
   0,
   "",
   0,
   NA_SNMP_NO_PRIV_PROTOCOL,
   "",
   0,
   "",
   0,
   ““public””,
   (sizeof “public”) - 1,
   NA_SNMP_PERMANENT,
   NA_SNMP_ROW_STATUS_ACTIVE
};
```

VTF_seeNone is set up to give the user access to none of the MIB. This will be used for write access.

```
NaSnmpVacmViewTreeFamilyEntry_t theMyMibVTF =
{
   "myMib",
         (sizeof "myMib") - 1,
   {1,3,6,1,4,1,905,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         8,
         {0xFE},
         1,
         NA_SNMP_VACM_INCLUDED,
         NA_SNMP_PERMANENT
};
```

VTF_seeNone is set up to give the user access to none of the MIB. This will be used for write access.

```
NaSnmpVacmViewTreeFamilyEntry_t digiVTF_seeNone =
{
   "digiNoneMib",
         (sizeof " digiNoneMib ") - 1,
   {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
         1,
         {0x00},
         1,
         NA_SNMP_VACM_EXCLUDED,
         NA_SNMP_PERMANENT
};
```

The accessDigiV1Community is set up to give users assigned to this group read access to the entire MIB, no write access and full notify access.

```
NaSnmpVacmAccessEntry_t accessDigiV1Community =
{
 “digiV1GroupRest”,
   (sizeof “digiV1GroupRest”) - 1,
   "",
   0,
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv1,
   NA_SNMP_VACM_SECURITY_LEVEL_NO_AUTH_NO_PRIV,
   NA_SNMP_VACM_ACCESS_CONTEXT_MATCH_EXACT,
   "myMib",    // read access to MIB
   (sizeof "myMib") - 1,
   "myMib",                 // write access to nothing
```

```
    (sizeof "myMib") -1,
    "myMib",     // notify do anything you want
    (sizeof "myMib") - 1,
    NA_SNMP_PERMANENT

};
```

This structure associates our newly made user name with a V1 group having the access models we want.

```
NaSnmpVacmSecurityToGroupEntry_t s2gFordigiV1 =
{
   NA_SNMP_VACM_SECURITY_MODEL_SNMPv1,
   “digiV1OpenRest”,
   (sizeof “digiV1OpenRest”) - 1,
   “digiV1GroupRest”,
   (sizeof “digiV1GroupRest”) - 1,
   NA_SNMP_PERMANENT
};
```

This last step is new relative to the users we set up in the prior section. This sets up a community name and associates that community name with an existing user name. Remember that when creating v1 and v2c communities, to keep the community names the same. This is because generally you want to have v1 and v2c communities of the same name but different versions accessing the agent.

```
NaSnmpCommunityTableEntry_t digiV1CommunityEntry =
{
        “digiV1CommunityIndex”,
        “digiCommunityRest”,
        “digiV1OpenRest”,
        "",
          ,
        "",
          ,
        "",
          ,
        (sizeof “digiV1CommunityIndex”) - 1,
        (sizeof “digiCommunityRest”) - 1,
        (sizeof “digiV1OpenRest”) - 1,
        0,
        0,
        0,
        NA_SNMP_PERMANENT
};
```

6.6.3 API Calls

```
naSnmpInsertUserEntry(&digiNoAuthNoEnc);
naSnmpInsertVtfEntry(&theMyMibVTF);
naSnmpInsertVtfEntry(&digiVTF_seeNone);
naSnmpInsertS2GEntry(&s2gFordigiV1);
naSnmpInsertVacmAccessEntry(&accessDigiV1Community);
naSnmpInsertCommunityTableEntry(&digiV1CommunityEntry);
```

7 Conclusion

There is a lot to know in successfully setting up users in SNMPv3. It can be a little daunting but it is not impossible. I hope that this document has help present information about setting up SNMPv3 security in a way that you can apply it to creating NET+OS applications and successfully accessing SNMP agents, in those applications view your SNMP management software.

We do recommend supplemental texts on this subject, for further information. William Stallings' SNMP, SNMPv2, SNMPv3 and RMON I and II is a good solid book describing all versions of SNMP including but not limited to SNMPv3.