Network Working Group
Request for Comments: 3416
STD: 62
Obsoletes: 1905
Category: Standards Track

Editor of this version:
R. Presuhn
BMC Software, Inc.
Authors of previous version:
J. Case
SNMP Research, Inc.
K. McCloghrie
Cisco Systems, Inc.
M. Rose
Dover Beach Consulting, Inc.
S. Waldbusser
International Network Services

December 2002

Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)

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.

Copyright Notice

Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

This document defines version 2 of the protocol operations for the Simple Network Management Protocol (SNMP). It defines the syntax and elements of procedure for sending, receiving, and processing SNMP PDUs. This document obsoletes RFC 1905.

Table of Contents

1. Introduction

The SNMP Management Framework at the time of this writing consists of five major components:

- An overall architecture, described in STD 62, RFC 3411 [RFC3411].

- Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in STD 16, RFC 1155 [RFC1155], STD 16, RFC 1212 [RFC1212] and RFC 1215 [RFC1215]. The second version, called SMIv2, is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157 [RFC1157]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [RFC1901] and STD 62, RFC 3417 [RFC3417]. The third version of the message protocol is called SNMPv3 and described in STD 62, RFC 3417 [RFC3417], RFC 3412 [RFC3412] and RFC 3414 [RFC3414].

- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in STD 15, RFC 1157 [RFC1157]. A second set of protocol operations and associated PDU formats is described in this document.

- A set of fundamental applications described in STD 62, RFC 3413 [RFC3413] and the view-based access control mechanism described in STD 62, RFC 3415 [RFC3415].

A more detailed introduction to the SNMP Management Framework at the time of this writing can be found in RFC 3410 [RFC3410].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

This document, Version 2 of the Protocol Operations for the Simple Network Management Protocol, defines the operations of the protocol with respect to the sending and receiving of PDUs to be carried by the message protocol.

2. Overview

SNMP entities supporting command generator or notification receiver applications (traditionally called "managers") communicate with SNMP entities supporting command responder or notification originator applications (traditionally called "agents"). The purpose of this protocol is the transport of management information and operations.

2.1. Management Information

The term "variable" refers to an instance of a non-aggregate object type defined according to the conventions set forth in the SMI [RFC2578] or the textual conventions based on the SMI [RFC2579]. The term "variable binding" normally refers to the pairing of the name of a variable and its associated value. However, if certain kinds of exceptional conditions occur during processing of a retrieval request, a variable binding will pair a name and an indication of that exception.

A variable-binding list is a simple list of variable bindings.

The name of a variable is an OBJECT IDENTIFIER which is the concatenation of the OBJECT IDENTIFIER of the corresponding object-type together with an OBJECT IDENTIFIER fragment identifying the instance. The OBJECT IDENTIFIER of the corresponding object-type is called the OBJECT IDENTIFIER prefix of the variable.

2.2. Retransmission of Requests

For all types of request in this protocol, the receiver is required under normal circumstances, to generate and transmit a response to the originator of the request. Whether or not a request should be retransmitted if no corresponding response is received in an appropriate time interval, is at the discretion of the application originating the request. This will normally depend on the urgency of the request. However, such an application needs to act responsibly in respect to the frequency and duration of re-transmissions. See BCP 41 [RFC2914] for discussion of relevant congestion control principles.

2.3. Message Sizes

The maximum size of an SNMP message is limited to the minimum of:

(1) the maximum message size which the destination SNMP entity can accept; and,

(2) the maximum message size which the source SNMP entity can generate.

The former may be known on a per-recipient basis; and in the absence of such knowledge, is indicated by transport domain used when sending the message. The latter is imposed by implementation-specific local constraints.

Each transport mapping for the SNMP indicates the minimum message size which a SNMP implementation must be able to produce or consume. Although implementations are encouraged to support larger values whenever possible, a conformant implementation must never generate messages larger than allowed by the receiving SNMP entity.

One of the aims of the GetBulkRequest-PDU, specified in this protocol, is to minimize the number of protocol exchanges required to retrieve a large amount of management information. As such, this PDU type allows an SNMP entity supporting command generator applications to request that the response be as large as possible given the constraints on message sizes. These constraints include the limits on the size of messages which the SNMP entity supporting command responder applications can generate, and the SNMP entity supporting command generator applications can receive.

However, it is possible that such maximum sized messages may be larger than the Path MTU of the path across the network traversed by the messages. In this situation, such messages are subject to fragmentation. Fragmentation is generally considered to be harmful [FRAG], since among other problems, it leads to a decrease in the reliability of the transfer of the messages. Thus, an SNMP entity which sends a GetBulkRequest-PDU must take care to set its parameters accordingly, so as to reduce the risk of fragmentation. In particular, under conditions of network stress, only small values should be used for max-repetitions.

2.4. Transport Mappings

It is important to note that the exchange of SNMP messages requires only an unreliable datagram service, with every message being entirely and independently contained in a single transport datagram. Specific transport mappings and encoding rules are specified elsewhere [RFC3417]. However, the preferred mapping is the use of the User Datagram Protocol [RFC768].

2.5. SMIv2 Data Type Mappings

The SMIv2 [RFC2578] defines 11 base types (INTEGER, OCTET STRING, OBJECT IDENTIFIER, Integer32, IpAddress, Counter32, Gauge32, Unsigned32, TimeTicks, Opaque, Counter64) and the BITS construct. The SMIv2 base types are mapped to the corresponding selection type in the SimpleSyntax and ApplicationSyntax choices of the ASN.1 SNMP protocol definition. Note that the INTEGER and Integer32 SMIv2 base types are mapped to the integer-value selection type of the SimpleSyntax choice. Similarly, the Gauge32 and Unsigned32 SMIv2 base types are mapped to the unsigned-integer-value selection type of the ApplicationSyntax choice.

The SMIv2 BITS construct is mapped to the string-value selection type of the SimpleSyntax choice. A BITS value is encoded as an OCTET STRING, in which all the named bits in (the definition of) the bitstring, commencing with the first bit and proceeding to the last bit, are placed in bits 8 (high order bit) to 1 (low order bit) of the first octet, followed by bits 8 to 1 of each subsequent octet in turn, followed by as many bits as are needed of the final subsequent octet, commencing with bit 8. Remaining bits, if any, of the final octet are set to zero on generation and ignored on receipt.

3. Definitions

The PDU syntax is defined using ASN.1 notation [ASN1].

```
SNMPv2-PDU DEFINITIONS ::= BEGIN

ObjectName ::= OBJECT IDENTIFIER

ObjectSyntax ::= CHOICE {
      simple           SimpleSyntax,
      application-wide ApplicationSyntax }

SimpleSyntax ::= CHOICE {
      integer-value   INTEGER (-2147483648..2147483647),
      string-value    OCTET STRING (SIZE (0..65535)),
      objectID-value  OBJECT IDENTIFIER }

ApplicationSyntax ::= CHOICE {
      ipAddress-value        IpAddress,
      counter-value          Counter32,
      timeticks-value        TimeTicks,
      arbitrary-value        Opaque,
      big-counter-value      Counter64,
      unsigned-integer-value Unsigned32 }
```

```
   IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING (SIZE (4))

   Counter32 ::= [APPLICATION 1] IMPLICIT INTEGER (0..4294967295)

   Unsigned32 ::= [APPLICATION 2] IMPLICIT INTEGER (0..4294967295)

   Gauge32 ::= Unsigned32

   TimeTicks ::= [APPLICATION 3] IMPLICIT INTEGER (0..4294967295)

   Opaque ::= [APPLICATION 4] IMPLICIT OCTET STRING

   Counter64 ::= [APPLICATION 6]
                 IMPLICIT INTEGER (0..18446744073709551615)

   -- protocol data units

   PDUs ::= CHOICE {
        get-request      GetRequest-PDU,
        get-next-request GetNextRequest-PDU,
        get-bulk-request GetBulkRequest-PDU,
        response         Response-PDU,
        set-request      SetRequest-PDU,
        inform-request   InformRequest-PDU,
        snmpV2-trap      SNMPv2-Trap-PDU,
        report           Report-PDU }

   -- PDUs

   GetRequest-PDU ::= [0] IMPLICIT PDU

   GetNextRequest-PDU ::= [1] IMPLICIT PDU

   Response-PDU ::= [2] IMPLICIT PDU

   SetRequest-PDU ::= [3] IMPLICIT PDU

   -- [4] is obsolete

   GetBulkRequest-PDU ::= [5] IMPLICIT BulkPDU

   InformRequest-PDU ::= [6] IMPLICIT PDU

   SNMPv2-Trap-PDU ::= [7] IMPLICIT PDU

   --   Usage and precise semantics of Report-PDU are not defined
   --   in this document.  Any SNMP administrative framework making
   --   use of this PDU must define its usage and semantics.
```

```
   Report-PDU ::= [8] IMPLICIT PDU

   max-bindings INTEGER ::= 2147483647

   PDU ::= SEQUENCE {
           request-id INTEGER (-214783648..214783647),

           error-status               -- sometimes ignored
               INTEGER {
                   noError(0),
                   tooBig(1),
                   noSuchName(2),      -- for proxy compatibility
                   badValue(3),        -- for proxy compatibility
                   readOnly(4),        -- for proxy compatibility
                   genErr(5),
                   noAccess(6),
                   wrongType(7),
                   wrongLength(8),
                   wrongEncoding(9),
                   wrongValue(10),
                   noCreation(11),
                   inconsistentValue(12),
                   resourceUnavailable(13),
                   commitFailed(14),
                   undoFailed(15),
                   authorizationError(16),
                   notWritable(17),
                   inconsistentName(18)
               },

           error-index                -- sometimes ignored
               INTEGER (0..max-bindings),

           variable-bindings          -- values are sometimes ignored
               VarBindList
       }

   BulkPDU ::=                        -- must be identical in
       SEQUENCE {                     -- structure to PDU
           request-id      INTEGER (-214783648..214783647),
           non-repeaters   INTEGER (0..max-bindings),
           max-repetitions INTEGER (0..max-bindings),

           variable-bindings          -- values are ignored
               VarBindList
       }

   -- variable binding
```

```
   VarBind ::= SEQUENCE {
           name ObjectName,

           CHOICE {
               value          ObjectSyntax,
               unSpecified    NULL,    -- in retrieval requests

                                       -- exceptions in responses
               noSuchObject   [0] IMPLICIT NULL,
               noSuchInstance [1] IMPLICIT NULL,
               endOfMibView   [2] IMPLICIT NULL
           }
       }

   -- variable-binding list

   VarBindList ::= SEQUENCE (SIZE (0..max-bindings)) OF VarBind

   END
```

4. Protocol Specification

4.1. Common Constructs

The value of the request-id field in a Response-PDU takes the value of the request-id field in the request PDU to which it is a response. By use of the request-id value, an application can distinguish the (potentially multiple) outstanding requests, and thereby correlate incoming responses with outstanding requests. In cases where an unreliable datagram service is used, the request-id also provides a simple means of identifying messages duplicated by the network. Use of the same request-id on a retransmission of a request allows the response to either the original transmission or the retransmission to satisfy the request. However, in order to calculate the round trip time for transmission and processing of a request-response transaction, the application needs to use a different request-id value on a retransmitted request. The latter strategy is recommended for use in the majority of situations.

A non-zero value of the error-status field in a Response-PDU is used to indicate that an error occurred to prevent the processing of the request. In these cases, a non-zero value of the Response-PDU's error-index field provides additional information by identifying which variable binding in the list caused the error. A variable binding is identified by its index value. The first variable binding in a variable-binding list is index one, the second is index two, etc.

SNMP limits OBJECT IDENTIFIER values to a maximum of 128 sub-identifiers, where each sub-identifier has a maximum value of 2**32-1.

4.2. PDU Processing

In the elements of procedure below, any field of a PDU which is not referenced by the relevant procedure is ignored by the receiving SNMP entity. However, all components of a PDU, including those whose values are ignored by the receiving SNMP entity, must have valid ASN.1 syntax and encoding. For example, some PDUs (e.g., the GetRequest-PDU) are concerned only with the name of a variable and not its value. In this case, the value portion of the variable binding is ignored by the receiving SNMP entity. The unSpecified value is defined for use as the value portion of such bindings.

On generating a management communication, the message "wrapper" to encapsulate the PDU is generated according to the "Elements of Procedure" of the administrative framework in use. The definition of "max-bindings" imposes an upper bound on the number of variable bindings. In practice, the size of a message is also limited by constraints on the maximum message size. A compliant implementation must support as many variable bindings in a PDU or BulkPDU as fit into the overall maximum message size limit of the SNMP engine, but no more than 2147483647 variable bindings.

On receiving a management communication, the "Elements of Procedure" of the administrative framework in use is followed, and if those procedures indicate that the operation contained within the message is to be performed locally, then those procedures also indicate the MIB view which is visible to the operation.

4.2.1. The GetRequest-PDU

A GetRequest-PDU is generated and transmitted at the request of an application.

Upon receipt of a GetRequest-PDU, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU. All fields of the Response-PDU have the same values as the corresponding fields of the received request except as indicated below. Each variable binding is processed as follows:

(1) If the variable binding’s name exactly matches the name of a variable accessible by this request, then the variable binding’s value field is set to the value of the named variable.

(2) Otherwise, if the variable binding's name does not have an OBJECT IDENTIFIER prefix which exactly matches the OBJECT IDENTIFIER prefix of any (potential) variable accessible by this request, then its value field is set to "noSuchObject".

(3) Otherwise, the variable binding's value field is set to "noSuchInstance".

If the processing of any variable binding fails for a reason other than listed above, then the Response-PDU is re-formatted with the same values in its request-id and variable-bindings fields as the received GetRequest-PDU, with the value of its error-status field set to "genErr", and the value of its error-index field is set to the index of the failed variable binding.

Otherwise, the value of the Response-PDU's error-status field is set to "noError", and the value of its error-index field is zero.

The generated Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetRequest-PDU.

Otherwise, an alternate Response-PDU is generated. This alternate Response-PDU is formatted with the same value in its request-id field as the received GetRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded.

4.2.2. The GetNextRequest-PDU

A GetNextRequest-PDU is generated and transmitted at the request of an application.

Upon receipt of a GetNextRequest-PDU, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU. All fields of the Response-PDU have the same values as the corresponding fields of the received request except as indicated below. Each variable binding is processed as follows:

(1) The variable is located which is in the lexicographically ordered list of the names of all variables which are

accessible by this request and whose name is the first lexicographic successor of the variable binding's name in the incoming GetNextRequest-PDU. The corresponding variable binding's name and value fields in the Response-PDU are set to the name and value of the located variable.

(2) If the requested variable binding's name does not lexicographically precede the name of any variable accessible by this request, i.e., there is no lexicographic successor, then the corresponding variable binding produced in the Response-PDU has its value field set to "endOfMibView", and its name field set to the variable binding's name in the request.

If the processing of any variable binding fails for a reason other than listed above, then the Response-PDU is re-formatted with the same values in its request-id and variable-bindings fields as the received GetNextRequest-PDU, with the value of its error-status field set to "genErr", and the value of its error-index field is set to the index of the failed variable binding.

Otherwise, the value of the Response-PDU's error-status field is set to "noError", and the value of its error-index field is zero.

The generated Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetNextRequest-PDU.

Otherwise, an alternate Response-PDU is generated. This alternate Response-PDU is formatted with the same values in its request-id field as the received GetNextRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetNextRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded.

4.2.2.1. Example of Table Traversal

An important use of the GetNextRequest-PDU is the traversal of conceptual tables of information within a MIB. The semantics of this type of request, together with the method of identifying individual instances of objects in the MIB, provides access to related objects in the MIB as if they enjoyed a tabular organization.

In the protocol exchange sketched below, an application retrieves the media-dependent physical address and the address-mapping type for each entry in the IP net-to-media Address Translation Table [RFC1213] of a particular network element. It also retrieves the value of sysUpTime [RFC3418], at which the mappings existed. Suppose that the command responder's IP net-to-media table has three entries:

Interface-Number	Network-Address	Physical-Address	Type
1	10.0.0.51	00:00:10:01:23:45	static
1	9.2.3.4	00:00:10:54:32:10	dynamic
2	10.0.0.15	00:00:10:98:76:54	dynamic

The SNMP entity supporting a command generator application begins by sending a GetNextRequest-PDU containing the indicated OBJECT IDENTIFIER values as the requested variable names:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress,
                  ipNetToMediaType )
```

The SNMP entity supporting a command responder application responds with a Response-PDU:

```
 Response (( sysUpTime.0 =  "123456" ),
            ( ipNetToMediaPhysAddress.1.9.2.3.4 = "000010543210" ),
          ( ipNetToMediaType.1.9.2.3.4 =  "dynamic" ))
```

The SNMP entity supporting the command generator application continues with:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress.1.9.2.3.4,
                  ipNetToMediaType.1.9.2.3.4 )
```

The SNMP entity supporting the command responder application responds with:

```
 Response (( sysUpTime.0 =  "123461" ),
            ( ipNetToMediaPhysAddress.1.10.0.0.51 = "000010012345" ),
          ( ipNetToMediaType.1.10.0.0.51 =  "static" ))
```

The SNMP entity supporting the command generator application continues with:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress.1.10.0.0.51,
                  ipNetToMediaType.1.10.0.0.51 )
```

The SNMP entity supporting the command responder application responds with:

```
 Response (( sysUpTime.0 =  "123466" ),
            ( ipNetToMediaPhysAddress.2.10.0.0.15 = "000010987654" ),
          ( ipNetToMediaType.2.10.0.0.15 =  "dynamic" ))
```

The SNMP entity supporting the command generator application continues with:

```
 GetNextRequest ( sysUpTime,
                  ipNetToMediaPhysAddress.2.10.0.0.15,
                  ipNetToMediaType.2.10.0.0.15 )
```

As there are no further entries in the table, the SNMP entity supporting the command responder application responds with the variables that are next in the lexicographical ordering of the accessible object names, for example:

```
 Response (( sysUpTime.0 =  "123471" ),
            ( ipNetToMediaNetAddress.1.9.2.3.4 = "9.2.3.4" ),
          ( ipRoutingDiscards.0 =  "2" ))
```

Note how, having reached the end of the column for ipNetToMediaPhysAddress, the second variable binding from the command responder application has now "wrapped" to the first row in the next column. Furthermore, note how, having reached the end of the ipNetToMediaTable for the third variable binding, the command responder application has responded with the next available object, which is outside that table. This response signals the end of the table to the command generator application.

4.2.3. The GetBulkRequest-PDU

A GetBulkRequest-PDU is generated and transmitted at the request of an application. The purpose of the GetBulkRequest-PDU is to request the transfer of a potentially large amount of data, including, but not limited to, the efficient and rapid retrieval of large tables.

Upon receipt of a GetBulkRequest-PDU, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU with its request-id field having the same value as in the request.

For the GetBulkRequest-PDU type, the successful processing of each variable binding in the request generates zero or more variable bindings in the Response-PDU. That is, the one-to-one mapping between the variable bindings of the GetRequest-PDU, GetNextRequest-

PDU, and SetRequest-PDU types and the resultant Response-PDUs does not apply for the mapping between the variable bindings of a GetBulkRequest-PDU and the resultant Response-PDU.

The values of the non-repeaters and max-repetitions fields in the request specify the processing requested. One variable binding in the Response-PDU is requested for the first N variable bindings in the request and M variable bindings are requested for each of the R remaining variable bindings in the request. Consequently, the total number of requested variable bindings communicated by the request is given by N + (M * R), where N is the minimum of: a) the value of the non-repeaters field in the request, and b) the number of variable bindings in the request; M is the value of the max-repetitions field in the request; and R is the maximum of: a) number of variable bindings in the request - N, and b) zero.

The receiving SNMP entity produces a Response-PDU with up to the total number of requested variable bindings communicated by the request. The request-id shall have the same value as the received GetBulkRequest-PDU.

If N is greater than zero, the first through the (N)-th variable bindings of the Response-PDU are each produced as follows:

(1) The variable is located which is in the lexicographically ordered list of the names of all variables which are accessible by this request and whose name is the first lexicographic successor of the variable binding's name in the incoming GetBulkRequest-PDU. The corresponding variable binding's name and value fields in the Response-PDU are set to the name and value of the located variable.

(2) If the requested variable binding's name does not lexicographically precede the name of any variable accessible by this request, i.e., there is no lexicographic successor, then the corresponding variable binding produced in the Response-PDU has its value field set to "endOfMibView", and its name field set to the variable binding's name in the request.

If M and R are non-zero, the (N + 1)-th and subsequent variable bindings of the Response-PDU are each produced in a similar manner. For each iteration i, such that i is greater than zero and less than or equal to M, and for each repeated variable, r, such that r is greater than zero and less than or equal to R, the (N + ((i-1) * R) + r)-th variable binding of the Response-PDU is produced as follows:

(1) The variable which is in the lexicographically ordered list of the names of all variables which are accessible by this request and whose name is the (i)-th lexicographic successor of the (N + r)-th variable binding's name in the incoming GetBulkRequest-PDU is located and the variable binding's name and value fields are set to the name and value of the located variable.

(2) If there is no (i)-th lexicographic successor, then the corresponding variable binding produced in the Response-PDU has its value field set to "endOfMibView", and its name field set to either the last lexicographic successor, or if there are no lexicographic successors, to the (N + r)-th variable binding's name in the request.

While the maximum number of variable bindings in the Response-PDU is bounded by N + (M * R), the response may be generated with a lesser number of variable bindings (possibly zero) for either of three reasons.

(1) If the size of the message encapsulating the Response-PDU containing the requested number of variable bindings would be greater than either a local constraint or the maximum message size of the originator, then the response is generated with a lesser number of variable bindings. This lesser number is the ordered set of variable bindings with some of the variable bindings at the end of the set removed, such that the size of the message encapsulating the Response-PDU is approximately equal to but no greater than either a local constraint or the maximum message size of the originator. Note that the number of variable bindings removed has no relationship to the values of N, M, or R.

(2) The response may also be generated with a lesser number of variable bindings if for some value of iteration i, such that i is greater than zero and less than or equal to M, that all of the generated variable bindings have the value field set to "endOfMibView". In this case, the variable bindings may be truncated after the (N + (i * R))-th variable binding.

(3) In the event that the processing of a request with many repetitions requires a significantly greater amount of processing time than a normal request, then a command responder application may terminate the request with less than the full number of repetitions, providing at least one repetition is completed.

If the processing of any variable binding fails for a reason other than listed above, then the Response-PDU is re-formatted with the same values in its request-id and variable-bindings fields as the received GetBulkRequest-PDU, with the value of its error-status field set to "genErr", and the value of its error-index field is set to the index of the variable binding in the original request which corresponds to the failed variable binding.

Otherwise, the value of the Response-PDU's error-status field is set to "noError", and the value of its error-index field to zero.

The generated Response-PDU (possibly with an empty variable-bindings field) is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the GetBulkRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded.

4.2.3.1. Another Example of Table Traversal

This example demonstrates how the GetBulkRequest-PDU can be used as an alternative to the GetNextRequest-PDU. The same traversal of the IP net-to-media table as shown in Section 4.2.2.1 is achieved with fewer exchanges.

The SNMP entity supporting the command generator application begins by sending a GetBulkRequest-PDU with the modest max-repetitions value of 2, and containing the indicated OBJECT IDENTIFIER values as the requested variable names:

```
 GetBulkRequest [ non-repeaters = 1, max-repetitions = 2 ]
                ( sysUpTime,
                  ipNetToMediaPhysAddress,
                  ipNetToMediaType )
```

The SNMP entity supporting the command responder application responds with a Response-PDU:

```
 Response (( sysUpTime.0 =  "123456" ),
             ( ipNetToMediaPhysAddress.1.9.2.3.4 = "000010543210" ),
          ( ipNetToMediaType.1.9.2.3.4 =  "dynamic" ),
             ( ipNetToMediaPhysAddress.1.10.0.0.51 = "000010012345" ),
          ( ipNetToMediaType.1.10.0.0.51 =  "static" ))
```

The SNMP entity supporting the command generator application continues with:

```
  GetBulkRequest [ non-repeaters = 1, max-repetitions = 2 ]
                  ( sysUpTime,
                    ipNetToMediaPhysAddress.1.10.0.0.51,
                    ipNetToMediaType.1.10.0.0.51 )
```

The SNMP entity supporting the command responder application responds with:

```
 Response (( sysUpTime.0 =  "123466" ),
            ( ipNetToMediaPhysAddress.2.10.0.0.15 = "000010987654" ),
            ( ipNetToMediaType.2.10.0.0.15 = "dynamic" ),
            ( ipNetToMediaNetAddress.1.9.2.3.4 = "9.2.3.4" ),
          ( ipRoutingDiscards.0 =  "2" ))
```

Note how, as in the first example, the variable bindings in the response indicate that the end of the table has been reached. The fourth variable binding does so by returning information from the next available column; the fifth variable binding does so by returning information from the first available object lexicographically following the table. This response signals the end of the table to the command generator application.

4.2.4. The Response-PDU

The Response-PDU is generated by an SNMP entity only upon receipt of a GetRequest-PDU, GetNextRequest-PDU, GetBulkRequest-PDU, SetRequest-PDU, or InformRequest-PDU, as described elsewhere in this document.

If the error-status field of the Response-PDU is non-zero, the value fields of the variable bindings in the variable binding list are ignored.

If both the error-status field and the error-index field of the Response-PDU are non-zero, then the value of the error-index field is the index of the variable binding (in the variable-binding list of the corresponding request) for which the request failed. The first variable binding in a request's variable-binding list is index one, the second is index two, etc.

A compliant SNMP entity supporting a command generator application must be able to properly receive and handle a Response-PDU with an error-status field equal to "noSuchName", "badValue", or "readOnly". (See sections 1.3 and 4.3 of [RFC2576].)

Upon receipt of a Response-PDU, the receiving SNMP entity presents its contents to the application which generated the request with the same request-id value. For more details, see [RFC3412].

4.2.5. The SetRequest-PDU

A SetRequest-PDU is generated and transmitted at the request of an application.

Upon receipt of a SetRequest-PDU, the receiving SNMP entity determines the size of a message encapsulating a Response-PDU having the same values in its request-id and variable-bindings fields as the received SetRequest-PDU, and the largest possible sizes of the error-status and error-index fields. If the determined message size is greater than either a local constraint or the maximum message size of the originator, then an alternate Response-PDU is generated, transmitted to the originator of the SetRequest-PDU, and processing of the SetRequest-PDU terminates immediately thereafter. This alternate Response-PDU is formatted with the same values in its request-id field as the received SetRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the SetRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded. Regardless, processing of the SetRequest-PDU terminates.

Otherwise, the receiving SNMP entity processes each variable binding in the variable-binding list to produce a Response-PDU. All fields of the Response-PDU have the same values as the corresponding fields of the received request except as indicated below.

The variable bindings are conceptually processed as a two phase operation. In the first phase, each variable binding is validated; if all validations are successful, then each variable is altered in the second phase. Of course, implementors are at liberty to implement either the first, or second, or both, of these conceptual phases as multiple implementation phases. Indeed, such multiple implementation phases may be necessary in some cases to ensure consistency.

The following validations are performed in the first phase on each variable binding until they are all successful, or until one fails:

(1) If the variable binding's name specifies an existing or non-existent variable to which this request is/would be denied access because it is/would not be in the appropriate MIB view, then the value of the Response-PDU's error-status field is set to "noAccess", and the value of its error-index field is set to the index of the failed variable binding.

(2) Otherwise, if there are no variables which share the same OBJECT IDENTIFIER prefix as the variable binding's name, and which are able to be created or modified no matter what new value is specified, then the value of the Response-PDU's error-status field is set to "notWritable", and the value of its error-index field is set to the index of the failed variable binding.

(3) Otherwise, if the variable binding's value field specifies, according to the ASN.1 language, a type which is inconsistent with that required for all variables which share the same OBJECT IDENTIFIER prefix as the variable binding's name, then the value of the Response-PDU's error-status field is set to "wrongType", and the value of its error-index field is set to the index of the failed variable binding.

(4) Otherwise, if the variable binding's value field specifies, according to the ASN.1 language, a length which is inconsistent with that required for all variables which share the same OBJECT IDENTIFIER prefix as the variable binding's name, then the value of the Response-PDU's error-status field is set to "wrongLength", and the value of its error-index field is set to the index of the failed variable binding.

(5) Otherwise, if the variable binding's value field contains an ASN.1 encoding which is inconsistent with that field's ASN.1 tag, then the value of the Response-PDU's error-status field is set to "wrongEncoding", and the value of its error-index field is set to the index of the failed variable binding. (Note that not all implementation strategies will generate this error.)

(6) Otherwise, if the variable binding's value field specifies a value which could under no circumstances be assigned to the variable, then the value of the Response-PDU's error-status field is set to "wrongValue", and the value of its error-index field is set to the index of the failed variable binding.

(7) Otherwise, if the variable binding's name specifies a variable which does not exist and could not ever be created (even though some variables sharing the same OBJECT IDENTIFIER prefix might under some circumstances be able to be created), then the value of the Response-PDU's error-status field is set to "noCreation", and the value of its error-index field is set to the index of the failed variable binding.

(8) Otherwise, if the variable binding's name specifies a variable which does not exist but can not be created under the present circumstances (even though it could be created under other circumstances), then the value of the Response-PDU's error-status field is set to "inconsistentName", and the value of its error-index field is set to the index of the failed variable binding.

(9) Otherwise, if the variable binding's name specifies a variable which exists but can not be modified no matter what new value is specified, then the value of the Response-PDU's error-status field is set to "notWritable", and the value of its error-index field is set to the index of the failed variable binding.

(10) Otherwise, if the variable binding's value field specifies a value that could under other circumstances be held by the variable, but is presently inconsistent or otherwise unable to be assigned to the variable, then the value of the Response-PDU's error-status field is set to "inconsistentValue", and the value of its error-index field is set to the index of the failed variable binding.

(11) When, during the above steps, the assignment of the value specified by the variable binding's value field to the specified variable requires the allocation of a resource which is presently unavailable, then the value of the Response-PDU's error-status field is set to "resourceUnavailable", and the value of its error-index field is set to the index of the failed variable binding.

(12) If the processing of the variable binding fails for a reason other than listed above, then the value of the Response-PDU's error-status field is set to "genErr", and the value of its error-index field is set to the index of the failed variable binding.

(13) Otherwise, the validation of the variable binding succeeds.

At the end of the first phase, if the validation of all variable bindings succeeded, then the value of the Response-PDU's error-status field is set to "noError" and the value of its error-index field is zero, and processing continues as follows.

For each variable binding in the request, the named variable is created if necessary, and the specified value is assigned to it. Each of these variable assignments occurs as if simultaneously with respect to all other assignments specified in the same request. However, if the same variable is named more than once in a single request, with different associated values, then the actual assignment made to that variable is implementation-specific.

If any of these assignments fail (even after all the previous validations), then all other assignments are undone, and the Response-PDU is modified to have the value of its error-status field set to "commitFailed", and the value of its error-index field set to the index of the failed variable binding.

If and only if it is not possible to undo all the assignments, then the Response-PDU is modified to have the value of its error-status field set to "undoFailed", and the value of its error-index field is set to zero. Note that implementations are strongly encouraged to take all possible measures to avoid use of either "commitFailed" or "undoFailed" - these two error-status codes are not to be taken as license to take the easy way out in an implementation.

Finally, the generated Response-PDU is encapsulated into a message, and transmitted to the originator of the SetRequest-PDU.

4.2.6. The SNMPv2-Trap-PDU

An SNMPv2-Trap-PDU is generated and transmitted by an SNMP entity on behalf of a notification originator application. The SNMPv2-Trap-PDU is often used to notify a notification receiver application at a logically remote SNMP entity that an event has occurred or that a condition is present. There is no confirmation associated with this notification delivery mechanism.

The destination(s) to which an SNMPv2-Trap-PDU is sent is determined in an implementation-dependent fashion by the SNMP entity. The first two variable bindings in the variable binding list of an SNMPv2-Trap-PDU are sysUpTime.0 [RFC3418] and snmpTrapOID.0 [RFC3418] respectively. If the OBJECTS clause is present in the invocation of the corresponding NOTIFICATION-TYPE macro, then each corresponding variable, as instantiated by this notification, is copied, in order,

to the variable-bindings field. If any additional variables are being included (at the option of the generating SNMP entity), then each is copied to the variable-bindings field.

4.2.7. The InformRequest-PDU

An InformRequest-PDU is generated and transmitted by an SNMP entity on behalf of a notification originator application. The InformRequest-PDU is often used to notify a notification receiver application that an event has occurred or that a condition is present. This is a confirmed notification delivery mechanism, although there is, of course, no guarantee of delivery.

The destination(s) to which an InformRequest-PDU is sent is specified by the notification originator application. The first two variable bindings in the variable binding list of an InformRequest-PDU are sysUpTime.0 [RFC3418] and snmpTrapOID.0 [RFC3418] respectively. If the OBJECTS clause is present in the invocation of the corresponding NOTIFICATION-TYPE macro, then each corresponding variable, as instantiated by this notification, is copied, in order, to the variable-bindings field. If any additional variables are being included (at the option of the generating SNMP entity), then each is copied to the variable-bindings field.

Upon receipt of an InformRequest-PDU, the receiving SNMP entity determines the size of a message encapsulating a Response-PDU with the same values in its request-id, error-status, error-index and variable-bindings fields as the received InformRequest-PDU. If the determined message size is greater than either a local constraint or the maximum message size of the originator, then an alternate Response-PDU is generated, transmitted to the originator of the InformRequest-PDU, and processing of the InformRequest-PDU terminates immediately thereafter. This alternate Response-PDU is formatted with the same values in its request-id field as the received InformRequest-PDU, with the value of its error-status field set to "tooBig", the value of its error-index field set to zero, and an empty variable-bindings field. This alternate Response-PDU is then encapsulated into a message. If the size of the resultant message is less than or equal to both a local constraint and the maximum message size of the originator, it is transmitted to the originator of the InformRequest-PDU. Otherwise, the snmpSilentDrops [RFC3418] counter is incremented and the resultant message is discarded. Regardless, processing of the InformRequest-PDU terminates.

Otherwise, the receiving SNMP entity:

(1) presents its contents to the appropriate application;

(2) generates a Response-PDU with the same values in its request-id and variable-bindings fields as the received InformRequest-PDU, with the value of its error-status field set to "noError" and the value of its error-index field set to zero; and

(3) transmits the generated Response-PDU to the originator of the InformRequest-PDU.

5. Notice on 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.

6. Acknowledgments

This document is the product of the SNMPv3 Working Group. Some special thanks are in order to the following Working Group members:

Randy Bush
Jeffrey D. Case
Mike Daniele
Rob Frye
Lauren Heintz
Keith McCloghrie
Russ Mundy
David T. Perkins
Randy Presuhn
Aleksey Romanov
Juergen Schoenwaelder
Bert Wijnen

This version of the document, edited by Randy Presuhn, was initially based on the work of a design team whose members were:

Jeffrey D. Case
Keith McCloghrie
David T. Perkins
Randy Presuhn
Juergen Schoenwaelder

The previous versions of this document, edited by Keith McCloghrie, was the result of significant work by four major contributors:

Jeffrey D. Case
Keith McCloghrie
Marshall T. Rose
Steven Waldbusser

Additionally, the contributions of the SNMPv2 Working Group to the previous versions are also acknowledged. In particular, a special thanks is extended for the contributions of:

Alexander I. Alten
Dave Arneson
Uri Blumenthal
Doug Book
Kim Curran
Jim Galvin
Maria Greene
Iain Hanson
Dave Harrington
Nguyen Hien
Jeff Johnson
Michael Kornegay
Deirdre Kostick
David Levi
Daniel Mahoney
Bob Natale
Brian O'Keefe
Andrew Pearson
Dave Perkins
Randy Presuhn
Aleksey Romanov
Shawn Routhier
Jon Saperia
Juergen Schoenwaelder
Bob Stewart

Kaj Tesink
Glenn Waters
Bert Wijnen

7. Security Considerations

The protocol defined in this document by itself does not provide a secure environment. Even if the network itself is secure (for example by using IPSec), there is no control as to who on the secure network is allowed access to management information.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model STD 62, RFC 3414 [RFC3414] and the View-based Access Control Model STD 62, RFC 3415 [RFC3415] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity is properly configured so that:

- only those principals (users) having legitimate rights can access or modify the values of any MIB objects supported by that entity;

- the occurrence of particular events on the entity will be communicated appropriately;

- the entity responds appropriately and with due credence to events and information that have been communicated to it.

8. References

8.1. Normative References

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC2578] 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.

[RFC2579] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC3411] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3412] Case, J., Harrington, D., Presuhn, R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3412, December 2002.

[RFC3413] Levi, D., Meyer, P. and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "The User-Based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3415] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3415, December 2002.

[RFC3417] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for the Simple Network Management Protocol", STD 62, RFC 3417, December 2002.

[RFC3418] Presuhn, R., Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

[ASN1] Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), International Organization for Standardization. International Standard 8824, December 1987.

8.2. Informative References

[FRAG] Kent, C. and J. Mogul, "Fragmentation Considered Harmful," Proceedings, ACM SIGCOMM '87, Stowe, VT, August 1987.

[RFC1155] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", STD 16, RFC 1155, May 1990.

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1212] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[RFC1213] McCloghrie, K. and M. Rose, Editors, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, March 1991.

[RFC1215] Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[RFC1901] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC2576] Frye, R., Levi, D., Routhier, S. and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-Standard Network Management Framework", RFC 2576, March 2000.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC3410] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

9. Changes from RFC 1905

These are the changes from RFC 1905:

- Corrected spelling error in copyright statement;
- Updated copyright date;
- Updated with new editor’s name and contact information;
- Added notice on intellectual property;

- Cosmetic fixes to layout and typography;

- Added table of contents;

- Title changed;

- Updated document headers and footers;

- Deleted the old clause 2.3, entitled "Access to Management Information";

- Changed the way in which request-id was defined, though with the same ultimate syntax and semantics, to avoid coupling with SMI. This does not affect the protocol in any way;

- Replaced the word "exception" with the word "error" in the old clause 4.1. This does not affect the protocol in any way;

- Deleted the first two paragraphs of the old clause 4.2;

- Clarified the maximum number of variable bindings that an implementation must support in a PDU. This does not affect the protocol in any way;

- Replaced occurrences of "SNMPv2 application" with "application";

- Deleted three sentences in old clause 4.2.3 describing the handling of an impossible situation. This does not affect the protocol in any way;

- Clarified the use of the SNMPv2-Trap-Pdu in the old clause 4.2.6. This does not affect the protocol in any way;

- Aligned description of the use of the InformRequest-Pdu in old clause 4.2.7 with the architecture. This does not affect the protocol in any way;

- Updated references;

- Re-wrote introduction clause;

- Replaced manager/agent/SNMPv2 entity terminology with terminology from RFC 2571. This does not affect the protocol in any way;

- Eliminated IMPORTS from the SMI, replaced with equivalent in-line ASN.1. This does not affect the protocol in any way;

- Added notes calling attention to two different manifestations of reaching the end of a table in the table walk examples;

- Added content to security considerations clause;

- Updated ASN.1 comment on use of Report-PDU. This does not affect the protocol in any way;

- Updated acknowledgments section;

- Included information on handling of BITS;

- Deleted spurious comma in ASN.1 definition of PDUs;

- Added abstract;

- Made handling of additional variable bindings in informs consistent with that for traps. This was a correction of an editorial oversight, and reflects implementation practice;

- Added reference to RFC 2914.

10. Editor's Address

Randy Presuhn
BMC Software, Inc.
2141 North First Street
San Jose, CA 95131
USA

Phone: +1 408 546 1006
EMail: randy_presuhn@bmc.com

11. Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.