Network Working Group   R. Frye
Request for Comments: 2576   CoSine Communications
Category: Standards Track   D. Levi
                                                     Nortel Networks
                                                        S. Routhier
                                            Integrated Systems Inc.
                                                          B. Wijnen
                                                 Lucent Technologies
                                                         March 2000

               Coexistence between Version 1, Version 2, and Version 3
                of the Internet-standard Network Management Framework

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Abstract

   The purpose of this document is to describe coexistence between
   version 3 of the Internet-standard Network Management Framework,
   (SNMPv3), version 2 of the Internet-standard Network Management
   Framework (SNMPv2), and the original Internet-standard Network
   Management Framework (SNMPv1). This document obsoletes RFC 1908 [13]
   and RFC2089 [14].

Table Of Contents

1. Overview

   The purpose of this document is to describe coexistence between
   version 3 of the Internet-standard Network Management Framework,
   termed the SNMP version 3 framework (SNMPv3), version 2 of the
   Internet-standard Network Management Framework, termed the SNMP
   version 2 framework (SNMPv2), and the original Internet-standard
   Network Management Framework (SNMPv1).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in RFC2119 [15].

There are four general aspects of coexistence described in this
document. Each of these is described in a separate section:

   - Conversion of MIB documents between SMIv1 and SMIv2 formats is
     documented in section 2.

   - Mapping of notification parameters is documented in section 3.

   - Approaches to coexistence between entities which support the
     various versions of SNMP in a multi-lingual network is
     documented in section 4. This section addresses the processing
     of protocol operations in multi-lingual implementations, as
     well as behaviour of proxy implementations.

   - The SNMPv1 Message Processing Model and Community-Based
     Security Model, which provides mechanisms for adapting SNMPv1
     into the View-Based Access Control Model (VACM) [20], is
     documented in section 5 (this section also addresses the
     SNMPv2c Message Processing Model and Community-Based Security
     Model).

1.1. SNMPv1

   SNMPv1 is defined by these documents:

   - STD 15, RFC 1157 [2] which defines the Simple Network
     Management Protocol (SNMPv1), the protocol used for network
     access to managed objects.

   - STD 16, RFC 1155 [1] which defines the Structure of Management
     Information (SMIv1), the mechanisms used for describing and
     naming objects for the purpose of management.

   - STD 16, RFC 1212 [3] which defines a more concise description
     mechanism, which is wholly consistent with the SMIv1.

   - RFC 1215 [4] which defines a convention for defining Traps for
     use with the SMIv1.

   Note that throughout this document, the term 'SMIv1' is used. This
   term generally refers to the information presented in RFC 1155, RFC
   1212, and RFC 1215.

1.2. SNMPv2

   SNMPv2 is defined by these documents:

        - STD 58, RFC 2578 which defines Version 2 of the Structure of
           Management Information (SMIv2) [7].

        - STD 58, RFC 2579 which defines common MIB "Textual Conventions"
           [8].

        - STD 58, RFC 2580 which defines Conformance Statements and
           requirements for defining agent and manager capabilities [9].

        - RFC 1905 which defines the Protocol Operations used in
           processing [10].

        - RFC 1906 which defines the Transport Mappings used "on the
           wire" [11].

        - RFC 1907 which defines the basic Management Information Base
           for monitoring and controlling some basic common functions of
           SNMP entities [12].

   Note that SMIv2 as used throughout this document refers to the first
   three documents listed above (RFCs 2578, 2579, and 2580).

   The following document augments the definition of SNMPv2:

        - RFC 1901 [6] is an Experimental definition for using SNMPv2
           PDUs within a community-based message wrapper. This is
           referred to throughout this document as SNMPv2c.

1.3. SNMPv3

   SNMPv3 is defined by these documents:

        - RFC 2571 which defines an Architecture for Describing SNMP
           Management Frameworks [16].

        - RFC 2572 which defines Message Processing and Dispatching [17].

        - RFC 2573 which defines various SNMP Applications [18].

        - RFC 2574 which defines the User-based Security Model (USM),
           providing for both Authenticated and Private (encrypted) SNMP
           messages [19].

        - RFC 2575 which defines the View-based Access Control Model
          (VACM), providing the ability to limit access to different MIB
          objects on a per-user basis [20].

    SNMPv3 also uses the SNMPv2 definitions of RFCs 1905 through 1907 and
    the SMIv2 definitions of 2578 through 2580 described above.

1.4.   SNMPv1 and SNMPv2 Access to MIB Data

    In several places, this document refers to 'SNMPv1 Access to MIB
    Data' and 'SNMPv2 Access to MIB Data'. These terms refer to the part
    of an SNMP agent which actually accesses instances of MIB objects,
    and which actually initiates generation of notifications.
    Differences between the two types of access to MIB data are:

        - Error-status values generated.

        - Generation of exception codes.

        - Use of the Counter64 data type.

        - The format of parameters provided when a notification is
          generated.

    SNMPv1 access to MIB data may generate SNMPv1 error-status values,
    will never generate exception codes nor use the Counter64 data type,
    and will provide SNMPv1 format parameters for generating
    notifications. Note also that SNMPv1 access to MIB data will
    actually never generate a readOnly error (a noSuchName error would
    always occur in the situation where one would expect a readOnly
    error).

    SNMPv2 access to MIB data may generate SNMPv2 error-status values,
    may generate exception codes, may use the Counter64 data type, and
    will provide SNMPv2 format parameters for generating notifications.
    Note that SNMPv2 access to MIB data will never generate readOnly,
    noSuchName, or badValue errors.

    Note that a particular multi-lingual implementation may choose to
    implement all access to MIB data as SNMPv2 access to MIB data, and
    perform the translations described herein for SNMPv1-based
    transactions.

2. SMI and Management Information Mappings

    The SMIv2 approach towards describing collections of managed objects
    is nearly a proper superset of the approach defined in the SMIv1.
    For example, both approaches use an adapted subset of ASN.1 (1988)

[11] as the basis for a formal descriptive notation. Indeed, one
might note that the SMIv2 approach largely codifies the existing
practice for defining MIB modules, based on extensive experience with
the SMIv1.

The following sections consider the three areas: MIB modules,
compliance statements, and capabilities statements.

2.1. MIB Modules

MIB modules defined using the SMIv1 may continue to be used with
protocol versions which use SNMPv2 PDUs. However, for the MIB
modules to conform to the SMIv2, the following changes SHALL be made:

2.1.1. Object Definitions

In general, conversion of a MIB module does not require the
deprecation of the objects contained therein. If the definition of
an object is truly inadequate for its intended purpose, the object
SHALL be deprecated or obsoleted, otherwise deprecation is not
required.

(1)   The IMPORTS statement MUST reference SNMPv2-SMI, instead of
      RFC1155-SMI and RFC-1212.

(2)   The MODULE-IDENTITY macro MUST be invoked immediately after any
      IMPORTs statement.

(3)   For any object with an integer-valued SYNTAX clause, in which
      the corresponding INTEGER does not have a range restriction
      (i.e., the INTEGER has neither a defined set of named-number
      enumerations nor an assignment of lower- and upper-bounds on its
      value), the object MUST have the value of its SYNTAX clause
      changed to Integer32, or have an appropriate range specified.

(4)   For any object with a SYNTAX clause value of Counter, the object
      MUST have the value of its SYNTAX clause changed to Counter32.

(5)   For any object with a SYNTAX clause value of Gauge, the object
      MUST have the value of its SYNTAX clause changed to Gauge32, or
      Unsigned32 where appropriate.

(6)   For all objects, the ACCESS clause MUST be replaced by a MAX-
      ACCESS clause. The value of the MAX-ACCESS clause SHALL be the
      same as that of the ACCESS clause unless some other value makes
      "protocol sense" as the maximal level of access for the object.
      In particular, object types for which instances can be
      explicitly created by a protocol set operation, SHALL have a

MAX-ACCESS clause of "read-create". If the value of the ACCESS
clause is "write-only", then the value of the MAX-ACCESS clause
MUST be "read-write", and the DESCRIPTION clause SHALL note that
reading this object will result in implementation-specific
results. Note that in SMIv1, the ACCESS clause specifies the
minimal required access, while in SMIv2, the MAX-ACCESS clause
specifies the maximum allowed access. This should be considered
when converting an ACCESS clause to a MAX-ACCESS clause.

(7) For all objects, if the value of the STATUS clause is
     "mandatory" or "optional", the value MUST be replaced with
     "current", "deprecated", or "obsolete" depending on the current
     usage of such objects.

(8) For any object not containing a DESCRIPTION clause, the object
     MUST have a DESCRIPTION clause defined.

(9) For any object corresponding to a conceptual row which does not
     have an INDEX clause, the object MUST have either an INDEX
     clause or an AUGMENTS clause defined.

(10) If any INDEX clause contains a reference to an object with a
     syntax of NetworkAddress, then a new object MUST be created and
     placed in this INDEX clause immediately preceding the object
     whose syntax is NetworkAddress. This new object MUST have a
     syntax of INTEGER, it MUST be not-accessible, and its value MUST
     always be 1. This approach allows one to convert a MIB module
     in SMIv1 format to one in SMIv2 format, and then use it with the
     SNMPv1 protocol with no impact to existing SNMPv1 agents and
     managers.

(11) For any object with a SYNTAX of NetworkAddress, the SYNTAX MUST
     be changed to IpAddress. Note that the use of NetworkAddress in
     new MIB documents is strongly discouraged (in fact, new MIB
     documents should be written using SMIv2, which does not define
     NetworkAddress).

(12) For any object containing a DEFVAL clause with an OBJECT
     IDENTIFIER value which is expressed as a collection of sub-
     identifiers, the value MUST be changed to reference a single
     ASN.1 identifier. This may require defining a series of new
     administrative assignments (OBJECT IDENTIFIERS) in order to
     define the single ASN.1 identifier.

(13) One or more OBJECT-GROUPS MUST be defined, and related objects
     SHOULD be collected into appropriate groups. Note that SMIv2
     requires all OBJECT-TYPEs to be a member of at least one
     OBJECT-GROUP.

Other changes are desirable, but not necessary:

(1)   Creation and deletion of conceptual rows is inconsistent using
      the SMIv1. The SMIv2 corrects this. As such, if the MIB module
      undergoes review early in its lifetime, and it contains
      conceptual tables which allow creation and deletion of
      conceptual rows, then the objects relating to those tables MAY
      be deprecated and replaced with objects defined using the new
      approach. The approach based on SMIv2 can be found in section 7
      of RFC2578 [7], and the RowStatus and StorageType TEXTUAL-
      CONVENTIONs are described in section 2 of RFC2579 [8].

(2)   For any object with a string-valued SYNTAX clause, in which the
      corresponding OCTET STRING does not have a size restriction
      (i.e., the OCTET STRING has no assignment of lower- and upper-
      bounds on its length), the bounds for the size of the object
      SHOULD be defined.

(3)   All textual conventions informally defined in the MIB module
      SHOULD be redefined using the TEXTUAL-CONVENTION macro. Such a
      change would not necessitate deprecating objects previously
      defined using an informal textual convention.

(4)   For any object which represents a measurement in some kind of
      units, a UNITS clause SHOULD be added to the definition of that
      object.

(5)   For any conceptual row which is an extension of another
      conceptual row, i.e., for which subordinate columnar objects
      both exist and are identified via the same semantics as the
      other conceptual row, an AUGMENTS clause SHOULD be used in place
      of the INDEX clause for the object corresponding to the
      conceptual row which is an extension.

Finally, to avoid common errors in SMIv1 MIB modules:

(1)   For any non-columnar object that is instanced as if it were
      immediately subordinate to a conceptual row, the value of the
      STATUS clause of that object MUST be changed to "obsolete".

(2)   For any conceptual row object that is not contained immediately
      subordinate to a conceptual table, the value of the STATUS
      clause of that object (and all subordinate objects) MUST be
      changed to "obsolete".

2.1.2. Trap and Notification Definitions

   If a MIB module is changed to conform to the SMIv2, then each
   occurrence of the TRAP-TYPE macro MUST be changed to a corresponding
   invocation of the NOTIFICATION-TYPE macro:

   (1)   The IMPORTS statement MUST NOT reference RFC-1215 [4], and MUST
         reference SNMPv2-SMI instead.

   (2)   The ENTERPRISE clause MUST be removed.

   (3)   The VARIABLES clause MUST be renamed to the OBJECTS clause.

   (4)   A STATUS clause MUST be added, with an appropriate value.
         Normally the value should be 'current,' although 'deprecated' or
         'obsolete' may be used as needed.

   (5)   The value of an invocation of the NOTIFICATION-TYPE macro is an
         OBJECT IDENTIFIER, not an INTEGER, and MUST be changed
         accordingly. Specifically, if the value of the ENTERPRISE
         clause is not 'snmp' then the value of the invocation SHALL be
         the value of the ENTERPRISE clause extended with two sub-
         identifiers, the first of which has the value 0, and the second
         has the value of the invocation of the TRAP-TYPE. If the value
         of the ENTERPRISE clause is 'snmp', then the value of the
         invocation of the NOTIFICATION-TYPE macro SHALL be mapped in the
         same manner as described in section 3.1 in this document.

   (6)   A DESCRIPTION clause MUST be added, if not already present.

   (7)   One or more NOTIFICATION-GROUPs MUST be defined, and related
         notifications MUST be collected into those groups. Note that
         SMIv2 requires that all NOTIFICATION-TYPEs be a member of at
         least one NOTIFICATION-GROUP.

2.2. Compliance Statements

   For those information modules which are "standards track", a
   corresponding invocation of the MODULE-COMPLIANCE macro and related
   OBJECT-GROUP and/or NOTIFICATION-GROUP macros MUST be included within
   the information module (or in a companion information module), and
   any commentary text in the information module which relates to
   compliance SHOULD be removed. Typically this editing can occur when
   the information module undergoes review.

Note that a MODULE-COMPLIANCE statement is not required for a MIB
document that is not on the standards track (for example, an
enterprise MIB), though it may be useful in some circumstances to
define a MODULE-COMPLIANCE statement for such a MIB document.

2.3. Capabilities Statements

RFC1303 [5] uses the MODULE-CONFORMANCE macro to describe an agent's
capabilities with respect to one or more MIB modules. Converting
such a description for use with the SMIv2 requires these changes:

(1) The macro name AGENT-CAPABILITIES SHOULD be used instead of
     MODULE-CONFORMANCE.

(2) The STATUS clause SHOULD be added, with a value of 'current'.

(3) All occurrences of the CREATION-REQUIRES clause MUST either be
     omitted if appropriate, or be changed such that the semantics
     are consistent with RFC2580 [9].

In order to ease coexistence, object groups defined in an SMIv1
compliant MIB module may be referenced by the INCLUDES clause of an
invocation of the AGENT-CAPABILITIES macro: upon encountering a
reference to an OBJECT IDENTIFIER subtree defined in an SMIv1 MIB
module, all leaf objects which are subordinate to the subtree and
have a STATUS clause value of mandatory are deemed to be INCLUDED.
(Note that this method is ambiguous when different revisions of an
SMIv1 MIB have different sets of mandatory objects under the same
subtree; in such cases, the only solution is to rewrite the MIB using
the SMIv2 in order to define the object groups unambiguously.)

3. Translating Notifications Parameters

This section describes how parameters used for generating
notifications are translated between the format used for SNMPv1
notification protocol operations and the format used for SNMPv2
notification protocol operations. The parameters used to generate a
notification are called 'notification parameters'. The format of
parameters used for SNMPv1 notification protocol operations is
refered to in this document as 'SNMPv1 notification parameters'. The
format of parameters used for SNMPv2 notification protocol operations
is refered to in this document as 'SNMPv2 notification parameters'.

The situations where notification parameters MUST be translated are:

   - When an entity generates a set of notification parameters in a
      particular format, and the configuration of the entity
      indicates that the notification must be sent using an SNMP
      message version that requires the other format for notification
      parameters.

   - When a proxy receives a notification that was sent using an
      SNMP message version that requires one format of notification
      parameters, and must forward the notification using an SNMP
      message version that requires the other format of notification
      parameters.

In addition, it MAY be desirable to translate notification parameters
in a notification receiver application in order to present
notifications to the end user in a consistent format.

Note that for the purposes of this section, the set of notification
parameters is independent of whether the notification is to be sent
as a trap or an inform.

SNMPv1 notification parameters consist of:

   - An enterprise parameter (OBJECT IDENTIFIER).

   - An agent-addr parameter (NetworkAddress).

   - A generic-trap parameter (INTEGER).

   - A specific-trap parameter (INTEGER).

   - A time-stamp parameter (TimeTicks).

   - A list of variable-bindings (VarBindList).

SNMPv2 notification parameters consist of:

   - A sysUpTime parameter (TimeTicks). This appears in the first
      variable-binding in an SNMPv2-Trap-PDU or InformRequest-PDU.

   - An snmpTrapOID parameter (OBJECT IDENTIFIER). This appears in
      the second variable-binding in an SNMPv2-Trap-PDU or
      InformRequest-PDU.

   - A list of variable-bindings (VarBindList). This refers to all
      but the first two variable-bindings in an SNMPv2-Trap-PDU or
      InformRequest-PDU.

3.1. Translating SNMPv1 Notification Parameters to SNMPv2 Notification
      Parameters

   The following procedure describes how to translate SNMPv1
   notification parameters into SNMPv2 notification parameters:

   (1) The SNMPv2 sysUpTime parameter SHALL be taken directly from the
        SNMPv1 time-stamp parameter.

   (2) If the SNMPv1 generic-trap parameter is 'enterpriseSpecific(6)',
        the SNMPv2 snmpTrapOID parameter SHALL be the concatenation of
        the SNMPv1 enterprise parameter and two additional sub-
        identifiers, '0', and the SNMPv1 specific-trap parameter.

   (3) If the SNMPv1 generic-trap parameter is not '
        enterpriseSpecific(6)', the SNMPv2 snmpTrapOID parameter SHALL
        be the corresponding trap as defined in section 2 of RFC1907
        [12]:

        generic-trap parameter   snmpTrapOID.0
        ======================   =============
        0   1.3.6.1.6.3.1.1.5.1 (coldStart)
        1   1.3.6.1.6.3.1.1.5.2 (warmStart)
        2   1.3.6.1.6.3.1.1.5.3 (linkDown)
        3   1.3.6.1.6.3.1.1.5.4 (linkUp)
        4   1.3.6.1.6.3.1.1.5.5 (authenticationFailure)
        5   1.3.6.1.6.3.1.1.5.6 (egpNeighborLoss)

   (4) The SNMPv2 variable-bindings SHALL be the SNMPv1 variable-
        bindings. In addition, if the translation is being performed by
        a proxy in order to forward a received trap, three additional
        variable-bindings will be appended, if these three additional
        variable-bindings do not already exist in the SNMPv1 variable-
        bindings. The name portion of the first additional variable
        binding SHALL contain snmpTrapAddress.0, and the value SHALL
        contain the SNMPv1 agent-addr parameter. The name portion of
        the second additional variable binding SHALL contain
        snmpTrapCommunity.0, and the value SHALL contain the value of
        the community-string field from the received SNMPv1 message
        which contained the SNMPv1 Trap-PDU. The name portion of the
        third additional variable binding SHALL contain
        snmpTrapEnterprise.0 [12], and the value SHALL be the SNMPv1
        enterprise parameter.

3.2. Translating SNMPv2 Notification Parameters to SNMPv1 Notification
      Parameters

   The following procedure describes how to translate SNMPv2
   notification parameters into SNMPv1 notification parameters:

   (1) The SNMPv1 enterprise parameter SHALL be determined as follows:

      - If the SNMPv2 snmpTrapOID parameter is one of the standard
         traps as defined in RFC1907 [12], then the SNMPv1 enterprise
         parameter SHALL be set to the value of the variable-binding in
         the SNMPv2 variable-bindings whose name is snmpTrapEnterprise.0
         if that variable-binding exists. If it does not exist, the
         SNMPv1 enterprise parameter SHALL be set to the value '
         snmpTraps' as defined in RFC1907 [12].

      - If the SNMPv2 snmpTrapOID parameter is not one of the standard
         traps as defined in RFC1907 [12], then the SNMPv1 enterprise
         parameter SHALL be determined from the SNMPv2 snmpTrapOID
         parameter as follows:

         - If the next-to-last sub-identifier of the snmpTrapOID is
            zero, then the SNMPv1 enterprise SHALL be the SNMPv2
            snmpTrapOID with the last 2 sub-identifiers removed,
            otherwise

         - If the next-to-last sub-identifier of the snmpTrapOID is
            non-zero, then the SNMPv1 enterprise SHALL be the SNMPv2
            snmpTrapOID with the last sub-identifier removed.

   (2) The SNMPv1 agent-addr parameter SHALL be determined based on the
        situation in which the translation occurs.

      - If the translation occurs within a notification originator
         application, and the notification is to be sent over IP, the
         SNMPv1 agent-addr parameter SHALL be set to the IP address of
         the SNMP entity in which the notification originator resides.
         If the notification is to be sent over some other transport,
         the SNMPv1 agent-addr parameter SHALL be set to 0.0.0.0.

      - If the translation occurs within a proxy application, the proxy
         must attempt to extract the original source of the notification
         from the variable-bindings. If the SNMPv2 variable-bindings
         contains a variable binding whose name is snmpTrapAddress.0,
         the agent-addr parameter SHALL be set to the value of that
         variable binding. Otherwise, the SNMPv1 agent-addr parameter
         SHALL be set to 0.0.0.0.

(3) If the SNMPv2 snmpTrapOID parameter is one of the standard traps
     as defined in RFC1907 [12], the SNMPv1 generic-trap parameter
     SHALL be set as follows:

```
         snmpTrapOID.0 parameter                  generic-trap
         ===============================          ============
         1.3.6.1.6.3.1.1.5.1 (coldStart)                0
         1.3.6.1.6.3.1.1.5.2 (warmStart)                1
         1.3.6.1.6.3.1.1.5.3 (linkDown)                 2
         1.3.6.1.6.3.1.1.5.4 (linkUp)                   3
         1.3.6.1.6.3.1.1.5.5 (authenticationFailure)    4
         1.3.6.1.6.3.1.1.5.6 (egpNeighborLoss)          5
```

     Otherwise, the SNMPv1 generic-trap parameter SHALL be set to 6.

(4) If the SNMPv2 snmpTrapOID parameter is one of the standard traps
     as defined in RFC1907 [12], the SNMPv1 specific-trap parameter
     SHALL be set to zero. Otherwise, the SNMPv1 specific-trap
     parameter SHALL be set to the last sub-identifier of the SNMPv2
     snmpTrapOID parameter.

(5) The SNMPv1 time-stamp parameter SHALL be taken directly from the
     SNMPv2 sysUpTime parameter.

(6) The SNMPv1 variable-bindings SHALL be the SNMPv2 variable-
     bindings. Note, however, that if the SNMPv2 variable-bindings
     contain any objects whose type is Counter64, the translation to
     SNMPv1 notification parameters cannot be performed. In this
     case, the notification cannot be encoded in an SNMPv1 packet
     (and so the notification cannot be sent using SNMPv1, see
     section 4.1.3 and section 4.2).

4. Approaches to Coexistence in a Multi-lingual Network

   There are two basic approaches to coexistence in a multi-lingual
   network, multi-lingual implementations and proxy implementations.
   Multi-lingual implementations allow elements in a network to
   communicate with each other using an SNMP version which both elements
   support. This allows a multi-lingual implementation to communicate
   with any mono-lingual implementation, regardless of the SNMP version
   supported by the mono-lingual implementation.

   Proxy implementations provide a mechanism for translating between
   SNMP versions using a third party network element. This allows
   network elements which support only a single, but different, SNMP
   version to communicate with each other. Proxy implementations are
   also useful for securing communications over an insecure link between
   two locally secure networks.

4.1. Multi-lingual implementations

   This approach requires an entity to support multiple SNMP message
   versions. Typically this means supporting SNMPv1, SNMPv2c, and
   SNMPv3 message versions. The behaviour of various types of SNMP
   applications which support multiple message versions is described in
   the following sections. This approach allows entities which support
   multiple SNMP message versions to coexist with and communicate with
   entities which support only a single SNMP message version.

4.1.1. Command Generator

   A command generator must select an appropriate message version when
   sending requests to another entity. One way to achieve this is to
   consult a local database to select the appropriate message version.

   In addition, a command generator MUST 'downgrade' GetBulk requests to
   GetNext requests when selecting SNMPv1 as the message version for an
   outgoing request. This is done by simply changing the operation type
   to GetNext, ignoring any non-repeaters and max-repetitions values,
   and setting error-status and error-index to zero.

4.1.2. Command Responder

   A command responder must be able to deal with both SNMPv1 and SNMPv2
   access to MIB data. There are three aspects to dealing with this. A
   command responder must:

      - Deal correctly with SNMPv2 access to MIB data that returns a
         Counter64 value while processing an SNMPv1 message,

      - Deal correctly with SNMPv2 access to MIB data that returns one
         of the three exception values while processing an SNMPv1
         message, and

      - Map SNMPv2 error codes returned from SNMPv2 access to MIB data
         into SNMPv1 error codes when processing an SNMPv1 message.

   Note that SNMPv1 error codes SHOULD NOT be used without any change
   when processing SNMPv2c or SNMPv3 messages, except in the case of
   proxy forwarding. In the case of proxy forwarding, for backwards
   compatibility, SNMPv1 error codes may be used without any change in a
   forwarded SNMPv2c or SNMPv3 message.

   The following sections describe the behaviour of a command responder
   application which supports multiple SNMP message versions, and which
   uses some combination of SNMPv1 and SNMPv2 access to MIB data.

4.1.2.1. Handling Counter64

   The SMIv2 [7] defines one new syntax that is incompatible with SMIv1.
   This syntax is Counter64. All other syntaxes defined by SMIv2 are
   compatible with SMIv1.

   The impact on multi-lingual command responders is that they MUST NOT
   ever return a variable binding containing a Counter64 value in a
   response to a request that was received using the SNMPv1 message
   version.

   Multi-lingual command responders SHALL take the approach that object
   instances whose type is Counter64 are implicitly excluded from view
   when processing an SNMPv1 message. So:

      - On receipt of an SNMPv1 GetRequest-PDU containing a variable
         binding whose name field points to an object instance of type
         Counter64, a GetResponsePDU SHALL be returned, with an error-
         status of noSuchName and the error-index set to the variable
         binding that caused this error.

      - On an SNMPv1 GetNextRequest-PDU, any object instance which
         contains a syntax of Counter64 SHALL be skipped, and the next
         accessible object instance that does not have the syntax of
         Counter64 SHALL be retrieved. If no such object instance
         exists, then an error-status of noSuchName SHALL be returned,
         and the error-index SHALL be set to the variable binding that
         caused this error.

      - Any SNMPv1 request which contains a variable binding with a
         Counter64 value is ill-formed, so the foregoing rules do not
         apply. If that error is detected, a response SHALL NOT be
         returned, since it would contain a copy of the ill-formed
         variable binding. Instead, the offending PDU SHALL be
         discarded and the counter snmpInASNParseErrs SHALL be
         incremented.

4.1.2.2. Mapping SNMPv2 Exceptions

   SNMPv2 provides a feature called exceptions, which allow an SNMPv2
   Response PDU to return as much management information as possible,
   even when an error occurs. However, SNMPv1 does not support
   exceptions, and so an SNMPv1 Response PDU cannot return any
   management information, and can only return an error-status and
   error-index value.

When an SNMPv1 request is received, a command responder MUST check
any variable bindings returned using SNMPv2 access to MIB data for
exception values, and convert these exception values into SNMPv1
error codes.

The type of exception that can be returned when accessing MIB data
and the action taken depends on the type of SNMP request.

   - For a GetRequest, a noSuchObject or noSuchInstance exception
      may be returned.

   - For a GetNextRequest, an endOfMibView exception may be
      returned.

   - No exceptions will be returned for a SetRequest, and a
      GetBulkRequest should only be received in an SNMPv2c or SNMPv3
      message, so these request types may be ignored when mapping
      exceptions.

Note that when a response contains multiple exceptions, it is an
implementation choice as to which variable binding the error-index
should reference.

4.1.2.2.1. Mapping noSuchObject and noSuchInstance

A noSuchObject or noSuchInstance exception generated by an SNMPv2
access to MIB data indicates that the requested object instance can
not be returned. The SNMPv1 error code for this condition is
noSuchName, and so the error-status field of the response PDU SHALL
be set to noSuchName. Also, the error-index field SHALL be set to
the index of the variable binding for which an exception occurred
(there may be more than one and it is an implementation decision as
to which is used), and the variable binding list from the original
request SHALL be returned with the response PDU.

4.1.2.2.2. Mapping endOfMibView

When an SNMPv2 access to MIB data returns a variable binding
containing an endOfMibView exception, it indicates that there are no
object instances available which lexicographically follow the object
in the request. In an SNMPv1 agent, this condition normally results
in a noSuchName error, and so the error-status field of the response
PDU SHALL be set to noSuchName. Also, the error-index field SHALL be
set to the index of the variable binding for which an exception
occurred (there may be more than one and it is an implementation
decision as to which is used), and the variable binding list from the
original request SHALL be returned with the response PDU.

4.1.2.3. Processing An SNMPv1 GetRequest

   When processing an SNMPv1 GetRequest, the following procedures MUST
   be followed when using an SNMPv2 access to MIB data.

   When such an access to MIB data returns response data using SNMPv2
   syntax and error-status values, then:

   (1) If the error-status is anything other than noError,

      - The error status SHALL be translated to an SNMPv1 error-status
         using the table in section 4.3, "Error Status Mappings".

      - The error-index SHALL be set to the position (in the original
         request) of the variable binding that caused the error-status.

      - The variable binding list of the response PDU SHALL be made
         exactly the same as the variable binding list that was received
         in the original request.

   (2) If the error-status is noError, the variable bindings SHALL be
        checked for any SNMPv2 exception (noSuchObject or
        noSuchInstance) or an SNMPv2 syntax that is unknown to SNMPv1
        (Counter64). If there are any such variable bindings, one of
        those variable bindings SHALL be selected (it is an
        implementation choice as to which is selected), and:

      - The error-status SHALL be set to noSuchName,

      - The error-index SHALL be set to the position (in the variable
         binding list of the original request) of the selected variable
         binding, and

      - The variable binding list of the response PDU SHALL be exactly
         the same as the variable binding list that was received in the
         original request.

   (3) If there are no such variable bindings, then:

      - The error-status SHALL be set to noError,

      - The error-index SHALL be set to zero, and

      - The variable binding list of the response SHALL be composed
         from the data as it is returned by the access to MIB data.

4.1.2.4. Processing An SNMPv1 GetNextRequest

   When processing an SNMPv1 GetNextRequest, the following procedures
   MUST be followed when an SNMPv2 access to MIB data is called as part
   of processing the request. There may be repetitive accesses to MIB
   data to try to find the first object which lexicographically follows
   each of the objects in the request. This is implementation specific.
   These procedures are followed only for data returned when using
   SNMPv2 access to MIB data. Data returned using SNMPv1 access to MIB
   data may be treated in the normal manner for an SNMPv1 request.

   First, if the access to MIB data returns an error-status of anything
   other than noError:

   (1)   The error status SHALL be translated to an SNMPv1 error-status
         using the table in section 4.3, "Error Status Mappings".

   (2)   The error-index SHALL be set to the position (in the original
         request) of the variable binding that caused the error-status.

   (3)   The variable binding list of the response PDU SHALL be exactly
         the same as the variable binding list that was received in the
         original request.

   Otherwise, if the access to MIB data returns an error-status of
   noError:

   (1)   Any variable bindings containing an SNMPv2 syntax of Counter64
         SHALL be considered to be not in view, and MIB data SHALL be
         accessed as many times as is required until either a value other
         than Counter64 is returned, or an error occurs.

   (2)   If there is any variable binding that contains an SNMPv2
         exception endOfMibView (there may be more than one, it is an
         implementation decision as to which is chosen):

      -   The error-status SHALL be set to noSuchName,

      -   The error-index SHALL be set to the position (in the variable
          binding list of the original request) of the variable binding
          that returned such an SNMPv2 exception, and

      -   The variable binding list of the response PDU SHALL be exactly
          the same as the variable binding list that was received in the
          original request.

   (3)   If there are no such variable bindings, then:

        - The error-status SHALL be set to noError,

        - The error-index SHALL be set to zero, and

        - The variable binding list of the response SHALL be composed
           from the data as it is returned by the access to MIB data.

4.1.2.5. Processing An SNMPv1 SetRequest

   When processing an SNMPv1 SetRequest, the following procedures MUST
   be followed when calling SNMPv2 MIB access routines.

   When such MIB access routines return response data using SNMPv2
   syntax and error-status values, and the error-status is anything
   other than noError, then:

        - The error status SHALL be translated to an SNMPv1 error-status
           using the table in section 4.3, "Error Status Mappings".

        - The error-index SHALL be set to the position (in the original
           request) of the variable binding that caused the error-status.

        - The variable binding list of the response PDU SHALL be made
           exactly the same as the variable binding list that was received
           in the original request.

4.1.3. Notification Originator

   A notification originator must be able to translate between SNMPv1
   notifications parameters and SNMPv2 notification parameters in order
   to send a notification using a particular SNMP message version. If a
   notification is generated using SNMPv1 notification parameters, and
   configuration information specifies that notifications be sent using
   SNMPv2c or SNMPv3, the notification parameters must be translated to
   SNMPv2 notification parameters. Likewise, if a notification is
   generated using SNMPv2 notification parameters, and configuration
   information specifies that notifications be sent using SNMPv1, the
   notification parameters must be translated to SNMPv1 notification
   parameters. In this case, if the notification cannot be translated
   (due to the presence of a Counter64 type), it will not be sent using
   SNMPv1.

   When a notification originator generates a notification, using
   parameters obtained from the SNMP-TARGET-MIB and SNMP-NOTIFICATION-
   MIB, if the SNMP version used to generate the notification is SNMPv1,
   the PDU type used will always be a TrapPDU, regardless of whether the
   value of snmpNotifyType is trap(1) or inform(2).

Note also that access control and notification filtering are
performed in the usual manner for notifications, regardless of the
SNMP message version to be used when sending a notification. The
parameters for performing access control are found in the usual
manner (i.e., from inspecting the SNMP-TARGET-MIB and SNMP-
NOTIFICATION-MIB). In particular, when generating an SNMPv1 Trap, in
order to perform the access check specified in [18], section 3.3,
bullet (3), the notification originator may need to generate a value
for snmpTrapOID.0 as described in section 3.1, bullets (2) and (3) of
this document. If the SNMPv1 notification parameters being used were
previously translated from a set of SNMPv2 notification parameters,
this value may already be known, in which case it need not be
generated.

4.1.4. Notification Receiver

There are no special requirements of a notification receiver.
However, an implementation may find it useful to allow a higher level
application to request whether notifications should be delivered to a
higher level application using SNMPv1 notification parameter or
SNMPv2 notification parameters. The notification receiver would then
translate notification parameters when required in order to present a
notification using the desired set of parameters.

4.2. Proxy Implementations

A proxy implementation may be used to enable communication between
entities which support different SNMP message versions. This is
accomplished in a proxy forwarder application by performing
translations on PDUs. These translations depend on the PDU type, the
SNMP version of the packet containing a received PDU, and the SNMP
version to be used to forward a received PDU. The following sections
describe these translations. In all cases other than those described
below, the proxy SHALL forward a received PDU without change, subject
to size constraints as defined in section 5.3 (Community MIB) of this
document. Note that in the following sections, the 'Upstream
Version' refers to the version used between the command generator and
the proxy, and the 'Downstream Version' refers to the version used
between the proxy and the command responder, regardless of the PDU
type or direction.

4.2.1. Upstream Version Greater Than Downstream Version

   - If a GetBulkRequest-PDU is received and must be forwarded using
      the SNMPv1 message version, the proxy forwarder SHALL set the
      non-repeaters and max-repetitions fields to 0, and SHALL set the
      tag of the PDU to GetNextRequest-PDU.

- If a GetResponse-PDU is received whose error-status field has a
  value of 'tooBig', the message will be forwarded using the SNMPv2c
  or SNMPv3 message version, and the original request received by
  the proxy was not a GetBulkRequest-PDU, the proxy forwarder SHALL
  remove the contents of the variable-bindings field before
  forwarding the response.

- If a GetResponse-PDU is received whose error-status field has a
  value of 'tooBig,' and the message will be forwarded using the
  SNMPv2c or SNMPv3 message version, and the original request
  received by the proxy was a GetBulkRequest-PDU, the proxy
  forwarder SHALL re-send the forwarded request (which would have
  been altered to be a GetNextRequest-PDU) with all but the first
  variable-binding removed. The proxy forwarder SHALL only re-send
  such a request a single time. If the resulting GetResponse-PDU
  also contains an error-status field with a value of 'tooBig,' then
  the proxy forwarder SHALL remove the contents of the variable-
  bindings field, and change the error-status field to 'noError'
  before forwarding the response. Note that if the original request
  only contained a single variable-binding, the proxy may skip re-
  sending the request and simply remove the variable-bindings and
  change the error-status to 'noError.'

- If a Trap-PDU is received, and will be forwarded using the SNMPv2c
  or SNMPv3 message version, the proxy SHALL apply the translation
  rules described in section 3, and SHALL forward the notification
  as an SNMPv2-Trap-PDU.

  Note that when an SNMPv1 agent generates a message containing a
  Trap-PDU which is subsequently forwarded by one or more proxy
  forwarders using SNMP versions other than SNMPv1, the community
  string and agent-addr fields from the original message generated
  by the SNMPv1 agent will be preserved through the use of the
  snmpTrapAddress and snmpTrapCommunity nobjects.

4.2.2. Upstream Version Less Than Downstream Version

- If a GetResponse-PDU is received in response to a GetRequest-PDU
  (previously generated by the proxy) which contains variable-
  bindings of type Counter64 or which contain an SNMPv2 exception
  code, and the message would be forwarded using the SNMPv1 message
  version, the proxy MUST generate an alternate response PDU
  consisting of the request-id and variable bindings from the
  original SNMPv1 request, containing a noSuchName error-status
  value, and containing an error-index value indicating the position
  of the variable-binding containing the Counter64 type or exception
  code.

- If a GetResponse-PDU is received in response to a GetNextRequest-
  PDU (previously generated by the proxy) which contains variable-
  bindings that contain an SNMPv2 exception code, and the message
  would be forwarded using the SNMPv1 message version, the proxy
  MUST generate an alternate response PDU consisting of the
  request-id and variable bindings from the original SNMPv1 request,
  containing a noSuchName error-status value, and containing an
  error-index value indicating the position of the variable-binding
  containing the exception code.

- If a GetResponse-PDU is received in response to a GetNextRequest-
  PDU (previously generated by the proxy) which contains variable-
  bindings of type Counter64, the proxy MUST re-send the entire
  GetNextRequest-PDU, with the following modifications. For any
  variable bindings in the received GetResponse which contained
  Counter64 types, the proxy substitutes the object names of these
  variable bindings for the corresponding object names in the
  previously-sent GetNextRequest. The proxy MUST repeat this
  process until no Counter64 objects are returned. Note that an
  implementation may attempt to optimize this process of skipping
  Counter64 objects. One approach to such an optimization would be
  to replace the last sub-identifier of the object names of varbinds
  containing a Counter64 type with 65535 if that sub-identifier is
  less than 65535, or with 4294967295 if that sub-identifier is
  greater than 65535. This approach should skip multiple instances
  of the same Counter64 object, while maintaining compatibility with
  some broken agent implementations (which only use 16-bit integers
  for sub-identifiers).

  Deployment Hint: The process of repeated GetNext requests used by
  a proxy when Counter64 types are returned can be expensive. When
  deploying a proxy, this can be avoided by configuring the target
  agents to which the proxy forwards requests in a manner such that
  any objects of type Counter64 are in fact not-in-view for the
  principal that the proxy is using when communicating with these
  agents.

- If a GetResponse-PDU is received which contains an SNMPv2 error-
  status value of wrongValue, wrongEncoding, wrongType, wrongLength,
  inconsistentValue, noAccess, notWritable, noCreation,
  inconsistentName, resourceUnavailable, commitFailed, undoFailed,
  or authorizationError, the error-status value is modified using
  the mappings in section 4.3.

- If an SNMPv2-Trap-PDU is received, and will be forwarded using the
  SNMPv1 message version, the proxy SHALL apply the translation
  rules described in section 3, and SHALL forward the notification

as a Trap-PDU. Note that if the translation fails due to the
existence of a Counter64 data-type in the received SNMPv2-Trap-
PDU, the trap cannot be forwarded using SNMPv1.

- If an InformRequest-PDU is received, any configuration information
   indicating that it would be forwarded using the SNMPv1 message
   version SHALL be ignored. An InformRequest-PDU can only be
   forwarded using the SNMPv2c or SNMPv3 message version. The
   InformRequest-PDU may still be forwarded if there is other
   configuration information indicating that it should be forwarded
   using SNMPv2c or SNMPv3.

4.3. Error Status Mappings

   The following tables shows the mappings of SNMPv1 error-status values
   into SNMPv2 error-status values, and the mappings of SNMPv2 error-
   status values into SNMPv1 error-status values.

```
            SNMPv1 error-status    SNMPv2 error-status
            ===================    ===================
            noError                noError
            tooBig                 tooBig
            noSuchName             noSuchName
            badValue               badValue
            genErr                 genErr


            SNMPv2 error-status    SNMPv1 error-status
            ===================    ===================
            noError                noError
            tooBig                 tooBig
            genErr                 genErr
            wrongValue             badValue
            wrongEncoding          badValue
            wrongType              badValue
            wrongLength            badValue
            inconsistentValue      badValue
            noAccess               noSuchName
            notWritable            noSuchName
            noCreation             noSuchName
            inconsistentName       noSuchName
            resourceUnavailable    genErr
            commitFailed           genErr
            undoFailed             genErr
            authorizationError     noSuchName
```

Whenever the SNMPv2 error-status value of authorizationError is
translated to an SNMPv1 error-status value of noSuchName, the value
of snmpInBadCommunityUses MUST be incremented.

5. Message Processing Models and Security Models

In order to adapt SNMPv1 (and SNMPv2c) into the SNMP architecture,
the following models are defined in this document:

   - The SNMPv1 Message Processing Model

   - The SNMPv1 Community-Based Security Model

The following models are also described in this document:

   - The SNMPv2c Message Processing Model

   - The SNMPv2c Community-Based Security Model

      In most respects, the SNMPv1 Message Processing Model and the
      SNMPv2c Message Processing Model are identical, and so these
      are not discussed independently in this document. Differences
      between the two models are described as required.

      Similarly, the SNMPv1 Community-Based Security Model and the
      SNMPv2c Community-Based Security Model are nearly identical,
      and so are not discussed independently. Differences between
      these two models are also described as required.

5.1. Mappings

The SNMPv1 (and SNMPv2c) Message Processing Model and Security Model
require mappings between parameters used in SNMPv1 (and SNMPv2c)
messages, and the version independent parameters used in the SNMP
architecture [16]. The parameters which MUST be mapped consist of
the SNMPv1 (and SNMPv2c) community name, and the SNMP securityName
and contextEngineID/contextName pair. A MIB module (the SNMP-
COMMUNITY-MIB) is provided in this document in order to perform these
mappings. This MIB provides mappings in both directions, that is, a
community name may be mapped to a securityName, contextEngineID, and
contextName, or the combination of securityName, contextEngineID, and
contextName may be mapped to a community name.

5.2. The SNMPv1 MP Model and SNMPv1 Community-based Security Model

   The SNMPv1 Message Processing Model handles processing of SNMPv1
   messages. The processing of messages is handled generally in the
   same manner as described in RFC1157 [2], with differences and
   clarifications as described in the following sections. The
   SnmpMessageProcessingModel value for SNMPv1 is 0 (the value for
   SNMPv2c is 1).

5.2.1. Processing An Incoming Request

   In RFC1157 [2], section 4.1, item (3) for an entity which receives a
   message, states that various parameters are passed to the 'desired
   authentication scheme.' The desired authentication scheme in this
   case is the SNMPv1 Community-Based Security Model, which will be
   called using the processIncomingMsg ASI. The parameters passed to
   this ASI are:

      - The messageProcessingModel, which will be 0 (or 1 for SNMPv2c).

      - The maxMessageSize, which should be the maximum size of a
         message that the receiving entity can generate (since there is
         no such value in the received message).

      - The securityParameters, which consist of the community string
         and the message's source and destination transport domains and
         addresses.

      - The securityModel, which will be 1 (or 2 for SNMPv2c).

      - The securityLevel, which will be noAuthNoPriv.

      - The wholeMsg and wholeMsgLength.

   The Community-Based Security Model will attempt to select a row in
   the snmpCommunityTable. This is done by performing a search through
   the snmpCommunityTable in lexicographic order. The first entry for
   which the following matching criteria are satisfied will be selected:

      - The community string is equal to the snmpCommunityName value.

      - If the snmpCommunityTransportTag is an empty string, it is
         ignored for the purpose of matching. If the
         snmpCommunityTransportTag is not an empty string, the
         transportDomain and transportAddress from which the message was
         received must match one of the entries in the
         snmpTargetAddrTable selected by the snmpCommunityTransportTag

value. The snmpTargetAddrTMask object is used as described in
section 5.3 when checking whether the transportDomain and
transportAddress matches a entry in the snmpTargetAddrTable.

If no such entry can be found, an authentication failure occurs as
described in RFC1157 [2], and the snmpInBadCommunityNames counter is
incremented.

The parameters returned from the Community-Based Security Model are:

-   The securityEngineID, which will always be the local value of
    snmpEngineID.0.

-   The securityName.

-   The scopedPDU. Note that this parameter will actually consist
    of three values, the contextSnmpEngineID, the contextName, and
    the PDU. These must be separate values, since the first two do
    not actually appear in the message.

-   The maxSizeResponseScopedPDU.

-   The securityStateReference.

The appropriate SNMP application will then be called (depending on
the value of the contextEngineID and the request type in the PDU)
using the processPdu ASI. The parameters passed to this ASI are:

-   The messageProcessingModel, which will be 0 (or 1 for SNMPv2c).

-   The securityModel, which will be 1 (or 2 for SNMPv2c).

-   The securityName, which was returned from the call to
    processIncomingMsg.

-   The securityLevel, which is noAuthNoPriv.

-   The contextEngineID, which was returned as part of the
    ScopedPDU from the call to processIncomingMsg.

-   The contextName, which was returned as part of the ScopedPDU
    from the call to processIncomingMsg.

-   The pduVersion, which should indicate an SNMPv1 version PDU (if
    the message version was SNMPv2c, this would be an SNMPv2
    version PDU).

- The PDU, which was returned as part of the ScopedPDU from the
  call to processIncomingMsg.

- The maxSizeResponseScopedPDU which was returned from the call
  to processIncomingMsg.

- The stateReference which was returned from the call to
  processIncomingMsg.

The SNMP application should process the request as described
previously in this document. Note that access control is applied by
an SNMPv3 command responder application as usual. The parameters as
passed to the processPdu ASI will be used in calls to the
isAccessAllowed ASI.

5.2.2. Generating An Outgoing Response

There is no special processing required for generating an outgoing
response. However, the community string used in an outgoing response
must be the same as the community string from the original request.
The original community string MUST be present in the stateReference
information of the original request.

5.2.3. Generating An Outgoing Notification

In a multi-lingual SNMP entity, the parameters used for generating
notifications will be obtained by examining the SNMP-TARGET-MIB and
SNMP-NOTIFICATION-MIB. These parameters will be passed to the SNMPv1
Message Processing Model using the sendPdu ASI. The SNMPv1 Message
Processing Model will attempt to locate an appropriate community
string in the snmpCommunityTable based on the parameters passed to
the sendPdu ASI. This is done by performing a search through the
snmpCommunityTable in lexicographic order. The first entry for which
the following matching criteria are satisfied will be selected:

- The securityName must be equal to the snmpCommunitySecurityName
  value.

- The contextEngineID must be equal to the
  snmpCommunityContextEngineID value.

- The contextName must be equal to the snmpCommunityContextName
  value.

- If the snmpCommunityTransportTag is an empty string, it is
  ignored for the purpose of matching. If the
  snmpCommunityTransportTag is not an empty string, the

transportDomain and transportAddress must match one of the
entries in the snmpTargetAddrTable selected by the
snmpCommunityTransportTag value.

If no such entry can be found, the notification is not sent.
Otherwise, the community string used in the outgoing notification
will be the value of the snmpCommunityName column of the selected
row.

5.3. The SNMP Community MIB Module

The SNMP-COMMUNITY-MIB contains objects for mapping between community
strings and version-independent SNMP message parameters. In
addition, this MIB provides a mechanism for performing source address
validation on incoming requests, and for selecting community strings
based on target addresses for outgoing notifications. These two
features are accomplished by providing a tag in the
snmpCommunityTable which selects sets of entries in the
snmpTargetAddrTable [18]. In addition, the SNMP-COMMUNITY-MIB
augments the snmpTargetAddrTable with a transport address mask value
and a maximum message size value. These values are used only where
explicitly stated. In cases where the snmpTargetAddrTable is used
without mention of these augmenting values, the augmenting values
should be ignored.

The mask value, snmpTargetAddrTMask, allows selected entries in the
snmpTargetAddrTable to specify multiple addresses (rather than just a
single address per entry). This would typically be used to specify a
subnet in an snmpTargetAddrTable rather than just a single address.
The mask value is used to select which bits of a transport address
must match bits of the corresponding instance of
snmpTargetAddrTAddress, in order for the transport address to match a
particular entry in the snmpTargetAddrTable. The value of an
instance of snmpTargetAddrTMask must always be an OCTET STRING whose
length is either zero or the same as that of the corresponding
instance of snmpTargetAddrTAddress.

Note that the snmpTargetAddrTMask object is only used where
explicitly stated. In particular, it is not used when generating
notifications (i.e., when generating notifications, entries in the
snmpTargetAddrTable only specify individual addresses).

When checking whether a transport address matches an entry in the
snmpTargetAddrTable, if the value of snmpTargetAddrTMask is a zero-
length OCTET STRING, the mask value is ignored, and the value of
snmpTargetAddrTAddress must exactly match a transport address.
Otherwise, each bit of each octet in the snmpTargetAddrTMask value
corresponds to the same bit of the same octet in the

     snmpTargetAddrTAddress value. For bits that are set in the
     snmpTargetAddrTMask value (i.e., bits equal to 1), the corresponding
     bits in the snmpTargetAddrTAddress value must match the bits in a
     transport address. If all such bits match, the transport address is
     matched by that snmpTargetAddrTable entry. Otherwise, the transport
     address is not matched.

     The maximum message size value, snmpTargetAddrMMS, is used to
     determine the maximum message size acceptable to another SNMP entity
     when the value cannot be determined from the protocol.

  SNMP-COMMUNITY-MIB DEFINITIONS ::= BEGIN

  IMPORTS
      IpAddress,
      MODULE-IDENTITY,
      OBJECT-TYPE,
      Integer32,
      snmpModules
          FROM SNMPv2-SMI
      RowStatus,
      StorageType
          FROM SNMPv2-TC
      SnmpAdminString,
      SnmpEngineID
          FROM SNMP-FRAMEWORK-MIB
      SnmpTagValue,
      snmpTargetAddrEntry
          FROM SNMP-TARGET-MIB
      MODULE-COMPLIANCE,
      OBJECT-GROUP
          FROM SNMPv2-CONF;

  snmpCommunityMIB MODULE-IDENTITY
      LAST-UPDATED "200003060000Z"   -- 6 Mar 2000, midnight
      ORGANIZATION "SNMPv3 Working Group"
      CONTACT-INFO "WG-email: snmpv3@lists.tislabs.com
                    Subscribe: majordomo@lists.tislabs.com
                                In msg body: subscribe snmpv3

                    Chair: Russ Mundy
                                TIS Labs at Network Associates
                    Postal: 3060 Washington Rd
                                Glenwood MD 21738
                                USA
                    Email: mundy@tislabs.com
                    Phone: +1-301-854-6889

                    Co-editor: Rob Frye
                                 CoSine Communications
                    Postal: 1200 Bridge Parkway
                                 Redwood City, CA 94065
                                 USA
                    E-mail: rfrye@cosinecom.com
                    Phone: +1 703 725 1130

                    Co-editor: David B. Levi
                                 Nortel Networks
                    Postal: 3505 Kesterwood Drive
                                 Knoxville, TN 37918
                    E-mail: dlevi@nortelnetworks.com
                    Phone: +1 423 686 0432

                    Co-editor: Shawn A. Routhier
                                 Integrated Systems Inc.
                    Postal: 333 North Ave 4th Floor
                                 Wakefield, MA 01880
                    E-mail: sar@epilogue.com
                    Phone: +1 781 245 0804

                    Co-editor: Bert Wijnen
                                 Lucent Technologies
                    Postal: Schagen 33
                                 3461 GL Linschoten
                                 Netherlands
                    Email: bwijnen@lucent.com
                    Phone: +31-348-407-775
                    "

        DESCRIPTION
            "This MIB module defines objects to help support coexistence
             between SNMPv1, SNMPv2c, and SNMPv3."
        REVISION "200003060000Z" -- 6 Mar 2000
        DESCRIPTION "This version published as RFC 2576."
        REVISION "199905130000Z" -- 13 May 1999
        DESCRIPTION "The Initial Revision"
    ::= { snmpModules 18 }

-- Administrative assignments ***************************************

snmpCommunityMIBObjects   OBJECT IDENTIFIER ::= { snmpCommunityMIB 1 }
snmpCommunityMIBConformance OBJECT IDENTIFIER ::= { snmpCommunityMIB 2 }

--
-- The snmpCommunityTable contains a database of community strings.
-- This table provides mappings between community strings, and the

-- parameters required for View-based Access Control.
--

snmpCommunityTable OBJECT-TYPE
    SYNTAX   SEQUENCE OF SnmpCommunityEntry
    MAX-ACCESS   not-accessible
    STATUS   current
    DESCRIPTION
        "The table of community strings configured in the SNMP
         engine's Local Configuration Datastore (LCD)."
    ::= { snmpCommunityMIBObjects 1 }

snmpCommunityEntry OBJECT-TYPE
    SYNTAX   SnmpCommunityEntry
    MAX-ACCESS   not-accessible
    STATUS   current
    DESCRIPTION
        "Information about a particular community string."
    INDEX   { IMPLIED snmpCommunityIndex }
    ::= { snmpCommunityTable 1 }

SnmpCommunityEntry ::= SEQUENCE {
    snmpCommunityIndex   SnmpAdminString,
    snmpCommunityName   OCTET STRING,
    snmpCommunitySecurityName   SnmpAdminString,
    snmpCommunityContextEngineID   SnmpEngineID,
    snmpCommunityContextName   SnmpAdminString,
    snmpCommunityTransportTag   SnmpTagValue,
    snmpCommunityStorageType   StorageType,
    snmpCommunityStatus   RowStatus
}

snmpCommunityIndex OBJECT-TYPE
    SYNTAX   SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   not-accessible
    STATUS   current
    DESCRIPTION
        "The unique index value of a row in this table."
    ::= { snmpCommunityEntry 1 }

snmpCommunityName OBJECT-TYPE
    SYNTAX   OCTET STRING
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION
        "The community string for which a row in this table
         represents a configuration."
    ::= { snmpCommunityEntry 2 }

```
snmpCommunitySecurityName OBJECT-TYPE
    SYNTAX        SnmpAdminString (SIZE(1..32))
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION
        "A human readable string representing the corresponding
         value of snmpCommunityName in a Security Model
         independent format."
    ::= { snmpCommunityEntry 3 }

snmpCommunityContextEngineID OBJECT-TYPE
    SYNTAX        SnmpEngineID
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION
        "The contextEngineID indicating the location of the
         context in which management information is accessed
         when using the community string specified by the
         corresponding instance of snmpCommunityName.

         The default value is the snmpEngineID of the entity in
         which this object is instantiated."
    ::= { snmpCommunityEntry 4 }

snmpCommunityContextName OBJECT-TYPE
    SYNTAX        SnmpAdminString (SIZE(0..32))
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION
        "The context in which management information is accessed
         when using the community string specified by the corresponding
         instance of snmpCommunityName."
    DEFVAL        { ''H }    -- the empty string
    ::= { snmpCommunityEntry 5 }

snmpCommunityTransportTag OBJECT-TYPE
    SYNTAX        SnmpTagValue
    MAX-ACCESS    read-create
    STATUS        current
    DESCRIPTION
        "This object specifies a set of transport endpoints
         from which a command responder application will accept
         management requests.  If a management request containing
         this community is received on a transport endpoint other
         than the transport endpoints identified by this object,
         the request is deemed unauthentic.

         The transports identified by this object are specified
```

         in the snmpTargetAddrTable. Entries in that table
         whose snmpTargetAddrTagList contains this tag value
         are identified.

         If the value of this object has zero-length, transport
         endpoints are not checked when authenticating messages
         containing this community string."
    DEFVAL   { ''H }   -- the empty string
    ::= { snmpCommunityEntry 6 }

snmpCommunityStorageType OBJECT-TYPE
    SYNTAX   StorageType
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION
        "The storage type for this conceptual row in the
         snmpCommunityTable. Conceptual rows having the value
         'permanent' need not allow write-access to any
         columnar object in the row."
    ::= { snmpCommunityEntry 7 }

snmpCommunityStatus OBJECT-TYPE
    SYNTAX   RowStatus
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION
        "The status of this conceptual row in the snmpCommunityTable.

         An entry in this table is not qualified for activation
         until instances of all corresponding columns have been
         initialized, either through default values, or through
         Set operations. The snmpCommunityName and
         snmpCommunitySecurityName objects must be explicitly set.

         There is no restriction on setting columns in this table
         when the value of snmpCommunityStatus is active(1)."
    ::= { snmpCommunityEntry 8 }

--
-- The snmpTargetAddrExtTable
--

snmpTargetAddrExtTable OBJECT-TYPE
    SYNTAX   SEQUENCE OF SnmpTargetAddrExtEntry
    MAX-ACCESS   not-accessible
    STATUS   current
    DESCRIPTION
        "The table of mask and mms values associated with the

             snmpTargetAddrTable.

             The snmpTargetAddrExtTable augments the
             snmpTargetAddrTable with a transport address mask value
             and a maximum message size value. The transport address
             mask allows entries in the snmpTargetAddrTable to define
             a set of addresses instead of just a single address.
             The maximum message size value allows the maximum
             message size of another SNMP entity to be configured for
             use in SNMPv1 (and SNMPv2c) transactions, where the
             message format does not specify a maximum message size."
        ::= { snmpCommunityMIBObjects 2 }

snmpTargetAddrExtEntry OBJECT-TYPE
     SYNTAX   SnmpTargetAddrExtEntry
     MAX-ACCESS   not-accessible
     STATUS   current
     DESCRIPTION
         "Information about a particular mask and mms value."
     AUGMENTS   { snmpTargetAddrEntry }
     ::= { snmpTargetAddrExtTable 1 }

SnmpTargetAddrExtEntry ::= SEQUENCE {
     snmpTargetAddrTMask   OCTET STRING,
     snmpTargetAddrMMS   Integer32
}

snmpTargetAddrTMask OBJECT-TYPE
     SYNTAX   OCTET STRING (SIZE (0..255))
     MAX-ACCESS   read-create
     STATUS   current
     DESCRIPTION
         "The mask value associated with an entry in the
          snmpTargetAddrTable. The value of this object must
          have the same length as the corresponding instance of
          snmpTargetAddrTAddress, or must have length 0. An
          attempt to set it to any other value will result in
          an inconsistentValue error.

          The value of this object allows an entry in the
          snmpTargetAddrTable to specify multiple addresses.
          The mask value is used to select which bits of
          a transport address must match bits of the corresponding
          instance of snmpTargetAddrTAddress, in order for the
          transport address to match a particular entry in the
          snmpTargetAddrTable. Bits which are 1 in the mask
          value indicate bits in the transport address which
          must match bits in the snmpTargetAddrTAddress value.

         Bits which are 0 in the mask indicate bits in the
         transport address which need not match. If the
         length of the mask is 0, the mask should be treated
         as if all its bits were 1 and its length were equal
         to the length of the corresponding value of
         snmpTargetAddrTable.

         This object may not be modified while the value of the
         corresponding instance of snmpTargetAddrRowStatus is
         active(1). An attempt to set this object in this case
         will result in an inconsistentValue error."
    DEFVAL { ''H }
    ::= { snmpTargetAddrExtEntry 1 }

snmpTargetAddrMMS OBJECT-TYPE
    SYNTAX   Integer32 (0|484..2147483647)
    MAX-ACCESS   read-create
    STATUS   current
    DESCRIPTION
        "The maximum message size value associated with an entry
         in the snmpTargetAddrTable."
    DEFVAL { 484 }
    ::= { snmpTargetAddrExtEntry 2 }

--
-- The snmpTrapAddress and snmpTrapCommunity objects are included
-- in notifications that are forwarded by a proxy, which were
-- originally received as SNMPv1 Trap messages.
--

snmpTrapAddress OBJECT-TYPE
    SYNTAX   IpAddress
    MAX-ACCESS   accessible-for-notify
    STATUS   current
    DESCRIPTION
        "The value of the agent-addr field of a Trap PDU which
         is forwarded by a proxy forwarder application using
         an SNMP version other than SNMPv1. The value of this
         object SHOULD contain the value of the agent-addr field
         from the original Trap PDU as generated by an SNMPv1
         agent."
    ::= { snmpCommunityMIBObjects 3 }

snmpTrapCommunity OBJECT-TYPE
    SYNTAX   OCTET STRING
    MAX-ACCESS   accessible-for-notify
    STATUS   current
    DESCRIPTION

```
        "The value of the community string field of an SNMPv1
         message containing a Trap PDU which is forwarded by a
         a proxy forwarder application using an SNMP version
         other than SNMPv1.  The value of this object SHOULD
         contain the value of the community string field from
         the original SNMPv1 message containing a Trap PDU as
         generated by an SNMPv1 agent."
    ::= { snmpCommunityMIBObjects 4 }

-- Conformance Information ****************************************

snmpCommunityMIBCompliances OBJECT IDENTIFIER
                        ::= { snmpCommunityMIBConformance 1 }
snmpCommunityMIBGroups      OBJECT IDENTIFIER
                        ::= { snmpCommunityMIBConformance 2 }


-- Compliance statements

snmpCommunityMIBCompliance MODULE-COMPLIANCE
    STATUS        current
    DESCRIPTION
        "The compliance statement for SNMP engines which
         implement the SNMP-COMMUNITY-MIB."

    MODULE        -- this module
        MANDATORY-GROUPS { snmpCommunityGroup }

        OBJECT            snmpCommunityName
        MIN-ACCESS        read-only
        DESCRIPTION     "Write access is not required."

        OBJECT            snmpCommunitySecurityName
        MIN-ACCESS        read-only
        DESCRIPTION     "Write access is not required."

        OBJECT            snmpCommunityContextEngineID
        MIN-ACCESS        read-only
        DESCRIPTION     "Write access is not required."

        OBJECT            snmpCommunityContextName
        MIN-ACCESS        read-only
        DESCRIPTION     "Write access is not required."

        OBJECT            snmpCommunityTransportTag
        MIN-ACCESS        read-only
        DESCRIPTION     "Write access is not required."

        OBJECT            snmpCommunityStorageType
```

```
        MIN-ACCESS        read-only
        DESCRIPTION       "Write access is not required."

        OBJECT            snmpCommunityStatus
        MIN-ACCESS        read-only
        DESCRIPTION       "Write access is not required."

    ::= { snmpCommunityMIBCompliances 1 }

snmpProxyTrapForwardCompliance MODULE-COMPLIANCE
    STATUS       current
    DESCRIPTION
        "The compliance statement for SNMP engines which
         contain a proxy forwarding application which is
         capable of forwarding SNMPv1 traps using SNMPv2c
         or SNMPv3."
    MODULE        -- this module
        MANDATORY-GROUPS { snmpProxyTrapForwardGroup }
    ::= { snmpCommunityMIBCompliances 2 }

snmpCommunityGroup OBJECT-GROUP
    OBJECTS {
        snmpCommunityName,
        snmpCommunitySecurityName,
        snmpCommunityContextEngineID,
        snmpCommunityContextName,
        snmpCommunityTransportTag,
        snmpCommunityStorageType,
        snmpCommunityStatus,
        snmpTargetAddrTMask,
        snmpTargetAddrMMS
    }
    STATUS       current
    DESCRIPTION
        "A collection of objects providing for configuration
         of community strings for SNMPv1 (and SNMPv2c) usage."
    ::= { snmpCommunityMIBGroups 1 }

snmpProxyTrapForwardGroup OBJECT-GROUP
    OBJECTS {
        snmpTrapAddress,
        snmpTrapCommunity
    }
    STATUS       current
    DESCRIPTION
        "Objects which are used by proxy forwarding applications
         when translating traps between SNMP versions.  These are
         used to preserve SNMPv1-specific information when
```

```
        translating to SNMPv2c or SNMPv3."
    ::= { snmpCommunityMIBGroups 3 }
```

END

6. Intellectual Property

The IETF takes no position regarding the validity or scope of any
intellectual property or other rights that might be claimed to pertain
to the implementation or use of the technology described in this
document or the extent to which any license under such rights might or
might not be available; neither does it represent that it has made any
effort to identify any such rights. Information on the IETF's
procedures with respect to rights in standards-track and standards-
related documentation can be found in BCP-11. Copies of claims of
rights made available for publication and any assurances of licenses to
be made available, or the result of an attempt made to obtain a general
license or permission for the use of such proprietary rights by
implementors or users of this specification can be obtained from the
IETF Secretariat.

The IETF invites any interested party to bring to its attention any
copyrights, patents or patent applications, or other proprietary rights
which may cover technology that may be required to practice this
standard. Please address the information to the IETF Executive
Director.

7. Acknowledgments

This document is the result of the efforts of the SNMPv3 Working Group.
The design of the SNMP-COMMUNITY-MIB incorporates work done by the
authors of SNMPv2*:

     Jeff Case (SNMP Research, Inc.)
     David Harrington (Cabletron Systems Inc.)
     David Levi (SNMP Research, Inc.)
     Brian O'Keefe (Hewlett Packard)
     Jon Saperia (IronBridge Networks, Inc.)
     Steve Waldbusser (International Network Services)

8. Security Considerations

    Although SNMPv1 and SNMPv2 do not provide any security, allowing
    community names to be mapped into securityName/contextName provides
    the ability to use view-based access control to limit the access of
    unsecured SNMPv1 and SNMPv2 operations. In fact, it is important for
    network administrators to make use of this capability in order to
    avoid unauthorized access to MIB data that would otherwise be secure.

    Further, the SNMP-COMMUNITY-MIB has the potential to expose community
    strings which provide access to more information than that which is
    available using the usual 'public' community string. For this
    reason, a security administrator may wish to limit accessibility to
    the SNMP-COMMUNITY-MIB, and in particular, to make it inaccessible
    when using the 'public' community string.

    When a proxy implementation translates messages between SNMPv1 (or
    SNMPv2c) and SNMPv3, there may be a loss of security. For example,
    an SNMPv3 message received using authentication and privacy which is
    subsequently forwarded using SNMPv1 will lose the security benefits
    of using authentication and privacy. Careful configuration of
    proxies is required to address such situations. One approach to deal
    with such situations might be to use an encrypted tunnel.

9. References

    [1] Rose, M. and K. McCloghrie, "Structure and Identification of
         Management Information for TCP/IP-based internets", STD 16, RFC
         1155, May 1990.

    [2] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple
         Network Management Protocol", STD 15, RFC 1157, May 1990.

    [3] McCloghrie, K. and M. Rose, Editors, "Concise MIB Definitions",
         STD 16, RFC 1212, March 1991.

    [4] Rose, M., "A Convention for Defining Traps for use with the
         SNMP", RFC 1215, March 1991.

    [5] McCloghrie, K. and M. Rose, "A Convention for Describing SNMP-
         based Agents", RFC 1303, February 1992.

    [6] Case, J., McCloghrie, K., Rose, M. and S.Waldbusser,
         "Introduction to Community-based SNMPv2", RFC 1901, January
         1996.

   [7] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose,
        M. and S. Waldbusser, "Structure of Management Information
        Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

   [8] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose,
        M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58,
        RFC 2579, April 1999.

   [9] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose,
        M. and S. Waldbusser, "Conformance Statements for SMIv2", STD
        58, RFC 2580, April 1999.

   [10] Case, J., McCloghrie, K., Rose, M. and S.Waldbusser, "Protocol
        Operations for Version 2 of the Simple Network Management
        Protocol (SNMPv2)", RFC 1905, January 1996.

   [11] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport
        Mappings for Version 2 of the Simple Network Management Protocol
        (SNMPv2)", RFC 1906, January 1996.

   [12] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser,
        "Management Information Base for Version 2 of the Simple Network
        Management Protocol (SNMPv2)", RFC 1907, January 1996.

   [13] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser,
        "Coexistence between Version 1 and Version 2 of the Internet-
        standard Network Management Framework", RFC 1908, January 1996.

   [14] Levi, D. and B. Wijnen, "Mapping SNMPv2 onto SNMPv1 within a
        bi-lingual SNMP agent", RFC 2089, January 1997.

   [15] Bradner, S., "Key words for use in RFCs to Indicate Requirement
        Levels", BCP 14, RFC 2119, March 1997.

   [16] Harrington, D. and B. Wijnen, "An Architecture for Describing
        SNMP Management Frameworks", RFC 2571, May 1999.

   [17] Case, J., Harrington, D. and B. Wijnen, "Message Processing and
        Dispatching for the Simple Network Management Protocol (SNMP)",
        RFC 2572, May 1999.

   [18] Levi, D., Meyer, P. and B. Stewart, "SNMP Applications", RFC
        2573, May 1999.

   [19] Blumenthal, U. and Wijnen, B., "The User-Based Security Model
        for Version 3 of the Simple Network Management Protocol (SNMP)",
        RFC 2574, May 1999.

   [20] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access
        Control Model for the Simple Network Management Protocol
        (SNMP)", RFC 2575, May 1999.

10. Editor's Addresses

   Rob Frye
   CoSine Communications
   1200 Bridge Parkway
   Redwood City, CA 94065
   U.S.A.

   Phone: +1 703 725 1130
   EMail: rfrye@cosinecom.com

   David B. Levi
   Nortel Networks
   3505 Kesterwood Drive
   Knoxville, TN 37918
   U.S.A.

   Phone: +1 423 686 0432
   EMail: dlevi@nortelnetworks.com

   Shawn A. Routhier
   Integrated Systems Inc.
   333 North Ave 4th Floor
   Wakefield MA 01880
   U.S.A.

   Phone: + 1 781 245 0804
   EMail: sar@epilogue.com

   Bert Wijnen
   Lucent Technologies
   Schagen 33
   3461 GL Linschoten
   Netherlands

   Phone: +31 348 407-775
   EMail: wijnen@lucent.com

A. Changes From RFC1908

    -   Editorial changes to comply with current RFC requirements.

    -   Added/updated copyright statements.

    -   Added Intellectual Property section.

    -   Replaced old introduction with complete new introduction/overview.

    -   Added content for the Security Considerations Section.

    -   Updated References to current documents.

    -   Updated text to use current SNMP terminology.

    -   Added coexistence for/with SNMPv3.

    -   Added description for SNMPv1 and SNMPv2c Message Processing
        Models and SNMPv1 and SNMPv2c Community-based Security
        Models.

    -   Added snmpCommunityMIB so that SNMPv1 and SNMPv2 community
        strings can be mapped into the SNMP Version Independent
        paramaters which can then be used for access control using the
        standard SNMPv3 View-based Access Control Model and the
        snmpVacmMIB.

    -   Added two MIB objects such that when an SNMPv1 notification
        (trap) must be converted into an SNMPv2 notification we add
        those two objects in order to preserve information about the
        address and community of the originating SNMPv1 agent.

    -   Included (and extended) from RFC2089 the SNMPv2 to SNMPv1
        mapping within a multi-lingual SNMP Engine.

    -   Use keywords from RFC 2119 to describe requirements for
        compliance.

    -   Changed/added some rules for converting a MIB module from
        SMIv1 to SMIv2.

    -   Extended and improved the description of Proxy Forwarder
        behaviour when multiple SNMP versions are involved.

Full Copyright Statement

Acknowledgement