                              V2ToV1
                   Mapping SNMPv2 onto SNMPv1
                   within a bi-lingual SNMP agent

Status of this Memo

   This memo provides information for the Internet community. This memo
   does not specify an Internet standard of any kind. Distribution of
   this memo is unlimited.

Abstract

   The goal of this memo is to document a common way of mapping an
   SNMPv2 response into an SNMPv1 response within a bi-lingual SNMP
   agent (one that supports both SNMPv1 and SNMPv2).

Table of Contents

1.0 Introduction

   We now have the SNMPv1 protocol (RFC1157 [1]) as a full standard and
   the SNMPv2 protocol (RFC1905 [1]) as a DRAFT standard. It can be
   expected that many agent implementations will support both SNMPv1 and
   SNMPv2 requests coming from SNMP management entities. In many cases
   the underlying instrumentation will be implemented using the new
   SNMPv2 SMI and SNMPv2 protocol. The SNMP engine then gets the task
   to ensure that any SNMPv2 response data coming from such SNMPv2
   compliant instrumentation gets converted to a proper SNMPv1 response
   if the original request came in as an SNMPv1 request. The SNMP
   engine should also deal with mapping SNMPv2 traps which are generated
   by an application or by the SNMPv2 compliant instrumentation into
   SNMPv1 traps if the agent has been configured to send traps to an
   SNMPv1 manager.

   It seems beneficial if all such agents do this mapping in the same
   way. This document describes such a mapping based on discussions and
   perceived consensus on the various mailing lists. The authors of
   this document have also compared their own implementations of these
   mappings. They had a few minor differences and decided to make their
   implementation behave the same and document this mapping so others
   can benefit from it.

   We recommend that all agents implement this same mapping.

   Note that the mapping described in this document should also be
   followed by SNMP proxies that provide a mapping between SNMPv1
   management applications and SNMPv2 agents.

2.0 Mapping SNMPv2 into SNMPv1

   These are the type of mappings that we need:

      o   Mapping of the SNMPv2 error-status into SNMPv1 error-status

      o   Mapping of the SNMPv2 exceptions into SNMPv1 error-status

      o   Skipping object instances that have a non-SNMPv1 Syntax
           (specifically Counter64)

      o   Mapping of SNMPv2 traps into SNMPv1 traps

2.1 Mapping SNMPv2 error-status into SNMPv1 error-status

   With the new SNMPv2 protocol (RFC1905 [1]) we get a set of error-
   status values that return the cause of an error in much more detail.
   But an SNMPv1 manager does not understand such error-status values.

   So, when the instrumentation code returns response data and uses an
   SNMPv2 error-status to report on the success or failure of the
   requested operation and if the original SNMP request is an SNMPv1
   request, then we must map such an error-status into an SNMPv1 error-
   status when composing the SNMP response PDU.

   The SNMPv2 error status is mapped to an SNMPv1 error-status using
   this table:

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

2.2 Mapping SNMPv2 exceptions into SNMPv1

   In SNMPv2 we have so called exception values. These will allow an
   SNMPv2 response PDU to return as much management information as
   possible, even if one or more of the requested variables do not
   exist. SNMPv1 does not support exception values, and thus does not
   return the value of management information when any error occurs.

   When multiple variables do not exist, an SNMPv1 agent can return only
   a single error and index of a single variable. The agent determines
   by its implementation strategy which variable to identify as the

cause of the error via the value of the error-index field. Thus, an
SNMPv1 manager may make no assumption on the validity of the other
variables in the request.

So, when an SNMPv1 request is received, we must check the varBinds
returned from SNMPv2 compliant instrumentation for exception values,
and convert these exception values into SNMPv1 error codes.

The type of exception we can get back and the action we must take
depends on the SNMP operation that is requested.

  o   For SNMP GET requests we can get back noSuchObject and
       noSuchInstance.

  o   For SNMP GETNEXT requests we can get back endOfMibView.

  o   For SNMP SET requests we cannot get back any exceptions.

  o   For SNMP GETBULK requests we can get back endOfMibView, but
       such a request should only come in as an SNMPv2 request, so we
       do not have to worry about any mapping onto SNMPv1. If a
       GETBULK comes in as an SNMPv1 request, it is treated as an
       error and the packet is dropped.

2.3 Mapping noSuchObject and noSuchInstance

   A noSuchObject or noSuchInstance exception generated by SNMPv2
   compliant instrumentation 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 should be set to noSuchName. Also, the error-index
   field is set to the index of the varBind for which an exception
   occurred, and the varBind list from the original request is returned
   with the response PDU.

   Note that when the response contains multiple exceptions, that the
   agent may pick any one to be returned.

2.4 Mapping endOfMibView

   When SNMPv2 compliant instrumentation returns a varBind containing an
   endOfMibView exception in response to a GETNEXT request, 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 should be set to noSuchName. Also, the error-
   index field is set to the index of the varBind for which an exception
   occurred, and the varBind list from the original request is returned
   with the response PDU.

   Note that when the response contains multiple exceptions, that the
   agent may pick any one to be returned.

2.5 Mapping SNMPv2 SMI into SNMPv1

   The SNMPv2 SMI (RFC1902 [2]) defines basically one new syntax that is
   problematic for SNMPv1 managers. That is the syntax Counter64. All
   the others can be handled by SNMPv1 managers.

   The net impact on bi-lingual agents is that they should make sure
   that they never return a varBind with a Counter64 value to an SNMPv1
   manager.

   The best accepted practice is to consider such object instances
   implicitly excluded from the view. So:

      o   On an SNMPv1 GET request, we return an error-status of
           noSuchName and the error-index is set to the varBind that
           causes this error.

      o   On an SNMPv1 GETNEXT request, we skip the object instance and
           fetch the next object instance that follows the one with a
           syntax of Counter64.

      o   Any SET request that has a varBind with a Counter64 value must
           have come from a SNMPv2 manager, and so it should not cause a
           problem. If we do receive a Counter64 value in an SNMPv1 SET
           packet, it should result in an ASN.1 parse error since
           Counter64 is not valid in the SNMPv1 protocol. When an ASN.1
           parse error occurs, the counter snmpInASNParseErrs is
           incremented and no response is returned.

      o   The GETBULK is an SNMPv2 operation, so it should never come
           from an SNMPv1 manager. If we do receive a GETBULK PDU from in
           an SNMPv1 packet, then we consider it an invalid PDU-type and
           we drop the packet.

3.0 Processing SNMPv1 requests

   This sections contains a step by step description of how to handle
   SNMPv1 requests in an agent where the underlying instrumentation code
   is SNMPv2 compliant.

3.1 Processing an SNMPv1 GET request

   First, the request is converted into a call to the underlying
   instrumentation. This is implementation specific.

   When such instrumentation returns response data using SNMPv2 syntax
   and error-status values, then:

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

         a. The error status is translated to an SNMPv1 error-status
             using the table from 2.1, "Mapping SNMPv2 error-status into
             SNMPv1 error-status" on page 2

         b. The error-index is set to the position (in the original
             request) of the varBind that caused the error-status.

         c. The varBindList of the response PDU is made exactly the
             same as the varBindList that was received in the original
             request.

    2. If the error-status is noError, then find any varBind that
        contains an SNMPv2 exception (noSuchObject or noSuchInstance)
        or an SNMPv2 syntax that is unknown to SNMPv1 (Counter64).
        (Note that if there are more than one, the agent may choose any
        such varBind.) If there are any such varBinds, then for the
        one chosen:

         a. Set the error-status to noSuchName

         b. Set the error-index to the position (in the varBindList of
             the original request) of the varBind that returned such an
             SNMPv2 exception or syntax.

         c. Make the varBindList of the response PDU exactly the same
             as the varBindList that was received in the original
             request.

   3. If there are no such varBinds, then:

       a. Set the error-status to noError

       b. Set the error-index to zero

       c. Compose the varBindList of the response, using the data as
           it is returned by the instrumentation code.

3.2 Processing an SNMPv1 GETNEXT request

   First, the request is converted into a call to the underlying
   instrumentation. This is implementation specific. There may be
   repetitive calls to (possibly different pieces of) instrumentation
   code to try to find the first object which lexicographically follows
   each of the objects in the request. Again, this is implementation
   specific.

   When the instrumentation finally returns response data using SNMPv2
   syntax and error-status values, then:

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

       a. The error status is translated to an SNMPv1 error-status
           using the table from 2.1, "Mapping SNMPv2 error-status into
           SNMPv1 error-status" on page 2

       b. The error-index is set to the position (in the original
           request) of the varBind that caused the error-status.

       c. The varBindList of the response PDU is made exactly the
           same as the varBindList that was received in the original
           request.

   2. If the error-status is noError, then:

       a. If there are any varBinds containing an SNMPv2 syntax of
           Counter64, then consider these varBinds to be not in view
           and repeat the call to the instrumentation code as often as
           needed till a value other than Counter64 is returned.

       b. Find any varBind that contains an SNMPv2 exception
           endOfMibView. (Note that if there are more than one, the
           agent may choose any such varBind.) If there are any such
           varBinds, then for the one chosen:

           1) Set the error-status to noSuchName

        2) Set the error-index to the position (in the varBindList
            of the original request) of the varBind that returned
            such an SNMPv2 exception.

        3) Make the varBindList of the response PDU exactly the
            same as the varBindList that was received in the
            original request.

    c. If there are no such varBinds, then:

        1) Set the error-status to noError

        2) Set the error-index to zero

        3) Compose the varBindList of the response, using the data
            as it is returned by the instrumentation code.

3.3 Processing an outgoing SNMPv2 TRAP

   If SNMPv2 compliant instrumentation presents an SNMPv2 trap to the
   SNMP engine and such a trap passes all regular checking and then is
   to be sent to an SNMPv1 destination, then the following steps must be
   followed to convert such a trap to an SNMPv1 trap. This is basically
   the reverse of the SNMPv1 to SNMPv2 mapping as described in RFC1908
   [3].

   1. If any of the varBinds in the varBindList has an SNMPv2 syntax
       of Counter64, then such varBinds are implicitly considered to
       be not in view, and so they are removed from the varBindList to
       be sent with the SNMPv1 trap.

   2. The 3 special varBinds in the varBindList of an SNMPv2 trap
       (sysUpTime.0 (TimeTicks), snmpTrapOID.0 (OBJECT IDENTIFIER) and
       optionally snmpTrapEnterprise.0 (OBJECT IDENTIFIER)) are
       removed from the varBindList to be sent with the SNMPv1 trap.
       These 2 (or 3) varBinds are used to decide how to set other
       fields in the SNMPv1 trap PDU as follows:

       a. The value of sysUpTime.0 is copied into the timestamp field
           of the SNMPv1 trap.

      b. If the snmpTrapOID.0 value is one of the standard traps the
          specific-trap field is set to zero and the generic trap
          field is set according to this mapping:

```
value of snmpTrapOID.0                       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
```

          The enterprise field is set to the value of
          snmpTrapEnterprise.0 if this varBind is present, otherwise
          it is set to the value snmpTraps as defined in RFC1907 [4].

      c. If the snmpTrapOID.0 value is not one of the standard
          traps, then the generic-trap field is set to 6 and the
          specific-trap field is set to the last subid of the
          snmpTrapOID.0 value.

         o   If the next to last subid of snmpTrapOID.0 is zero,
            then the enterprise field is set to snmpTrapOID.0 value
            and the last 2 subids are truncated from that value.
         o   If the next to last subid of snmpTrapOID.0 is not zero,
            then the enterprise field is set to snmpTrapOID.0 value
            and the last 1 subid is truncated from that value.

          In any event, the snmpTrapEnterprise.0 varBind (if present)
          is ignored in this case.

   3. The agent-addr field is set with the appropriate address of the
      the sending SNMP entity, which is the IP address of the sending
      entity of the trap goes out over UDP; otherwise the agent-addr
      field is set to address 0.0.0.0.

4.0 Acknowledgements

   The authors wish to thank the contributions of the SNMPv2 Working
   Group in general. Special thanks for their detailed review and
   comments goes to these individuals:

      Mike Daniele (DEC)
      Dave Harrington (Cabletron)
      Brian O'Keefe (Hewlett Packard)
      Keith McCloghrie (Cisco Systems)
      Dave Perkins (independent)
      Shawn Routhier (Epilogue)
      Juergen Schoenwaelder (University of Twente)

5.0 References

   [1] Jeffrey D. Case, Mark Fedor, Martin Lee Schoffstall and James
        R. Davin, Simple Network Management Protocol (SNMP), SNMP
        Research, Performance Systems International, MIT Laboratory
        for Computer Science, RFC 1157, May 1990.

   [2] Jeffrey D. Case, Keith McCloghrie, Marshall T. Rose and Steven
        Waldbusser, Structure of Managment Information for Version 2
        of the Simple Network Management Protocol (SNMPv2), SNMP
        Research Inc, Cisco Systems Inc, Dover Beach Consulting Inc,
        International Network Services, RFC1902, January 1996.

   [3] Jeffrey D. Case, Keith McCloghrie, Marshall T. Rose and Steven
        Waldbusser, Coexistence between Version 1 and Version 2 of the
        Internet-standard Network Management Framework, SNMP Research
        Inc, Cisco Systems Inc, Dover Beach Consulting   Inc,
        International Network Services, RFC1908, January 1996.

   [4] Jeffrey D. Case, Keith McCloghrie, Marshall T. Rose and Steven
        Waldbusser, Management Information Base for Version 2 of the
        Simple Network Management Protocol (SNMPv2), SNMP Research
        Inc, Cisco Systems Inc, Dover Beach Consulting Inc,
        International Network Services, RFC1907, January 1996.

6.0 Security Considerations

   Security considerations are not discussed in this memo.

7.0 Authors' Addresses

        Bert Wijnen
        IBM International Operations
        Watsonweg 2
        1423 ND Uithoorn
        The Netherlands

        Phone: +31-079-322-8316
        E-mail: wijnen@vnet.ibm.com

        David Levi
        SNMP Research, Inc
        3001 Kimberlin Heights Rd.
        Knoxville, TN 37920-9716
        USA

        Phone: +1-615-573-1434
        E-mail: levi@snmp.com

APPENDIX A.   Background Information

   Here follows some reasoning as to why some choices were made.

   A.1   Mapping of error-status values

   The mapping of SNMPv2 error-status values to SNMPv1 error-status
   values is based on the common interpretation of how an SNMPv1 entity
   should create an error-status value based on the elements of
   procedure defined in RFC1157 [1].

   There was a suggestion to map wrongEncoding into genErr, because it
   could be caused by an ASN.1 parsing error. Such maybe true, but in
   most cases when we detect the ASN.1 parsing error, we do not yet know
   about the PDU data yet. Most people who responded to our queries
   have implemented the mapping to a badValue. So we "agreed" on the
   mapping to badValue.

   A.2   SNMPv1 Traps without Counter64 varBinds.

   RFC1448 says that if one of the objects in the varBindList is not
   included in the view, then the trap is NOT sent. Current SNMPv2u and
   SNMPv2* documents make the same statement. However, the "rough
   consensus" is that it is better to send partial information than no
   information at all. Besides:

     o   RFC1448 does not allow for a TRAP to be sent with the varBinds
         that are not included in the view removed, so it is an all or
         nothing decision.

     o   We do NOT include the Counter64 varBinds... so the "not in
         view" varBinds are not sent to the trap destination.

     o   The Counter64 objects are "implicit" not in view. If any
         objects are explicit not in view, then this is checked before
         we do the conversion from an SNMPv2 trap to an SNMPv1 trap, and
         so the trap is not sent at all.