

Network Working Group
Request for Comments: 2273
Obsoletes: 2263
Category: Standards Track

D. Levi
SNMP Research, Inc.
P. Meyer
Secure Computing Corporation
B. Stewart
Cisco Systems
January 1998

SNMPv3 Applications

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 (1998). All Rights Reserved.

IANA Note

Due to a clerical error in the assignment of the snmpModules in this memo, this RFC provides the corrected number assignments for this protocol. This memo obsoletes RFC 2263.

Abstract

This memo describes five types of SNMP applications which make use of an SNMP engine as described in [RFC2271]. The types of application described are Command Generators, Command Responders, Notification Originators, Notification Receivers, and Proxy Forwarders.

This memo also defines MIB modules for specifying targets of management operations, for notification filtering, and for proxy forwarding.

Table of Contents

1 Overview	2
1.1 Command Generator Applications	3
1.2 Command Responder Applications	3
1.3 Notification Originator Applications	3
1.4 Notification Receiver Applications	3
1.5 Proxy Forwarder Applications	3
2 Management Targets	5

3 Elements Of Procedure	6
3.1 Command Generator Applications	6
3.2 Command Responder Applications	8
3.3 Notification Originator Applications	13
3.4 Notification Receiver Applications	16
3.5 Proxy Forwarder Applications	18
3.5.1 Request Forwarding	19
3.5.1.1 Processing an Incoming Request	19
3.5.1.2 Processing an Incoming Response	22
3.5.1.3 Processing an Incoming Report Indication	23
3.5.2 Notification Forwarding	24
4 The Structure of the MIB Modules	27
4.1 The Management Target MIB Module	27
4.1.1 Tag Lists	28
4.1.2 Definitions	28
4.2 The Notification MIB Module	41
4.2.1 Definitions	42
4.3 The Proxy MIB Module	53
4.3.1 Definitions	53
5 Identification of Management Targets in Notification Originators	59
6 Notification Filtering	60
7 Management Target Translation in Proxy Forwarder Applications	61
7.1 Management Target Translation for Request Forwarding	61
7.2 Management Target Translation for Notification Forwarding	62
8 Intellectual Property	63
9 Acknowledgments	64
10 Security Considerations	65
11 References	65
12 Editors' Address	67
A. Trap Configuration Example	68
B. Full Copyright Statement	70

1. Overview

This document describes five types of SNMP applications:

- Applications which initiate SNMP Get, GetNext, GetBulk, and/or Set requests, called 'command generators.'
- Applications which respond to SNMP Get, GetNext, GetBulk, and/or Set requests, called 'command responders.'
- Applications which generate notifications, called 'notification originators.'

- Applications which receive notifications, called 'notification receivers.'
- Applications which forward SNMP Get, GetNext, GetBulk, and/or Set requests or notifications, called 'proxy forwarder.'

Note that there are no restrictions on which types of applications may be associated with a particular SNMP engine. For example, a single SNMP engine may, in fact, be associated with both command generator and command responder applications.

1.1. Command Generator Applications

A command generator application initiates SNMP Get, GetNext, GetBulk, and/or Set requests, as well as processing the response to a request which it generated.

1.2. Command Responder Applications

A command responder application receives SNMP Get, GetNext, GetBulk, and/or Set requests destined for the local system as indicated by the fact that the contextEngineID in the received request is equal to that of the local engine through which the request was received. The command responder application will perform the appropriate protocol operation, using access control, and will generate a response message to be sent to the request's originator.

1.3. Notification Originator Applications

A notification originator application conceptually monitors a system for particular events or conditions, and generates Trap and/or Inform messages based on these events or conditions. A notification originator must have a mechanism for determining where to send messages, and what SNMP version and security parameters to use when sending messages. A mechanism and MIB module for this purpose is provided in this document.

1.4. Notification Receiver Applications

A notification receiver application listens for notification messages, and generates response messages when a message containing an Inform PDU is received.

1.5. Proxy Forwarder Applications

A proxy forwarder application forwards SNMP messages. Note that implementation of a proxy forwarder application is optional. The sections describing proxy (4.5, 5.3, and 8) may be skipped for

implementations that do not include a proxy forwarder application.

The term "proxy" has historically been used very loosely, with multiple different meanings. These different meanings include (among others):

- (1) the forwarding of SNMP requests to other SNMP entities without regard for what managed object types are being accessed; for example, in order to forward an SNMP request from one transport domain to another, or to translate SNMP requests of one version into SNMP requests of another version;
- (2) the translation of SNMP requests into operations of some non-SNMP management protocol; and
- (3) support for aggregated managed objects where the value of one managed object instance depends upon the values of multiple other (remote) items of management information.

Each of these scenarios can be advantageous; for example, support for aggregation of management information can significantly reduce the bandwidth requirements of large-scale management activities. However, using a single term to cover multiple different scenarios causes confusion.

To avoid such confusion, this document uses the term "proxy" with a much more tightly defined meaning. The term "proxy" is used in this document to refer to a proxy forwarder application which forwards either SNMP requests, notifications, and responses without regard for what managed objects are contained within requests or notifications. This definition is most closely related to the first definition above. Note, however, that in the SNMP architecture [RFC2271], a proxy forwarder is actually an application, and need not be associated with what is traditionally thought of as an SNMP agent.

Specifically, the distinction between a traditional SNMP agent and a proxy forwarder application is simple:

- a proxy forwarder application forwards requests and/or notifications to other SNMP engines according to the context, and irrespective of the specific managed object types being accessed, and forwards the response to such previously forwarded messages back to the SNMP engine from which the original message was received;
- in contrast, the command responder application that is part of what is traditionally thought of as an SNMP agent, and which processes SNMP requests according to the (names of the)

individual managed object types and instances being accessed, is NOT a proxy forwarder application from the perspective of this document.

Thus, when a proxy forwarder application forwards a request or notification for a particular contextEngineID / contextName pair, not only is the information on how to forward the request specifically associated with that context, but the proxy forwarder application has no need of a detailed definition of a MIB view (since the proxy forwarder application forwards the request irrespective of the managed object types).

In contrast, a command responder application must have the detailed definition of the MIB view, and even if it needs to issue requests to other entities, via SNMP or otherwise, that need is dependent on the individual managed object instances being accessed (i.e., not only on the context).

Note that it is a design goal of a proxy forwarder application to act as an intermediary between the endpoints of a transaction. In particular, when forwarding Inform requests, the associated response is forwarded when it is received from the target to which the Inform request was forwarded, rather than generating a response immediately when an Inform request is received.

2. Management Targets

Some types of applications (notification generators and proxy forwarders in particular) require a mechanism for determining where and how to send generated messages. This document provides a mechanism and MIB module for this purpose. The set of information that describes where and how to send a message is called a 'Management Target', and consists of two kinds of information:

- Destination information, consisting of a transport domain and a transport address. This is also termed a transport endpoint.
- SNMP parameters, consisting of message processing model, security model, security level, and security name information.

The SNMP-TARGET-MIB module described later in this document contains one table for each of these types of information. There can be a many-to-many relationship in the MIB between these two types of information. That is, there may be multiple transport endpoints associated with a particular set of SNMP parameters, or a particular transport endpoint may be associated with several sets of SNMP parameters.

3. Elements Of Procedure

The following sections describe the procedures followed by each type of application when generating messages for transmission or when processing received messages. Applications communicate with the Dispatcher using the abstract service interfaces defined in [RFC2271].

3.1. Command Generator Applications

A command generator initiates an SNMP request by calling the Dispatcher using the following abstract service interface:

```
statusInformation =          -- sendPduHandle if success
                             -- errorIndication if failure
sendPdu(
  IN  transportDomain         -- transport domain to be used
  IN  transportAddress        -- destination network address
  IN  messageProcessingModel  -- typically, SNMP version
  IN  securityModel           -- Security Model to use
  IN  securityName            -- on behalf of this principal
  IN  securityLevel           -- Level of Security requested
  IN  contextEngineID        -- data from/at this entity
  IN  contextName             -- data from/in this context
  IN  pduVersion              -- the version of the PDU
  IN  PDU                     -- SNMP Protocol Data Unit
  IN  expectResponse          -- TRUE or FALSE
)
```

Where:

- The transportDomain is that of the destination of the message.
- The transportAddress is that of the destination of the message.
- The messageProcessingModel indicates which Message Processing Model the application wishes to use.
- The securityModel is the security model that the application wishes to use.
- The securityName is the security model independent name for the principal on whose behalf the application wishes the message is to be generated.
- The securityLevel is the security level that the application wishes to use.

- The contextEngineID is provided by the command generator if it wishes to explicitly specify the location of the management information it is requesting.
- The contextName is provided by the command generator if it wishes to explicitly specify the local context name for the management information it is requesting.
- The pduVersion indicates the version of the PDU to be sent.
- The PDU is a value constructed by the command generator containing the management operation that the command generator wishes to perform.
- The expectResponse argument indicates that a response is expected.

The result of the sendPdu interface indicates whether the PDU was successfully sent. If it was successfully sent, the returned value will be a sendPduHandle. The command generator should store the sendPduHandle so that it can correlate a response to the original request.

The Dispatcher is responsible for delivering the response to a particular request to the correct command generator application. The abstract service interface used is:

```

processResponsePdu(
    IN    messageProcessingModel  -- process Response PDU
    IN    securityModel           -- typically, SNMP version
    IN    securityName            -- Security Model in use
    IN    securityLevel           -- on behalf of this principal
    IN    contextEngineID         -- Level of Security
    IN    contextName             -- data from/at this SNMP entity
    IN    pduVersion              -- data from/in this context
    IN    PDU                     -- the version of the PDU
    IN    statusInformation        -- SNMP Protocol Data Unit
    IN    sendPduHandle           -- success or errorIndication
    )                             -- handle from sendPDU

```

Where:

- The messageProcessingModel is the value from the received response.
- The securityModel is the value from the received response.
- The securityName is the value from the received response.

- The securityLevel is the value from the received response.
- The contextEngineID is the value from the received response.
- The contextName is the value from the received response.
- The pduVersion indicates the version of the PDU in the received response.
- The PDU is the value from the received response.
- The statusInformation indicates success or failure in receiving the response.
- The sendPduHandle is the value returned by the sendPdu call which generated the original request to which this is a response.

The procedure when a command generator receives a message is as follows:

- (1) If the received values of messageProcessingModel, securityModel, securityName, contextEngineID, contextName, and pduVersion are not all equal to the values used in the original request, the response is discarded.
- (2) The operation type, request-id, error-status, error-index, and variable-bindings are extracted from the PDU and saved. If the request-id is not equal to the value used in the original request, the response is discarded.
- (3) At this point, it is up to the application to take an appropriate action. The specific action is implementation dependent. If the statusInformation indicates that the request failed, an appropriate action might be to attempt to transmit the request again, or to notify the person operating the application that a failure occurred.

3.2. Command Responder Applications

Before a command responder application can process messages, it must first associate itself with an SNMP engine. The abstract service interface used for this purpose is:


```

statusInformation =      -- success or errorIndication
  registerContextEngineID(
    IN  contextEngineID  -- take responsibility for this one
    IN  pduType          -- the pduType(s) to be registered
  )

```

Where:

- The statusInformation indicates success or failure of the registration attempt.
- The contextEngineID is equal to the snmpEngineID of the SNMP engine with which the command responder is registering.
- The pduType indicates a Get, GetNext, GetBulk, or Set pdu.

Note that if another command responder application is already registered with an SNMP engine, any further attempts to register with the same contextEngineID and pduType will be denied. This implies that separate command responder applications could register separately for the various pdu types. However, in practice this is undesirable, and only a single command responder application should be registered with an SNMP engine at any given time.

A command responder application can disassociate with an SNMP engine using the following abstract service interface:

```

unregisterContextEngineID(
  IN  contextEngineID  -- give up responsibility for this one
  IN  pduType          -- the pduType(s) to be unregistered
)

```

Where:

- The contextEngineID is equal to the snmpEngineID of the SNMP engine with which the command responder is cancelling the registration.
- The pduType indicates a Get, GetNext, GetBulk, or Set pdu.

Once the command responder has registered with the SNMP engine, it waits to receive SNMP messages. The abstract service interface used for receiving messages is:

```

processPdu(
  IN  messageProcessingModel  -- process Request/Notification PDU
  IN  securityModel          -- typically, SNMP version
  IN  securityName           -- Security Model in use
                                -- on behalf of this principal
)

```

```
IN  securityLevel          -- Level of Security
IN  contextEngineID        -- data from/at this SNMP entity
IN  contextName            -- data from/in this context
IN  pduVersion             -- the version of the PDU
IN  PDU                    -- SNMP Protocol Data Unit
IN  maxSizeResponseScopedPDU -- maximum size of the Response PDU
IN  stateReference         -- reference to state information
IN  )                     -- needed when sending a response
```

Where:

- The messageProcessingModel indicates which Message Processing Model received and processed the message.
- The securityModel is the value from the received message.
- The securityName is the value from the received message.
- The securityLevel is the value from the received message.
- The contextEngineID is the value from the received message.
- The contextName is the value from the received message.
- The pduVersion indicates the version of the PDU in the received message.
- The PDU is the value from the received message.
- The maxSizeResponseScopedPDU is the maximum allowable size of a ScopedPDU containing a Response PDU (based on the maximum message size that the originator of the message can accept).
- The stateReference is a value which references cached information about each received request message. This value must be returned to the Dispatcher in order to generate a response.

The procedure when a message is received is as follows.

- (1) The operation type is determined from the ASN.1 tag value associated with the PDU parameter. The operation type should always be one of the types previously registered by the application.
- (2) The request-id is extracted from the PDU and saved.

- (3) If the SNMPv2 operation type is GetBulk, the non-repeaters and max-repetitions values are extracted from the PDU and saved.
- (4) The variable-bindings are extracted from the PDU and saved.
- (5) The management operation represented by the SNMPv2 operation type is performed with respect to the relevant MIB view within the context named by the contextName, according to the procedures set forth in [RFC1905]. The relevant MIB view is determined by the securityLevel, securityModel, contextName, securityName, and SNMPv2 operation type. To determine whether a particular object instance is within the relevant MIB view, the following abstract service interface is called:

```

statusInformation =      -- success or errorIndication
  isAccessAllowed(
    IN  securityModel      -- Security Model in use
    IN  securityName       -- principal who wants to access
    IN  securityLevel      -- Level of Security
    IN  viewType           -- read, write, or notify view
    IN  contextName        -- context containing variableName
    IN  variableName       -- OID for the managed object
  )

```

Where:

- The securityModel is the value from the received message.
- The securityName is the value from the received message.
- The securityLevel is the value from the received message.
- The viewType indicates whether the PDU type is a read or write operation.
- The contextName is the value from the received message.
- The variableName is the object instance of the variable for which access rights are to be checked.

Normally, the result of the management operation will be a new PDU value, and processing will continue in step (6) below. However, at any time during the processing of the management operation:

- If the isAccessAllowed ASI returns a noSuchView, noAccessEntry, or noGroupName error, processing of the management operation is halted, a PDU value is constructed using the values from the originally received PDU, but

replacing the error_status with an authorizationError code, and error_index value of 0, and control is passed to step (6) below.

- If the isAccessAllowed ASI returns an otherError, processing of the management operation is halted, a different PDU value is constructed using the values from the originally received PDU, but replacing the error_status with a genError code, and control is passed to step (6) below.
- If the isAccessAllowed ASI returns a noSuchContext error, processing of the management operation is halted, no result PDU is generated, the snmpUnknownContexts counter is incremented, and control is passed to step (6) below.
- If the context named by the contextName parameter is unavailable, processing of the management operation is halted, no result PDU is generated, the snmpUnavailableContexts counter is incremented, and control is passed to step (6) below.

(6) The Dispatcher is called to generate a response or report message. The abstract service interface is:

```
returnResponsePdu(
  IN  messageProcessingModel  -- typically, SNMP version
  IN  securityModel           -- Security Model in use
  IN  securityName            -- on behalf of this principal
  IN  securityLevel           -- same as on incoming request
  IN  contextEngineID         -- data from/at this SNMP entity
  IN  contextName             -- data from/in this context
  IN  pduVersion              -- the version of the PDU
  IN  PDU                    -- SNMP Protocol Data Unit
  IN  maxSizeResponseScopedPDU -- maximum size of the Response PDU
  IN  stateReference          -- reference to state information
                                -- as presented with the request
  IN  statusInformation        -- success or errorIndication
  )                           -- error counter OID/value if error
```

Where:

- The messageProcessingModel is the value from the processPdu call.
- The securityModel is the value from the processPdu call.
- The securityName is the value from the processPdu call.

- The securityLevel is the value from the processPdu call.
- The contextEngineID is the value from the processPdu call.
- The contextName is the value from the processPdu call.
- The pduVersion indicates the version of the PDU to be returned. If no result PDU was generated, the pduVersion is an undefined value.
- The PDU is the result generated in step (5) above. If no result PDU was generated, the PDU is an undefined value.
- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.
- The stateReference is the value from the processPdu call.
- The statusInformation either contains an indication that no error occurred and that a response should be generated, or contains an indication that an error occurred along with the OID and counter value of the appropriate error counter object.

Note that a command responder application should always call the returnResponsePdu abstract service interface, even in the event of an error such as a resource allocation error. In the event of such an error, the PDU value passed to returnResponsePdu should contain appropriate values for errorStatus and errorIndex.

3.3. Notification Originator Applications

A notification originator application generates SNMP notification messages. A notification message may, for example, contain an SNMPv2-Trap PDU or an Inform PDU. However, a particular implementation is not required to be capable of generating both types of messages.

Notification originator applications require a mechanism for identifying the management targets to which notifications should be sent. The particular mechanism used is implementation dependent. However, if an implementation makes the configuration of management targets SNMP manageable, it MUST use the SNMP-TARGET-MIB module described in this document.

When a notification originator wishes to generate a notification, it must first determine in which context the information to be conveyed in the notification exists, i.e., it must determine the contextEngineID and contextName. It must then determine the set of

management targets to which the notification should be sent. The application must also determine, for each management target, whether the notification message should contain an SNMPv2-Trap PDU or Inform PDU, and if it is to contain an Inform PDU, the number of retries and retransmission algorithm.

The mechanism by which a notification originator determines this information is implementation dependent. Once the application has determined this information, the following procedure is performed for each management target:

- (1) Any appropriate filtering mechanisms are applied to determine whether the notification should be sent to the management target. If such filtering mechanisms determine that the notification should not be sent, processing continues with the next management target. Otherwise,
- (2) The appropriate set of variable-bindings is retrieved from local MIB instrumentation within the relevant MIB view. The relevant MIB view is determined by the securityLevel, securityModel, contextName, and securityName of the management target. To determine whether a particular object instance is within the relevant MIB view, the isAccessAllowed abstract service interface is used, in the same manner as described in the preceding section. If the statusInformation returned by isAccessAllowed does not indicate accessAllowed, the notification is not sent to the management target.
- (3) A PDU is constructed using a locally unique request-id value, an operation type of SNMPv2-Trap or Inform, an error-status and error-index value of 0, and the variable-bindings supplied previously in step (2).
- (4) If the notification contains an SNMPv2-Trap PDU, the Dispatcher is called using the following abstract service interface:

```

statusInformation =          -- sendPduHandle if success
                             -- errorIndication if failure
    sendPdu(
        IN  transportDomain    -- transport domain to be used
        IN  transportAddress   -- destination network address
        IN  messageProcessingModel -- typically, SNMP version
        IN  securityModel      -- Security Model to use
        IN  securityName       -- on behalf of this principal
        IN  securityLevel      -- Level of Security requested
        IN  contextEngineID    -- data from/at this entity
        IN  contextName        -- data from/in this context
        IN  pduVersion         -- the version of the PDU

```

```
IN    PDU                                -- SNMP Protocol Data Unit
IN    expectResponse                     -- TRUE or FALSE
)
```

Where:

- The transportDomain is that of the management target.
- The transportAddress is that of the management target.
- The messageProcessingModel is that of the management target.
- The securityModel is that of the management target.
- The securityName is that of the management target.
- The securityLevel is that of the management target.
- The contextEngineID is the value originally determined for the notification.
- The contextName is the value originally determined for the notification.
- The pduVersion is the version of the PDU to be sent.
- The PDU is the value constructed in step (3) above.
- The expectResponse argument indicates that no response is expected.

Otherwise,

- (5) If the notification contains an Inform PDU, then:
- a) The Dispatcher is called using the sendPdu abstract service interface as described in step (4) above, except that the expectResponse argument indicates that a response is expected.
 - b) The application caches information about the management target.
 - c) If a response is received within an appropriate time interval from the transport endpoint of the management target, the notification is considered acknowledged and the cached information is deleted. Otherwise,

- d) If a response is not received within an appropriate time period, or if a report indication is received, information about the management target is retrieved from the cache, and steps a) through d) are repeated. The number of times these steps are repeated is equal to the previously determined retry count. If this retry count is exceeded, the acknowledgement of the notification is considered to have failed, and processing of the notification for this management target is halted.

Responses to Inform PDU notifications will be received via the processResponsePDU abstract service interface.

3.4. Notification Receiver Applications

Notification receiver applications receive SNMP Notification messages from the Dispatcher. Before any messages can be received, the notification receiver must register with the Dispatcher using the registerContextEngineID abstract service interface. The parameters used are:

- The contextEngineID is an undefined 'wildcard' value. Notifications are delivered to a registered notification receiver regardless of the contextEngineID contained in the notification message.
- The pduType indicates the type of notifications that the application wishes to receive (for example, SNMPv2-Trap PDUs or Inform PDUs).

Once the notification receiver has registered with the Dispatcher, messages are received using the processPdu abstract service interface. Parameters are:

- The messageProcessingModel indicates which Message Processing Model received and processed the message.
- The securityModel is the value from the received message.
- The securityName is the value from the received message.
- The securityLevel is the value from the received message.
- The contextEngineID is the value from the received message.
- The contextName is the value from the received message.

- The pduVersion indicates the version of the PDU in the received message.
- The PDU is the value from the received message.
- The maxSizeResponseScopedPDU is the maximum allowable size of a ScopedPDU containing a Response PDU (based on the maximum message size that the originator of the message can accept).
- If the message contains an SNMPv2-Trap PDU, the stateReference is undefined and unused. Otherwise, the stateReference is a value which references cached information about the notification. This value must be returned to the Dispatcher in order to generate a response.

When an SNMPv2-Trap PDU is delivered to a notification receiver application, it first extracts the SNMP operation type, request-id, error-status, error-index, and variable-bindings from the PDU. After this, processing depends on the particular implementation.

When an Inform PDU is received, the notification receiver application follows the following procedure:

- (1) The SNMPv2 operation type, request-id, error-status, error-index, and variable-bindings are extracted from the PDU.
- (2) A Response PDU is constructed using the extracted request-id and variable-bindings, and with error-status and error-index both set to 0.
- (3) The Dispatcher is called to generate a response message using the returnResponsePdu abstract service interface. Parameters are:
 - The messageProcessingModel is the value from the processPdu call.
 - The securityModel is the value from the processPdu call.
 - The securityName is the value from the processPdu call.
 - The securityLevel is the value from the processPdu call.
 - The contextEngineID is the value from the processPdu call.
 - The contextName is the value from the processPdu call.
 - The pduVersion indicates the version of the PDU to be returned.

- The PDU is the result generated in step (2) above.
- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.
- The stateReference is the value from the processPdu call.
- The statusInformation indicates that no error occurred and that a response should be generated.

3.5. Proxy Forwarder Applications

A proxy forwarder application deals with forwarding SNMP messages. There are four basic types of messages which a proxy forwarder application may need to forward. These are grouped according to the PDU type contained in a message, or according to whether a report indication is contained in the message. The four basic types of messages are:

- Those containing PDU types which were generated by a command generator application (for example, Get, GetNext, GetBulk, and Set PDU types). These deal with requesting or modifying information located within a particular context.
- Those containing PDU types which were generated by a notification originator application (for example, SNMPv2-Trap and Inform PDU types). These deal with notifications concerning information located within a particular context.
- Those containing a Response PDU type. Forwarding of Response PDUs always occurs as a result of receiving a response to a previously forwarded message.
- Those containing a report indication. Forwarding of report indications always occurs as a result of receiving a report indication for a previously forwarded message.

For the first type, the proxy forwarder's role is to deliver a request for management information to an SNMP engine which is "closer" or "downstream in the path" to the SNMP engine which has access to that information, and to deliver the response containing the information back to the SNMP engine from which the request was received. The context information in a request is used to determine which SNMP engine has access to the requested information, and this is used to determine where and how to forward the request.

For the second type, the proxy forwarder's role is to determine which SNMP engines should receive notifications about management information from a particular location. The context information in a notification message determines the location to which the information contained in the notification applies. This is used to determine which SNMP engines should receive notification about this information.

For the third type, the proxy forwarder's role is to determine which previously forwarded request or notification (if any) the response matches, and to forward the response back to the initiator of the request or notification.

For the fourth type, the proxy forwarder's role is to determine which previously forwarded request or notification (if any) the report indication matches, and to forward the report indication back to the initiator of the request or notification.

When forwarding messages, a proxy forwarder application must perform a translation of incoming management target information into outgoing management target information. How this translation is performed is implementation specific. In many cases, this will be driven by a preconfigured translation table. If a proxy forwarder application makes the contents of this table SNMP manageable, it **MUST** use the SNMP-PROXY-MIB module defined in this document.

3.5.1. Request Forwarding

There are two phases for request forwarding. First, the incoming request needs to be passed through the proxy application. Then, the resulting response needs to be passed back. These phases are described in the following two sections.

3.5.1.1. Processing an Incoming Request

A proxy forwarder application that wishes to forward request messages must first register with the Dispatcher using the registerContextEngineID abstract service interface. The proxy forwarder must register each contextEngineID for which it wishes to forward messages, as well as for each pduType. Note that as the configuration of a proxy forwarder is changed, the particular contextEngineID values for which it is forwarding may change. The proxy forwarder should call the registerContextEngineID and unregisterContextEngineID abstract service interfaces as needed to reflect its current configuration.

A proxy forwarder application should never attempt to register a value of contextEngineID which is equal to the snmpEngineID of the SNMP engine to which the proxy forwarder is associated.

Once the proxy forwarder has registered for the appropriate contextEngineId values, it can start processing messages. The following procedure is used:

- (1) A message is received using the processPdu abstract service interface. The incoming management target information received from the processPdu interface is translated into outgoing management target information. Note that this translation may vary for different values of contextEngineID and/or contextName. The translation should result in a single management target.
- (2) If appropriate outgoing management target information cannot be found, the proxy forwarder increments the snmpProxyDrops counter [RFC1907], and then calls the Dispatcher using the returnResponsePdu abstract service interface. Parameters are:
 - The messageProcessingModel is the value from the processPdu call.
 - The securityModel is the value from the processPdu call.
 - The securityName is the value from the processPdu call.
 - The securityLevel is the value from the processPdu call.
 - The contextEngineID is the value from the processPdu call.
 - The contextName is the value from the processPdu call.
 - The pduVersion is the value from the processPdu call.
 - The PDU is an undefined value.
 - The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.
 - The stateReference is the value from the processPdu call.
 - The statusInformation indicates that an error occurred and includes the OID and value of the snmpProxyDrops object.

Processing of the message stops at this point. Otherwise,

- (3) A new PDU is constructed. A unique value of request-id should be used in the new PDU (this value will enable a subsequent response message to be correlated with this request). The remainder of the new PDU is identical to the received PDU, unless the incoming SNMP version is SNMPv2 or SNMPv3 and the outgoing SNMP version is SNMPv1, in which case the proxy forwarder must apply the translation rules as documented in [RFC1908].
- (4) The proxy forwarder calls the Dispatcher to generate the forwarded message, using the sendPdu abstract service interface. The parameters are:
- The transportDomain is that of the outgoing management target.
 - The transportAddress is that of the outgoing management target.
 - The messageProcessingModel is that of the outgoing management target.
 - The securityModel is that of the outgoing management target.
 - The securityName is that of the outgoing management target.
 - The securityLevel is that of the outgoing management target.
 - The contextEngineID is the value originally received.
 - The contextName is the value originally received.
 - The pduVersion is the version of the PDU to be sent.
 - The PDU is the value constructed in step (3) above.
 - The expectResponse argument indicates that a response is expected. If the sendPdu call is unsuccessful, the proxy forwarder performs the steps described in (2) above. Otherwise:
- (5) The proxy forwarder caches the following information in order to match an incoming response to the forwarded request:
- The sendPduHandle returned from the call to sendPdu,
 - The request-id from the received PDU.
 - the contextEngineID,

- the contextName,
- the stateReference,
- the incoming management target information,
- the outgoing management information,
- any other information needed to match an incoming response to the forwarded request.

If this information cannot be cached (possibly due to a lack of resources), the proxy forwarder performs the steps described in (2) above. Otherwise:

- (6) Processing of the request stops until a response to the forwarded request is received, or until an appropriate time interval has expired. If this time interval expires before a response has been received, the cached information about this request is removed.

3.5.1.2. Processing an Incoming Response

A proxy forwarder follows the following procedure when an incoming response is received:

- (1) The incoming response is received using the processResponsePdu interface. The proxy forwarder uses the received parameters to locate an entry in its cache of pending forwarded requests. This is done by matching the received parameters with the cached values of sendPduHandle, contextEngineID, contextName, outgoing management target information, and the request-id contained in the received PDU (the proxy forwarder must extract the request-id for this purpose). If an appropriate cache entry cannot be found, processing of the response is halted. Otherwise:
- (2) The cache information is extracted, and removed from the cache.
- (3) A new Response PDU is constructed, using the request-id value from the original forwarded request (as extracted from the cache). All other values are identical to those in the received Response PDU.
- (4) If the incoming SNMP version is SNMPv1 and the outgoing SNMP version is SNMPv2 or SNMPv3, the proxy forwarder must apply the translation rules documented in [RFC1908].
- (5) The proxy forwarder calls the Dispatcher using the returnResponsePdu abstract service interface. Parameters are:

- The `messageProcessingModel` indicates the Message Processing Model by which the original incoming message was processed.
- The `securityModel` is that of the original incoming management target extracted from the cache.
- The `securityName` is that of the original incoming management target extracted from the cache.
- The `securityLevel` is that of the original incoming management target extracted from the cache.
- The `contextEngineID` is the value extracted from the cache.
- The `contextName` is the value extracted from the cache.
- The `pduVersion` indicates the version of the PDU to be returned.
- The PDU is the (possibly translated) Response PDU.
- The `maxSizeResponseScopedPDU` is a local value indicating the maximum size of a ScopedPDU that the application can accept.
- The `stateReference` is the value extracted from the cache.
- The `statusInformation` indicates that no error occurred and that a Response PDU message should be generated.

3.5.1.3. Processing an Incoming Report Indication

A proxy forwarder follows the following procedure when an incoming report indication is received:

- (1) The incoming report indication is received using the `processResponsePdu` interface. The proxy forwarder uses the received parameters to locate an entry in its cache of pending forwarded requests. This is done by matching the received parameters with the cached values of `sendPduHandle`. If an appropriate cache entry cannot be found, processing of the report indication is halted. Otherwise:
- (2) The cache information is extracted, and removed from the cache.
- (3) If the original incoming management target information indicates SNMPv1, processing of the report indication is halted.

- (4) The proxy forwarder calls the Dispatcher using the `returnResponsePdu` abstract service interface. Parameters are:
- The `messageProcessingModel` indicates the Message Processing Model by which the original incoming message was processed.
 - The `securityModel` is that of the original incoming management target extracted from the cache.
 - The `securityName` is that of the original incoming management target extracted from the cache.
 - The `securityLevel` is that of the original incoming management target extracted from the cache.
 - The `contextEngineID` is the value extracted from the cache.
 - The `contextName` is the value extracted from the cache.
 - The `pduVersion` indicates the version of the PDU to be returned.
 - The PDU is unused.
 - The `maxSizeResponseScopedPDU` is a local value indicating the maximum size of a ScopedPDU that the application can accept.
 - The `stateReference` is the value extracted from the cache.
 - The `statusInformation` contain the `contextEngineID`, `contextName`, counter OID, and counter value received in the report indication.

3.5.2. Notification Forwarding

A proxy forwarder receives notifications in the same manner as a notification receiver application, using the `processPdu` abstract service interface. The following procedure is used when a notification is received:

- (1) The incoming management target information received from the `processPdu` interface is translated into outgoing management target information. Note that this translation may vary for different values of `contextEngineId` and/or `contextName`. The translation may result in multiple management targets.

- (2) If appropriate outgoing management target information cannot be found and the notification was a Trap, processing of the notification is halted. If appropriate outgoing management target information cannot be found and the notification was an Inform, the proxy forwarder increments the snmpProxyDrops object, and calls the Dispatcher using the returnResponsePdu abstract service interface.

The parameters are:

- The messageProcessingModel is the received value.
- The securityModel is the received value.
- The securityName is the received value.
- The securityLevel is the received value.
- The contextEngineID is the received value.
- The contextName is the received value.
- The pduVersion is the received value.
- The PDU is an undefined and unused value.
- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.
- The stateReference is the received value.
- The statusInformation indicates that an error occurred and that a Report message should be generated.

Processing of the message stops at this point. Otherwise,

- (3) The proxy forwarder generates a notification using the procedures described in the preceding section on Notification Originators, with the following exceptions:
- The contextEngineID and contextName values from the original received notification are used.
 - The outgoing management targets previously determined are used.
 - No filtering mechanisms are applied.

- The variable-bindings from the original received notification are used, rather than retrieving variable-bindings from local MIB instrumentation. In particular, no access-control is applied to these variable-bindings.
 - If for any of the outgoing management targets, the incoming SNMP version is SNMPv1 and the outgoing SNMP version is SNMPv2 or SNMPv3, the proxy forwarder must apply the translation rules as documented in [RFC1908].
 - If for any of the outgoing management targets, the incoming SNMP version is SNMPv2 or SNMPv3, and the outgoing SNMP version is SNMPv1, this outgoing management target is not used when generating the forwarded notifications.
- (4) If the original received notification contains an SNMPv2-Trap PDU, processing of the notification is now completed. Otherwise, the original received notification must contain an Inform PDU, and processing continues.
- (5) If the forwarded notifications included any Inform PDUs, processing continues when the procedures described in the section for Notification Originators determine that either:
- None of the generated notifications containing Inform PDUs have been successfully acknowledged within the longest of the time intervals, in which case processing of the original notification is halted, or,
 - At least one of the generated notifications containing Inform PDUs is successfully acknowledged, in which case a response to the original received notification containing an Inform PDU is generated as described in the following steps.
- (6) A Response PDU is constructed, using the values of request-id and variable-bindings from the original received Inform PDU, and error-status and error-index values of 0.
- (7) The Dispatcher is called using the returnResponsePdu abstract service interface. Parameters are:
- The messageProcessingModel is the originally received value.
 - The securityModel is the originally received value.
 - The securityName is the originally received value.
 - The securityLevel is the originally received value.

- The contextEngineID is the originally received value.
- The contextName is the originally received value.
- The pduVersion indicates the version of the PDU constructed in step (6) above.
- The PDU is the value constructed in step (6) above.
- The maxSizeResponseScopedPDU is a local value indicating the maximum size of a ScopedPDU that the application can accept.
- The stateReference is the originally received value.
- The statusInformation indicates that no error occurred and that a Response PDU message should be generated.

4. The Structure of the MIB Modules

There are three separate MIB modules described in this document, the management target MIB, the notification MIB, and the proxy MIB. The following sections describe the structure of these three MIB modules.

The use of these MIBs by particular types of applications is described later in this document:

- The use of the management target MIB and the notification MIB in notification originator applications is described in section 6.
- The use of the notification MIB for filtering notifications in notification originator applications is described in section 7.
- The use of the management target MIB and the proxy MIB in proxy forwarding applications is described in section 8.

4.1. The Management Target MIB Module

The SNMP-TARGET-MIB module contains objects for defining management targets. It consists of two tables and conformance/compliance statements.

The first table, the snmpTargetAddrTable, contains information about transport domains and addresses. It also contains an object, snmpTargetAddrTagList, which provides a mechanism for grouping entries.

The second table, the `snmpTargetParamsTable`, contains information about SNMP version and security information to be used when sending messages to particular transport domains and addresses.

4.1.1. Tag Lists

The `snmpTargetAddrTagList` object is used for grouping entries in the `snmpTargetAddrTable`. The value of this object contains a list of tag values which are used to select target addresses to be used for a particular operation.

A tag value, which may also be used in MIB objects other than `snmpTargetAddrTagList`, is an arbitrary string of octets, but may not contain a delimiter character. Delimiter characters are defined to be one of the following characters:

- An ASCII space character (0x20).
- An ASCII TAB character (0x09).
- An ASCII carriage return (CR) character (0x0D).
- An ASCII line feed (LF) character (0x0B).

In addition, a tag value may not have a zero length. Generally, a particular MIB object may contain either

- a single tag value, in which case the value of the MIB object may not contain a delimiter character, or:
- a MIB object may contain a list of tag values, separated by single delimiter characters.

For a list of tag values, these constraints imply certain restrictions on the value of a MIB object:

- There cannot be a leading or trailing delimiter character.
- There cannot be multiple adjacent delimiter characters.

4.1.2. Definitions

```
SNMP-TARGET-MIB DEFINITIONS ::= BEGIN
```

```
IMPORTS
```

```
    TEXTUAL-CONVENTION,  
    MODULE-IDENTITY,  
    OBJECT-TYPE,
```

snmpModules,
Integer32
FROM SNMPv2-SMI

TDomain,
TAddress,
TimeInterval,
RowStatus,
StorageType,
TestAndIncr
FROM SNMPv2-TC
SnmpSecurityModel,
SnmpMessageProcessingModel,
SnmpSecurityLevel,
SnmpAdminString
FROM SNMP-FRAMEWORK-MIB

OBJECT-GROUP
FROM SNMPv2-CONF;

snmpTargetMIB MODULE-IDENTITY
LAST-UPDATED "9711210000Z"
ORGANIZATION "IETF SNMPv3 Working Group"
CONTACT-INFO
 "WG-email: snmpv3@tis.com
 Subscribe: majordomo@tis.com
 In message body: subscribe snmpv3

 Chair: Russ Mundy
 Trusted Information Systems
Postal: 3060 Washington Rd
 Glenwood MD 21738
 USA
Email: mundy@tis.com
Phone: +1-301-854-6889

 Co-editor: David B. Levi
 SNMP Research, Inc.
Postal: 3001 Kimberlin Heights Road
 Knoxville, TN 37920-9716
E-mail: levi@snmp.com
Phone: +1 423 573 1434

 Co-editor: Paul Meyer
 Secure Computing Corporation
Postal: 2675 Long Lake Road
 Roseville, MN 55113
E-mail: paul_meyer@securecomputing.com

Phone: +1 612 628 1592

Co-editor: Bob Stewart
Cisco Systems, Inc.
Postal: 170 West Tasman Drive
San Jose, CA 95134-1706
E-mail: bstewart@cisco.com
Phone: +1 603 654 6923"

DESCRIPTION

"This MIB module defines MIB objects which provide mechanisms to remotely configure the parameters used by an SNMP entity for the generation of SNMP messages."

REVISION "9707140000Z"

DESCRIPTION

"The initial revision."

::= { snmpModules 12 }

snmpTargetObjects OBJECT IDENTIFIER ::= { snmpTargetMIB 1 }
snmpTargetConformance OBJECT IDENTIFIER ::= { snmpTargetMIB 3 }

SnmpTagValue ::= TEXTUAL-CONVENTION

DISPLAY-HINT "255a"

STATUS current

DESCRIPTION

"An octet string containing a tag value.
Tag values are preferably in human-readable form.

To facilitate internationalization, this information is represented using the ISO/IEC IS 10646-1 character set, encoded as an octet string using the UTF-8 character encoding scheme described in RFC 2044.

Since additional code points are added by amendments to the 10646 standard from time to time, implementations must be prepared to encounter any code point from 0x00000000 to 0x7fffffff.

The use of control codes should be avoided, and certain control codes are not allowed as described below.

For code points not directly supported by user interface hardware or software, an alternative means of entry and display, such as hexadecimal, may be provided.

For information encoded in 7-bit US-ASCII, the UTF-8 representation is identical to the US-ASCII encoding.

Note that when this TC is used for an object that is used or envisioned to be used as an index, then a SIZE restriction must be specified so that the number sub-identifiers for any object instance do not exceed the limit of 128, as defined by [RFC1905].

An object of this type contains a single tag value which is used to select a set of entries in a table. A tag value is an arbitrary string of octets, but may not contain a delimiter character. Delimiter characters are defined to be one of the following:

- An ASCII space character (0x20).
- An ASCII TAB character (0x09).
- An ASCII carriage return (CR) character (0x0D).
- An ASCII line feed (LF) character (0x0B).

Delimiter characters are used to separate tag values in a tag list. An object of this type may only contain a single tag value, and so delimiter characters are not allowed in a value of this type.

Some examples of valid tag values are:

- 'acme'
- 'router'
- 'host'

The use of a tag value to select table entries is application and MIB specific."

SYNTAX OCTET STRING (SIZE (0..255))

SnmpTagList ::= TEXTUAL-CONVENTION

DISPLAY-HINT "255a"

STATUS current

DESCRIPTION

"An octet string containing a list of tag values.
Tag values are preferably in human-readable form.

To facilitate internationalization, this information is represented using the ISO/IEC IS 10646-1 character set, encoded as an octet string using the UTF-8 character encoding scheme described in RFC 2044.

Since additional code points are added by amendments to the 10646 standard from time to time, implementations must be prepared to encounter any code point from 0x00000000 to 0x7fffffff.

The use of control codes should be avoided, except as described below.

For code points not directly supported by user interface hardware or software, an alternative means of entry and display, such as hexadecimal, may be provided.

For information encoded in 7-bit US-ASCII, the UTF-8 representation is identical to the US-ASCII encoding.

An object of this type contains a list of tag values which are used to select a set of entries in a table.

A tag value is an arbitrary string of octets, but may not contain a delimiter character. Delimiter characters are defined to be one of the following:

- An ASCII space character (0x20).
- An ASCII TAB character (0x09).
- An ASCII carriage return (CR) character (0x0D).
- An ASCII line feed (LF) character (0x0B).

Delimiter characters are used to separate tag values in a tag list. Only a single delimiter character may occur between two tag values. A tag value may not have a zero length. These constraints imply certain restrictions on the contents of this object:

- There cannot be a leading or trailing delimiter character.
- There cannot be multiple adjacent delimiter characters.

Some examples of valid tag lists are:

- An empty string
- 'acme router'

- 'host managerStation'

Note that although a tag value may not have a length of zero, an empty string is still valid. This indicates an empty list (i.e. there are no tag values in the list).

The use of the tag list to select table entries is application and MIB specific. Typically, an application will provide one or more tag values, and any entry which contains some combination of these tag values will be selected."

SYNTAX OCTET STRING (SIZE (0..255))

--
--
--
--
--

The snmpTargetObjects group

snmpTargetSpinLock OBJECT-TYPE

SYNTAX TestAndIncr

MAX-ACCESS read-write

STATUS current

DESCRIPTION

"This object is used to facilitate modification of table entries in the SNMP-TARGET-MIB module by multiple managers. In particular, it is useful when modifying the value of the snmpTargetAddrTagList object.

The procedure for modifying the snmpTargetAddrTagList object is as follows:

1. Retrieve the value of snmpTargetSpinLock and of snmpTargetAddrTagList.
2. Generate a new value for snmpTargetAddrTagList.
3. Set the value of snmpTargetSpinLock to the retrieved value, and the value of snmpTargetAddrTagList to the new value. If the set fails for the snmpTargetSpinLock object, go back to step 1."

::= { snmpTargetObjects 1 }

snmpTargetAddrTable OBJECT-TYPE

SYNTAX SEQUENCE OF SnmpTargetAddrEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A table of transport addresses to be used in the generation of SNMP messages."

::= { snmpTargetObjects 2 }

snmpTargetAddrEntry OBJECT-TYPE

SYNTAX SnmpTargetAddrEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A transport address to be used in the generation of SNMP operations."

Entries in the snmpTargetAddrTable are created and deleted using the snmpTargetAddrRowStatus object."

INDEX { IMPLIED snmpTargetAddrName }

::= { snmpTargetAddrTable 1 }

SnmpTargetAddrEntry ::= SEQUENCE {

snmpTargetAddrName SnmpAdminString,

snmpTargetAddrTDomain TDomain,

snmpTargetAddrTAddress TAddress,

snmpTargetAddrTimeout TimeInterval,

snmpTargetAddrRetryCount Integer32,

snmpTargetAddrTagList SnmpTagList,

snmpTargetAddrParams SnmpAdminString,

snmpTargetAddrStorageType StorageType,

snmpTargetAddrRowStatus RowStatus

}

snmpTargetAddrName OBJECT-TYPE

SYNTAX SnmpAdminString (SIZE(1..32))

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The locally arbitrary, but unique identifier associated with this snmpTargetAddrEntry."

::= { snmpTargetAddrEntry 1 }

snmpTargetAddrTDomain OBJECT-TYPE

SYNTAX TDomain

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object indicates the transport type of the address contained in the snmpTargetAddrTAddress object."

::= { snmpTargetAddrEntry 2 }

snmpTargetAddrTAddress OBJECT-TYPE

SYNTAX TAddress
MAX-ACCESS read-create
STATUS current
DESCRIPTION

"This object contains a transport address. The format of this address depends on the value of the snmpTargetAddrTDomain object."

::= { snmpTargetAddrEntry 3 }

snmpTargetAddrTimeout OBJECT-TYPE

SYNTAX TimeInterval
MAX-ACCESS read-create
STATUS current
DESCRIPTION

"This object should reflect the expected maximum round trip time for communicating with the transport address defined by this row. When a message is sent to this address, and a response (if one is expected) is not received within this time period, an implementation may assume that the response will not be delivered.

Note that the time interval that an application waits for a response may actually be derived from the value of this object. The method for deriving the actual time interval is implementation dependent. One such method is to derive the expected round trip time based on a particular retransmission algorithm and on the number of timeouts which have occurred. The type of message may also be considered when deriving expected round trip times for retransmissions. For example, if a message is being sent with a securityLevel that indicates both authentication and privacy, the derived value may be increased to compensate for extra processing time spent during authentication and encryption processing."

DEFVAL { 1500 }

::= { snmpTargetAddrEntry 4 }

snmpTargetAddrRetryCount OBJECT-TYPE

SYNTAX Integer32 (0..255)
MAX-ACCESS read-create
STATUS current
DESCRIPTION

"This object specifies a default number of retries to be attempted when a response is not received for a generated message. An application may provide its own retry count, in which case the value of this object is ignored."

DEFVAL { 3 }

::= { snmpTargetAddrEntry 5 }

snmpTargetAddrTagList OBJECT-TYPE

SYNTAX SnmpTagList

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object contains a list of tag values which are used to select target addresses for a particular operation."

::= { snmpTargetAddrEntry 6 }

snmpTargetAddrParams OBJECT-TYPE

SYNTAX SnmpAdminString (SIZE(1..32))

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The value of this object identifies an entry in the snmpTargetParamsTable. The identified entry contains SNMP parameters to be used when generating messages to be sent to this transport address."

::= { snmpTargetAddrEntry 7 }

snmpTargetAddrStorageType OBJECT-TYPE

SYNTAX StorageType

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The storage type for this conceptual row."

::= { snmpTargetAddrEntry 8 }

snmpTargetAddrRowStatus OBJECT-TYPE

SYNTAX RowStatus

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The status of this conceptual row.

To create a row in this table, a manager must set this object to either createAndGo(4) or createAndWait(5).

Until instances of all corresponding columns are appropriately configured, the value of the corresponding instance of the snmpTargetAddrRowStatus column is 'notReady'.

In particular, a newly created row cannot be made

active until the corresponding snmpTargetAddrTDomain and snmpTargetAddrTAddress have both been set.

The following objects may not be modified while the value of this object is active(1):

- snmpTargetAddrTDomain
- snmpTargetAddrTAddress"

::= { snmpTargetAddrEntry 9 }

snmpTargetParamsTable OBJECT-TYPE

SYNTAX SEQUENCE OF SnmpTargetParamsEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A table of SNMP target information to be used in the generation of SNMP messages."

::= { snmpTargetObjects 3 }

snmpTargetParamsEntry OBJECT-TYPE

SYNTAX SnmpTargetParamsEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"A set of SNMP target information.

Entries in the snmpTargetParamsTable are created and deleted using the snmpTargetParamsRowStatus object."

INDEX { IMPLIED snmpTargetParamsName }

::= { snmpTargetParamsTable 1 }

SnmpTargetParamsEntry ::= SEQUENCE {

snmpTargetParamsName	SnmpAdminString,
snmpTargetParamsMPModel	SnmpMessageProcessingModel,
snmpTargetParamsSecurityModel	SnmpSecurityModel,
snmpTargetParamsSecurityName	SnmpAdminString,
snmpTargetParamsSecurityLevel	SnmpSecurityLevel,
snmpTargetParamsStorageType	StorageType,
snmpTargetParamsRowStatus	RowStatus

}

snmpTargetParamsName OBJECT-TYPE

SYNTAX SnmpAdminString (SIZE(1..32))

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The locally arbitrary, but unique identifier associated with this snmpTargetParamsEntry."

::= { snmpTargetParamsEntry 1 }

snmpTargetParamsMPModel OBJECT-TYPE
SYNTAX SnmpMessageProcessingModel
MAX-ACCESS read-create
STATUS current
DESCRIPTION
 "The Message Processing Model to be used when generating
 SNMP messages using this entry."
 ::= { snmpTargetParamsEntry 2 }

snmpTargetParamsSecurityModel OBJECT-TYPE
SYNTAX SnmpSecurityModel (0..254 | 256..2147483647)
MAX-ACCESS read-create
STATUS current
DESCRIPTION
 "The Security Model to be used when generating SNMP
 messages using this entry."
 ::= { snmpTargetParamsEntry 3 }

snmpTargetParamsSecurityName OBJECT-TYPE
SYNTAX SnmpAdminString
MAX-ACCESS read-create
STATUS current
DESCRIPTION
 "The securityName which identifies the Principal on
 whose behalf SNMP messages will be generated using
 this entry."
 ::= { snmpTargetParamsEntry 4 }

snmpTargetParamsSecurityLevel OBJECT-TYPE
SYNTAX SnmpSecurityLevel
MAX-ACCESS read-create
STATUS current
DESCRIPTION
 "The Level of Security to be used when generating
 SNMP messages using this entry."
 ::= { snmpTargetParamsEntry 5 }

snmpTargetParamsStorageType OBJECT-TYPE
SYNTAX StorageType
MAX-ACCESS read-create
STATUS current
DESCRIPTION
 "The storage type for this conceptual row."
 ::= { snmpTargetParamsEntry 6 }

snmpTargetParamsRowStatus OBJECT-TYPE
SYNTAX RowStatus
MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The status of this conceptual row.

To create a row in this table, a manager must set this object to either createAndGo(4) or createAndWait(5).

Until instances of all corresponding columns are appropriately configured, the value of the corresponding instance of the snmpTargetParamsRowStatus column is 'notReady'.

In particular, a newly created row cannot be made active until the corresponding snmpTargetParamsMPModel, snmpTargetParamsSecurityModel, snmpTargetParamsSecurityName, and snmpTargetParamsSecurityLevel have all been set.

The following objects may not be modified while the value of this object is active(1):

- snmpTargetParamsMPModel
- snmpTargetParamsSecurityModel
- snmpTargetParamsSecurityName
- snmpTargetParamsSecurityLevel"

::= { snmpTargetParamsEntry 7 }

snmpUnavailableContexts OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total number of packets received by the SNMP engine which were dropped because the context contained in the message was unavailable."

::= { snmpTargetObjects 4 }

snmpUnknownContexts OBJECT-TYPE

SYNTAX Counter32

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The total number of packets received by the SNMP engine which were dropped because the context contained in the message was unknown."

::= { snmpTargetObjects 5 }

```

--
--
-- Conformance information
--
--
snmpTargetCompliances OBJECT IDENTIFIER ::=
                                { snmpTargetConformance 1 }
snmpTargetGroups      OBJECT IDENTIFIER ::=
                                { snmpTargetConformance 2 }

--
--
-- Compliance statements
--
--

snmpTargetCommandResponderCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION
        "The compliance statement for SNMP entities which include
        a command responder application."
    MODULE -- This Module
        MANDATORY-GROUPS { snmpTargetCommandResponderGroup }
    ::= { snmpTargetCompliances 1 }

snmpTargetBasicGroup OBJECT-GROUP
    OBJECTS {
        snmpTargetSpinLock,
        snmpTargetAddrTDomain,
        snmpTargetAddrTAddress,
        snmpTargetAddrTagList,
        snmpTargetAddrParams,
        snmpTargetAddrStorageType,
        snmpTargetAddrRowStatus,
        snmpTargetParamsMPModel,
        snmpTargetParamsSecurityModel,
        snmpTargetParamsSecurityName,
        snmpTargetParamsSecurityLevel,
        snmpTargetParamsStorageType,
        snmpTargetParamsRowStatus
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects providing basic remote
        configuration of management targets."

```



```

        ::= { snmpTargetGroups 1 }

snmpTargetResponseGroup OBJECT-GROUP
    OBJECTS {
        snmpTargetAddrTimeout,
        snmpTargetAddrRetryCount
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects providing remote configuration
        of management targets for applications which generate
        SNMP messages for which a response message would be
        expected."
    ::= { snmpTargetGroups 2 }

snmpTargetCommandResponderGroup OBJECT-GROUP
    OBJECTS {
        snmpUnavailableContexts,
        snmpUnknownContexts
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects required for command responder
        applications, used for counting error conditions."
    ::= { snmpTargetGroups 3 }

END

```

4.2. The Notification MIB Module

The SNMP-NOTIFICATION-MIB module contains objects for the remote configuration of the parameters used by an SNMP entity for the generation of notifications. It consists of three tables and conformance/compliance statements. The first table, the `snmpNotifyTable`, contains entries which select which entries in the `snmpTargetAddrTable` should be used for generating notifications, and the type of notifications to be generated.

The second table sparsely augments the `snmpTargetAddrTable` with an object which is used to associate a set of filters with a particular management target.

The third table defines filters which are used to limit the number of notifications which are generated using particular management targets.

4.2.1. Definitions

```
SNMP-NOTIFICATION-MIB DEFINITIONS ::= BEGIN
```

```
IMPORTS
```

```
    MODULE-IDENTITY,  
    OBJECT-TYPE,  
    snmpModules  
        FROM SNMPv2-SMI
```

```
    RowStatus,  
    StorageType  
        FROM SNMPv2-TC
```

```
    SnmpAdminString  
        FROM SNMP-FRAMEWORK-MIB
```

```
    SnmpTagValue,  
    snmpTargetParamsName  
        FROM SNMP-TARGET-MIB
```

```
    MODULE-COMPLIANCE,  
    OBJECT-GROUP  
        FROM SNMPv2-CONF;
```

```
snmpNotificationMIB MODULE-IDENTITY
```

```
    LAST-UPDATED "9711210000Z"
```

```
    ORGANIZATION "IETF SNMPv3 Working Group"
```

```
    CONTACT-INFO
```

```
        "WG-email:    snmpv3@tis.com  
        Subscribe:    majordomo@tis.com  
        In message body:  subscribe snmpv3
```

```
        Chair:        Russ Mundy  
                        Trusted Information Systems  
        Postal:        3060 Washington Rd  
                        Glenwood MD 21738  
                        USA  
        Email:         mundy@tis.com  
        Phone:         +1-301-854-6889
```

```
        Co-editor:    David B. Levi  
                        SNMP Research, Inc.  
        Postal:        3001 Kimberlin Heights Road  
                        Knoxville, TN 37920-9716  
        E-mail:        levi@snmp.com  
        Phone:         +1 423 573 1434
```

Co-editor: Paul Meyer
 Secure Computing Corporation
 Postal: 2675 Long Lake Road
 Roseville, MN 55113
 E-mail: paul_meyer@securecomputing.com
 Phone: +1 612 628 1592

Co-editor: Bob Stewart
 Cisco Systems, Inc.
 Postal: 170 West Tasman Drive
 San Jose, CA 95134-1706
 E-mail: bstewart@cisco.com
 Phone: +1 603 654 6923"

DESCRIPTION

"This MIB module defines MIB objects which provide mechanisms to remotely configure the parameters used by an SNMP entity for the generation of notifications."

REVISION "9707140000Z"

DESCRIPTION

"The initial revision."

::= { snmpModules 13 }

snmpNotifyObjects OBJECT IDENTIFIER ::= { snmpNotificationMIB 1 }
 snmpNotifyConformance OBJECT IDENTIFIER ::= { snmpNotificationMIB 3 }

--
 --
 -- The snmpNotifyObjects group
 --
 --

snmpNotifyTable OBJECT-TYPE

SYNTAX SEQUENCE OF SnmpNotifyEntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"This table is used to select management targets which should receive notifications, as well as the type of notification which should be sent to each selected management target."

::= { snmpNotifyObjects 1 }

snmpNotifyEntry OBJECT-TYPE

SYNTAX SnmpNotifyEntry

MAX-ACCESS not-accessible

STATUS current
DESCRIPTION

"An entry in this table selects a set of management targets which should receive notifications, as well as the type of notification which should be sent to each selected management target.

Entries in the snmpNotifyTable are created and deleted using the snmpNotifyRowStatus object."

INDEX { IMPLIED snmpNotifyName }
::= { snmpNotifyTable 1 }

SnmpNotifyEntry ::= SEQUENCE {
 snmpNotifyName SnmpAdminString,
 snmpNotifyTag SnmpTagValue,
 snmpNotifyType INTEGER,
 snmpNotifyStorageType StorageType,
 snmpNotifyRowStatus RowStatus
}

snmpNotifyName OBJECT-TYPE

SYNTAX SnmpAdminString (SIZE(1..32))

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The locally arbitrary, but unique identifier associated with this snmpNotifyEntry."

::= { snmpNotifyEntry 1 }

snmpNotifyTag OBJECT-TYPE

SYNTAX SnmpTagValue

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object contains a single tag value which is used to select entries in the snmpTargetAddrTable. Any entry in the snmpTargetAddrTable which contains a tag value which is equal to the value of an instance of this object is selected. If this object contains a value of zero length, no entries are selected."

::= { snmpNotifyEntry 2 }

snmpNotifyType OBJECT-TYPE

SYNTAX INTEGER {
 trap(1),
 inform(2)
}

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object determines the type of notification to be generated for entries in the snmpTargetAddrTable selected by the corresponding instance of snmpNotifyTag.

If the value of this object is trap(1), then any messages generated for selected rows will contain SNMPv2-Trap PDUs.

If the value of this object is inform(2), then any messages generated for selected rows will contain Inform PDUs.

Note that if an SNMP entity only supports generation of traps (and not informs), then this object may be read-only."

DEFVAL { trap }

::= { snmpNotifyEntry 3 }

snmpNotifyStorageType OBJECT-TYPE

SYNTAX StorageType

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The storage type for this conceptual row."

::= { snmpNotifyEntry 4 }

snmpNotifyRowStatus OBJECT-TYPE

SYNTAX RowStatus

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The status of this conceptual row.

To create a row in this table, a manager must set this object to either createAndGo(4) or createAndWait(5).

Until instances of all corresponding columns are appropriately configured, the value of the corresponding instance of the snmpNotifyRowStatus column is 'notReady'.

In particular, a newly created row cannot be made active until the corresponding snmpNotifyTag has been set."

```

 ::= { snmpNotifyEntry 5 }

snmpNotifyFilterProfileTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpNotifyFilterProfileEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table is used to associate a notification filter
         profile with a particular set of target parameters."
    ::= { snmpNotifyObjects 2 }

snmpNotifyFilterProfileEntry OBJECT-TYPE
    SYNTAX      SnmpNotifyFilterProfileEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry in this table indicates the name of the filter
         profile to be used when generating notifications using
         the corresponding entry in the snmpTargetParamsTable.

         Entries in the snmpNotifyFilterProfileTable are created
         and deleted using the snmpNotifyFilterProfileRowStatus
         object."
    INDEX { IMPLIED snmpTargetParamsName }
    ::= { snmpNotifyFilterProfileTable 1 }

SnmpNotifyFilterProfileEntry ::= SEQUENCE {
    snmpNotifyFilterProfileName      SnmpAdminString,
    snmpNotifyFilterProfileStorType  StorageType,
    snmpNotifyFilterProfileRowStatus RowStatus
}

snmpNotifyFilterProfileName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(1..32))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The name of the filter profile to be used when generating
         notifications using the corresponding entry in the
         snmpTargetAddrTable."
    ::= { snmpNotifyFilterProfileEntry 1 }

snmpNotifyFilterProfileStorType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The storage type of this conceptual row."

```

```
 ::= { snmpNotifyFilterProfileEntry 2 }
```

snmpNotifyFilterProfileRowStatus OBJECT-TYPE

```
SYNTAX      RowStatus
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
```

"The status of this conceptual row.

To create a row in this table, a manager must set this object to either createAndGo(4) or createAndWait(5)."

```
 ::= { snmpNotifyFilterProfileEntry 3 }
```

snmpNotifyFilterTable OBJECT-TYPE

```
SYNTAX      SEQUENCE OF SnmpNotifyFilterEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
```

"The table of filter profiles. Filter profiles are used to determine whether particular management targets should receive particular notifications.

When a notification is generated, it must be compared with the filters associated with each management target which is configured to receive notifications. If the notification is matched by a filter, it is not sent to the management target with which the filter is associated."

```
 ::= { snmpNotifyObjects 3 }
```

snmpNotifyFilterEntry OBJECT-TYPE

```
SYNTAX      SnmpNotifyFilterEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
```

"An element of a filter profile.

Entries in the snmpNotifyFilterTable are created and deleted using the snmpNotifyFilterRowStatus object."

```
INDEX {      snmpNotifyFilterProfileName,
            IMPLIED snmpNotifyFilterSubtree }
```

```
 ::= { snmpNotifyFilterTable 1 }
```

SnmpNotifyFilterEntry ::= SEQUENCE {

snmpNotifyFilterSubtree	OBJECT IDENTIFIER,
snmpNotifyFilterMask	OCTET STRING,
snmpNotifyFilterType	INTEGER,

```

        snmpNotifyFilterStorageType      StorageType,
        snmpNotifyFilterRowStatus        RowStatus
    }

```

snmpNotifyFilterSubtree OBJECT-TYPE

SYNTAX OBJECT IDENTIFIER

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The MIB subtree which, when combined with the corresponding instance of snmpNotifyFilterMask, defines a family of subtrees which are included in or excluded from the filter profile."

::= { snmpNotifyFilterEntry 1 }

snmpNotifyFilterMask OBJECT-TYPE

SYNTAX OCTET STRING (SIZE(0..16))

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The bit mask which, in combination with the corresponding instance of snmpNotifyFilterSubtree, defines a family of subtrees which are included in or excluded from the filter profile."

Each bit of this bit mask corresponds to a sub-identifier of snmpNotifyFilterSubtree, with the most significant bit of the i-th octet of this octet string value (extended if necessary, see below) corresponding to the (8*i - 7)-th sub-identifier, and the least significant bit of the i-th octet of this octet string corresponding to the (8*i)-th sub-identifier, where i is in the range 1 through 16.

Each bit of this bit mask specifies whether or not the corresponding sub-identifiers must match when determining if an OBJECT IDENTIFIER matches this family of filter subtrees; a '1' indicates that an exact match must occur; a '0' indicates 'wild card', i.e., any sub-identifier value matches.

Thus, the OBJECT IDENTIFIER X of an object instance is contained in a family of filter subtrees if, for each sub-identifier of the value of snmpNotifyFilterSubtree, either:

the i-th bit of snmpNotifyFilterMask is 0, or

the i-th sub-identifier of X is equal to the i-th sub-identifier of the value of snmpNotifyFilterSubtree.

If the value of this bit mask is M bits long and there are more than M sub-identifiers in the corresponding instance of snmpNotifyFilterSubtree, then the bit mask is extended with 1's to be the required length.

Note that when the value of this object is the zero-length string, this extension rule results in a mask of all-1's being used (i.e., no 'wild card'), and the family of filter subtrees is the one subtree uniquely identified by the corresponding instance of snmpNotifyFilterSubtree."

```
DEFVAL { ''H }
::= { snmpNotifyFilterEntry 2 }
```

snmpNotifyFilterType OBJECT-TYPE

```
SYNTAX      INTEGER {
                included(1),
                excluded(2)
            }
```

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object indicates whether the family of filter subtrees defined by this entry are included in or excluded from a filter."

```
DEFVAL { included }
::= { snmpNotifyFilterEntry 3 }
```

snmpNotifyFilterStorageType OBJECT-TYPE

```
SYNTAX      StorageType
```

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The storage type of this conceptual row."

```
::= { snmpNotifyFilterEntry 4 }
```

snmpNotifyFilterRowStatus OBJECT-TYPE

```
SYNTAX      RowStatus
```

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The status of this conceptual row."

```

        To create a row in this table, a manager must
        set this object to either createAndGo(4) or
        createAndWait(5)."
 ::= { snmpNotifyFilterEntry 5 }

--
--
-- Conformance information
--
--

snmpNotifyCompliances OBJECT IDENTIFIER ::=
                                { snmpNotifyConformance 1 }
snmpNotifyGroups      OBJECT IDENTIFIER ::=
                                { snmpNotifyConformance 2 }

--
--
-- Compliance statements
--
--

snmpNotifyBasicCompliance MODULE-COMPLIANCE
    STATUS      current
    DESCRIPTION
        "The compliance statement for minimal SNMP entities which
        implement only SNMP Traps and read-create operations on
        only the snmpTargetAddrTable."
    MODULE SNMP-TARGET-MIB
        MANDATORY-GROUPS { snmpTargetBasicGroup }

        OBJECT snmpTargetParamsMPPModel
        MIN-ACCESS      read-only
        DESCRIPTION
            "Create/delete/modify access is not required."

        OBJECT snmpTargetParamsSecurityModel
        MIN-ACCESS      read-only
        DESCRIPTION
            "Create/delete/modify access is not required."

        OBJECT snmpTargetParamsSecurityName
        MIN-ACCESS      read-only
        DESCRIPTION
            "Create/delete/modify access is not required."

        OBJECT snmpTargetParamsSecurityLevel
        MIN-ACCESS      read-only

```

DESCRIPTION

"Create/delete/modify access is not required."

OBJECT snmpTargetParamsStorageType

SYNTAX INTEGER {
 readOnly(5)

}

MIN-ACCESS read-only

DESCRIPTION

"Create/delete/modify access is not required.
Support of the values other(1), volatile(2),
nonVolatile(3), and permanent(4) is not required."

OBJECT snmpTargetParamsRowStatus

SYNTAX INTEGER {
 active(1)

}

MIN-ACCESS read-only

DESCRIPTION

"Create/delete/modify access to the
snmpTargetParamsTable is not required.
Support of the values notInService(2), notReady(3),
createAndGo(4), createAndWait(5), and destroy(6) is
not required."

MODULE -- This Module

MANDATORY-GROUPS { snmpNotifyGroup }

OBJECT snmpNotifyTag

MIN-ACCESS read-only

DESCRIPTION

"Create/delete/modify access is not required."

OBJECT snmpNotifyType

SYNTAX INTEGER {
 trap(1)

}

MIN-ACCESS read-only

DESCRIPTION

"Create/delete/modify access is not required.
Support of the value notify(2) is not required."

OBJECT snmpNotifyStorageType

SYNTAX INTEGER {
 readOnly(5)

}

MIN-ACCESS read-only

DESCRIPTION

"Create/delete/modify access is not required.
Support of the values other(1), volatile(2),
nonVolatile(3), and permanent(4) is not required."

```
OBJECT snmpNotifyRowStatus
SYNTAX INTEGER {
    active(1)
}
MIN-ACCESS      read-only
DESCRIPTION
    "Create/delete/modify access to the
    snmpNotifyTable is not required.
    Support of the values notInService(2), notReady(3),
    createAndGo(4), createAndWait(5), and destroy(6) is
    not required."
```

```
::= { snmpNotifyCompliances 1 }
```

```
snmpNotifyBasicFiltersCompliance MODULE-COMPLIANCE
```

```
STATUS      current
```

```
DESCRIPTION
```

"The compliance statement for SNMP entities which implement
SNMP Traps with filtering, and read-create operations on
all related tables."

```
MODULE SNMP-TARGET-MIB
```

```
MANDATORY-GROUPS { snmpTargetBasicGroup }
```

```
MODULE -- This Module
```

```
MANDATORY-GROUPS { snmpNotifyGroup,
                    snmpNotifyFilterGroup }
```

```
::= { snmpNotifyCompliances 2 }
```

```
snmpNotifyFullCompliance MODULE-COMPLIANCE
```

```
STATUS      current
```

```
DESCRIPTION
```

"The compliance statement for SNMP entities which either
implement only SNMP Informs, or both SNMP Traps and SNMP
Informs, plus filtering and read-create operations on
all related tables."

```
MODULE SNMP-TARGET-MIB
```

```
MANDATORY-GROUPS { snmpTargetBasicGroup,
                    snmpTargetResponseGroup }
```

```
MODULE -- This Module
```

```
MANDATORY-GROUPS { snmpNotifyGroup,
                    snmpNotifyFilterGroup }
```

```
::= { snmpNotifyCompliances 3 }
```

```
snmpNotifyGroup OBJECT-GROUP
```

```
OBJECTS {
```

```

        snmpNotifyTag,
        snmpNotifyType,
        snmpNotifyStorageType,
        snmpNotifyRowStatus
    }
    STATUS          current
    DESCRIPTION
        "A collection of objects for selecting which management
        targets are used for generating notifications, and the
        type of notification to be generated for each selected
        management target."
    ::= { snmpNotifyGroups 1 }

snmpNotifyFilterGroup OBJECT-GROUP
    OBJECTS {
        snmpNotifyFilterProfileName,
        snmpNotifyFilterProfileStorType,
        snmpNotifyFilterProfileRowStatus,
        snmpNotifyFilterMask,
        snmpNotifyFilterType,
        snmpNotifyFilterStorageType,
        snmpNotifyFilterRowStatus
    }
    STATUS          current
    DESCRIPTION
        "A collection of objects providing remote configuration
        of notification filters."
    ::= { snmpNotifyGroups 2 }

END

```

4.3. The Proxy MIB Module

The SNMP-PROXY-MIB module, which defines MIB objects that provide mechanisms to remotely configure the parameters used by an SNMP entity for proxy forwarding operations, contains a single table. This table, `snmpProxyTable`, is used to define translations between management targets for use when forwarding messages.

4.3.1. Definitions

```

SNMP-PROXY-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY,
    OBJECT-TYPE,
    snmpModules
    FROM SNMPv2-SMI

```

RowStatus,
StorageType
FROM SNMPv2-TC

SnmpEngineID,
SnmpAdminString
FROM SNMP-FRAMEWORK-MIB

SnmpTagValue,
FROM SNMP-TARGET-MIB

MODULE-COMPLIANCE,
OBJECT-GROUP
FROM SNMPv2-CONF;

snmpProxyMIB MODULE-IDENTITY
LAST-UPDATED "9711210000Z"
ORGANIZATION "IETF SNMPv3 Working Group"
CONTACT-INFO
 "WG-email: snmpv3@tis.com
 Subscribe: majordomo@tis.com
 In message body: subscribe snmpv3

 Chair: Russ Mundy
 Trusted Information Systems
Postal: 3060 Washington Rd
 Glenwood MD 21738
 USA
Email: mundy@tis.com
Phone: +1-301-854-6889

 Co-editor: David B. Levi
 SNMP Research, Inc.
Postal: 3001 Kimberlin Heights Road
 Knoxville, TN 37920-9716
E-mail: levi@snmp.com
Phone: +1 423 573 1434

 Co-editor: Paul Meyer
 Secure Computing Corporation
Postal: 2675 Long Lake Road
 Roseville, MN 55113
E-mail: paul_meyer@securecomputing.com
Phone: +1 612 628 1592

 Co-editor: Bob Stewart
 Cisco Systems, Inc.
Postal: 170 West Tasman Drive

```

                San Jose, CA 95134-1706
    E-mail:      bstewart@cisco.com
    Phone:       +1 603 654 6923"
DESCRIPTION
    "This MIB module defines MIB objects which provide
    mechanisms to remotely configure the parameters
    used by a proxy forwarding application."
REVISION       "9707140000Z"
DESCRIPTION
    "The initial revision."
    ::= { snmpModules 14 }

snmpProxyObjects      OBJECT IDENTIFIER ::= { snmpProxyMIB 1 }
snmpProxyConformance OBJECT IDENTIFIER ::= { snmpProxyMIB 3 }

--
--
-- The snmpProxyObjects group
--
--

snmpProxyTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF SnmpProxyEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The table of translation parameters used by proxy forwarder
        applications for forwarding SNMP messages."
    ::= { snmpProxyObjects 2 }

snmpProxyEntry OBJECT-TYPE
    SYNTAX      SnmpProxyEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A set of translation parameters used by a proxy forwarder
        application for forwarding SNMP messages.

        Entries in the snmpProxyTable are created and deleted
        using the snmpProxyRowStatus object."
    INDEX { IMPLIED snmpProxyName }
    ::= { snmpProxyTable 1 }

SnmpProxyEntry ::= SEQUENCE {
    snmpProxyName          SnmpAdminString,
    snmpProxyType          INTEGER,
    snmpProxyContextEngineID SnmpEngineID,
    snmpProxyContextName   SnmpAdminString,

```

```

    snmpProxyTargetParamsIn      SnmpAdminString,
    snmpProxySingleTargetOut     SnmpAdminString,
    snmpProxyMultipleTargetOut   SnmpTagValue,
    snmpProxyStorageType         StorageType,
    snmpProxyRowStatus           RowStatus
}

snmpProxyName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE(1..32))
    MAX-ACCESS   not-accessible
    STATUS       current
    DESCRIPTION
        "The locally arbitrary, but unique identifier associated
         with this snmpProxyEntry."
    ::= { snmpProxyEntry 1 }

snmpProxyType OBJECT-TYPE
    SYNTAX      INTEGER {
                    read(1),
                    write(2),
                    trap(3),
                    inform(4)
                }
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The type of message that may be forwarded using
         the translation parameters defined by this entry."
    ::= { snmpProxyEntry 2 }

snmpProxyContextEngineID OBJECT-TYPE
    SYNTAX      SnmpEngineID
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The contextEngineID contained in messages that
         may be forwarded using the translation parameters
         defined by this entry."
    ::= { snmpProxyEntry 3 }

snmpProxyContextName OBJECT-TYPE
    SYNTAX      SnmpAdminString
    MAX-ACCESS   read-create
    STATUS       current
    DESCRIPTION
        "The contextName contained in messages that may be
         forwarded using the translation parameters defined
         by this entry."

```


This object is optional, and if not supported, the contextName contained in a message is ignored when selecting an entry in the snmpProxyTable."
 ::= { snmpProxyEntry 4 }

snmpProxyTargetParamsIn OBJECT-TYPE

SYNTAX SnmpAdminString

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object selects an entry in the snmpTargetParamsTable. The selected entry is used to determine which row of the snmpProxyTable to use for forwarding received messages."

::= { snmpProxyEntry 5 }

snmpProxySingleTargetOut OBJECT-TYPE

SYNTAX SnmpAdminString

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object selects a management target defined in the snmpTargetAddrTable (in the SNMP-TARGET-MIB). The selected target is defined by an entry in the snmpTargetAddrTable whose index value (snmpTargetAddrName) is equal to this object.

This object is only used when selection of a single target is required (i.e. when forwarding an incoming read or write request)."

::= { snmpProxyEntry 6 }

snmpProxyMultipleTargetOut OBJECT-TYPE

SYNTAX SnmpTagValue

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"This object selects a set of management targets defined in the snmpTargetAddrTable (in the SNMP-TARGET-MIB).

This object is only used when selection of multiple targets is required (i.e. when forwarding an incoming notification)."

::= { snmpProxyEntry 7 }

snmpProxyStorageType OBJECT-TYPE

SYNTAX StorageType

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The storage type of this conceptual row."

::= { snmpProxyEntry 8 }

snmpProxyRowStatus OBJECT-TYPE

SYNTAX RowStatus

MAX-ACCESS read-create

STATUS current

DESCRIPTION

"The status of this conceptual row."

To create a row in this table, a manager must set this object to either createAndGo(4) or createAndWait(5).

The following objects may not be modified while the value of this object is active(1):

- snmpProxyType
- snmpProxyContextEngineID
- snmpProxyContextName
- snmpProxyTargetParamsIn
- snmpProxySingleTargetOut
- snmpProxyMultipleTargetOut"

::= { snmpProxyEntry 9 }

--

--

-- Conformance information

--

--

snmpProxyCompliances OBJECT IDENTIFIER ::=

{ snmpProxyConformance 1 }

snmpProxyGroups OBJECT IDENTIFIER ::=

{ snmpProxyConformance 2 }

--

--

-- Compliance statements

--

--

snmpProxyCompliance MODULE-COMPLIANCE

STATUS current

DESCRIPTION

"The compliance statement for SNMP entities which include a proxy forwarding application."

```

MODULE SNMP-TARGET-MIB
    MANDATORY-GROUPS { snmpTargetBasicGroup,
                        snmpTargetResponseGroup }
MODULE -- This Module
    MANDATORY-GROUPS { snmpProxyGroup }
    ::= { snmpProxyCompliances 1 }

snmpProxyGroup OBJECT-GROUP
    OBJECTS {
        snmpProxyType,
        snmpProxyContextEngineID,
        snmpProxyContextName,
        snmpProxyTargetParamsIn,
        snmpProxySingleTargetOut,
        snmpProxyMultipleTargetOut,
        snmpProxyStorageType,
        snmpProxyRowStatus
    }
    STATUS      current
    DESCRIPTION
        "A collection of objects providing remote configuration of
        management target translation parameters for use by
        proxy forwarder applications."

    ::= { snmpProxyGroups 3 }

END

```

5. Identification of Management Targets in Notification Originators

This section describes the mechanisms used by a notification originator application when using the MIB module described in this document to determine the set of management targets to be used when generating a notification.

A notification originator uses the `snmpNotifyTable` to find the management targets to be used for generating notifications. Each active entry in this table identifies zero or more entries in the `snmpTargetAddrTable`. Any entry in the `snmpTargetAddrTable` whose `snmpTargetAddrTagList` object contains a tag value which is equal to a value of `snmpNotifyTag` is selected by the `snmpNotifyEntry` which contains that instance of `snmpNotifyTag`. Note that a particular `snmpTargetAddrEntry` may be selected by multiple entries in the `snmpNotifyTable`, resulting in multiple notifications being generated using that `snmpTargetAddrEntry`.

Each `snmpTargetAddrEntry` contains a pointer to the `snmpTargetParamsTable` (`snmpTargetAddrParams`). This pointer selects a set of SNMP parameters to be used for generating notifications. If the selected entry in the `snmpTargetParamsTable` does not exist, the management target is not used to generate notifications.

The decision as to whether a notification should contain an SNMPv2-Trap or Inform PDU is determined by the value of the `snmpNotifyType` object. If the value of this object is `trap(1)`, the notification should contain an SNMPv2-Trap PDU. If the value of this object is `inform(2)`, then the notification should contain an Inform PDU, and the timeout time and number of retries for the Inform are the value of `snmpTargetAddrTimeout` and `snmpTargetAddrRetryCount`. Note that the exception to these rules is when the `snmpTargetParamsMModel` object indicates SNMPv1. In this case, the notification is sent as a Trap if the value of `snmpNotifyTargetType` is either `trap(1)` or `inform(2)`.

6. Notification Filtering

This section describes the mechanisms used by a notification originator application when using the MIB module described in this document to filter generation of notifications.

A notification originator uses the `snmpNotifyFilterTable` to filter notifications. A notification filter profile may be associated with a particular entry in the `snmpTargetParamsTable`. The associated filter profile is identified by an entry in the `snmpNotifyFilterProfileTable` whose index is equal to the index of the entry in the `snmpTargetParamsTable`. If no such entry exists in the `snmpNotifyFilterProfileTable`, no filtering is performed for that management target.

If such an entry does exist, the value of `snmpNotifyFilterProfileName` of the entry is compared with the corresponding portion of the index of all active entries in the `snmpNotifyFilterTable`. All such entries for which this comparison results in an exact match are used for filtering a notification generated using the associated `snmpTargetParamsEntry`. If no such entries exist, no filtering is performed, and a notification may be sent to the management target.

Otherwise, if matching entries do exist, a notification may be sent if the NOTIFICATION-TYPE OBJECT IDENTIFIER of the notification (this is the value of the element of the variable bindings whose name is `snmpTrapOID.0`, i.e., the second variable binding), and all of the object instances to be included in the variable-bindings of the notification, are not specifically excluded by the matching entries.

Each set of `snmpNotifyFilterTable` entries is divided into two collections of filter subtrees: the included filter subtrees, and the excluded filter subtrees. The `snmpNotifyFilterType` object defines the collection to which each matching entry belongs.

To determine whether a particular notification name or object instance is excluded by the set of matching entries, compare the notification name's or object instance's OBJECT IDENTIFIER with each of the matching entries. If none match, then the notification name or object instance is considered excluded, and the notification should not be sent to this management target. If one or more match, then the notification name or object instance is included or excluded, according to the value of `snmpNotifyFilterType` in the entry whose value of `snmpNotifyFilterSubtree` has the most sub-identifiers. If multiple entries match and have the same number of sub-identifiers, then the lexicographically greatest instance of `snmpNotifyFilterType` among those which match determines the inclusion or exclusion.

A notification name's or object instance's OBJECT IDENTIFIER X matches an entry in the `snmpNotifyFilterTable` when the number of sub-identifiers in X is at least as many as in the value of `snmpNotifyFilterSubtree` for the entry, and each sub-identifier in the value of `snmpNotifyFilterSubtree` matches its corresponding sub-identifier in X. Two sub-identifiers match either if the corresponding bit of `snmpNotifyFilterMask` is zero (the 'wild card' value), or if the two sub-identifiers are equal.

7. Management Target Translation in Proxy Forwarder Applications

This section describes the mechanisms used by a proxy forwarder application when using the MIB module described in this document to translate incoming management target information into outgoing management target information for the purpose of forwarding messages. There are actually two mechanisms a proxy forwarder may use, one for forwarding request messages, and one for forwarding notification messages.

7.1. Management Target Translation for Request Forwarding

When forwarding request messages, the proxy forwarder will select a single entry in the `snmpProxyTable`. To select this entry, it will perform the following comparisons:

- The `snmpProxyType` must be `read(1)` if the request is a `Get`, `GetNext`, or `GetBulk` request. The `snmpProxyType` must be `write(2)` if the request is a `Set` request.

- The contextEngineId must equal the snmpProxyContextEngineID object.
- If the snmpProxyContextName object is supported, it must equal the contextName.
- The snmpProxyTargetParamsIn object identifies an entry in the snmpTargetParamsTable. The messageProcessingModel, securityLevel, security model, and securityName must match the values of snmpTargetParamsMPModel, snmpTargetParamsSecurityModel, snmpTargetParamsSecurityName, and snmpTargetParamsSecurityLevel of the identified entry in the snmpTargetParamsTable.

There may be multiple entries in the snmpProxyTable for which these comparisons succeed. The entry whose snmpProxyName has the lexicographically smallest value and for which the comparisons succeed will be selected by the proxy forwarder.

The outgoing management target information is identified by the value of the snmpProxySingleTargetOut object of the selected entry. This object identifies an entry in the snmpTargetAddrTable. The identified entry in the snmpTargetAddrTable also contains a reference to the snmpTargetParamsTable (snmpTargetAddrParams). If either the identified entry in the snmpTargetAddrTable does not exist, or the identified entry in the snmpTargetParamsTable does not exist, then this snmpProxyEntry does not identify valid forwarding information, and the proxy forwarder should attempt to identify another row.

If there is no entry in the snmpProxyTable for which all of the conditions above may be met, then there is no appropriate forwarding information, and the proxy forwarder should take appropriate actions.

Otherwise, The snmpTargetAddrTDomain, snmpTargetAddrTAddress, snmpTargetAddrTimeout, and snmpTargetRetryCount of the identified snmpTargetAddrEntry, and the snmpTargetParamsMPModel, snmpTargetParamsSecurityModel, snmpTargetParamsSecurityName, and snmpTargetParamsSecurityLevel of the identified snmpTargetParamsEntry are used as the destination management target.

7.2. Management Target Translation for Notification Forwarding

When forwarding notification messages, the proxy forwarder will select multiple entries in the snmpProxyTable. To select these entries, it will perform the following comparisons:

- The `snmpProxyType` must be `trap(3)` if the notification is a Trap. The `snmpProxyType` must be `inform(4)` if the request is an Inform.
- The `contextEngineId` must equal the `snmpProxyContextEngineID` object.
- If the `snmpProxyContextName` object is supported, it must equal the `contextName`.
- The `snmpProxyTargetParamsIn` object identifies an entry in the `snmpTargetParamsTable`. The `messageProcessingModel`, `securityLevel`, `security model`, and `securityName` must match the values of `snmpTargetParamsMPModel`, `snmpTargetParamsSecurityModel`, `snmpTargetParamsSecurityName`, and `snmpTargetParamsSecurityLevel` of the identified entry in the `snmpTargetParamsTable`.

All entries for which these conditions are met are selected. The `snmpProxyMultipleTargetOut` object of each such entry is used to select a set of entries in the `snmpTargetAddrTable`. Any `snmpTargetAddrEntry` whose `snmpTargetAddrTagList` object contains a tag value equal to the value of `snmpProxyMultipleTargetOut`, and whose `snmpTargetAddrParams` object references an existing entry in the `snmpTargetParamsTable`, is selected as a destination for the forwarded notification.

8. 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.

9. Acknowledgments

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

Dave Battle (SNMP Research, Inc.)
Uri Blumenthal (IBM T.J. Watson Research Center)
Jeff Case (SNMP Research, Inc.)
John Curran (BBN)
T. Max Devlin (Hi-TECH Connections)
John Flick (Hewlett Packard)
David Harrington (Cabletron Systems Inc.)
N.C. Hien (IBM T.J. Watson Research Center)
Dave Levi (SNMP Research, Inc.)
Louis A Mamakos (UUNET Technologies Inc.)
Paul Meyer (Secure Computing Corporation)
Keith McCloghrie (Cisco Systems)
Russ Mundy (Trusted Information Systems, Inc.)
Bob Natale (ACE*COMM Corporation)
Mike O'Dell (UUNET Technologies Inc.)
Dave Perkins (DeskTalk)
Peter Polkinghorne (Brunel University)
Randy Presuhn (BMC Software, Inc.)
David Reid (SNMP Research, Inc.)
Shawn Routhier (Epilogue)
Juergen Schoenwaelder (TU Braunschweig)
Bob Stewart (Cisco Systems)
Bert Wijnen (IBM T.J. Watson Research Center)

The document is based on recommendations of the IETF Security and Administrative Framework Evolution for SNMP Advisory Team. Members of that Advisory Team were:

David Harrington (Cabletron Systems Inc.)
Jeff Johnson (Cisco Systems)
David Levi (SNMP Research Inc.)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (IBM T. J. Watson Research Center)

As recommended by the Advisory Team and the SNMPv3 Working Group Charter, the design incorporates as much as practical from previous RFCs and drafts. As a result, special thanks are due to the authors of previous designs known as SNMPv2u and SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Cabletron Systems Inc.)
David Levi (SNMP Research, Inc.)
Keith McCloghrie (Cisco Systems)
Brian O'Keefe (Hewlett Packard)
Marshall T. Rose (Dover Beach Consulting)
Jon Saperia (BGS Systems Inc.)
Steve Waldbusser (International Network Services)
Glenn W. Waters (Bell-Northern Research Ltd.)

10. Security Considerations

The SNMP applications described in this document typically have direct access to MIB instrumentation. Thus, it is very important that these applications be strict in their application of access control as described in this document.

In addition, there may be some types of notification generator applications which, rather than accessing MIB instrumentation using access control, will obtain MIB information through other means (such as from a command line). The implementors and users of such applications must be responsible for not divulging MIB information that normally would be inaccessible due to access control.

11. References

[RFC1157]

Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", RFC 1157, May 1990.

[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.

[RFC1902]

Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1902, January 1996.

[RFC1903]

Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1903, January 1996.

- [RFC1905]
SNMPv2 Working Group, 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.
- [RFC1907]
SNMPv2 Working Group, 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.
- [RFC1908]
SNMPv2 Working Group, 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.
- [RFC2271]
Harrington, D., and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2271, January 1998.
- [RFC2272]
Case, J., Harrington, D., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2272, January 1998.
- [RFC2275]
Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2275, January 1998.

12. Editors' Addresses

David B. Levi
SNMP Research, Inc.
3001 Kimberlin Heights Road
Knoxville, TN 37920-9716
U.S.A.

Phone: +1 423 573 1434
EMail: levi@snmp.com

Paul Meyer
Secure Computing Corporation
2675 Long Lake Road
Roseville, MN 55113
U.S.A.

Phone: +1 612 628 1592
EMail: paul_meyer@securecomputing.com

Bob Stewart
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
U.S.A.

Phone: +1 603 654 6923
EMail: bstewart@cisco.com

APPENDIX A - Trap Configuration Example

This section describes an example configuration for a Notification Generator application which implements the `snmpNotifyBasicCompliance` level. The example configuration specifies that the Notification Generator should send notifications to 3 separate managers, using authentication and no privacy for the first 2 managers, and using both authentication and privacy for the third manager.

The configuration consists of three rows in the `snmpTargetAddrTable`, and two rows in the `snmpTargetTable`.

<code>snmpTargetAddrName</code>	<code>SnmpAdminString,</code>
<code>snmpTargetAddrTDomain</code>	<code>TDomain,</code>
<code>snmpTargetAddrTAddress</code>	<code>TAddress,</code>
<code>snmpTargetAddrTimeout</code>	<code>TimeInterval,</code>
<code>snmpTargetAddrRetryCount</code>	<code>Integer32,</code>
<code>snmpTargetAddrTagList</code>	<code>SnmpAdminString,</code>
<code>snmpTargetAddrParams</code>	<code>SnmpAdminString,</code>
<code>snmpTargetAddrStorageType</code>	<code>StorageType,</code>
<code>snmpTargetAddrRowStatus</code>	<code>RowStatus</code>
* <code>snmpTargetAddrName</code> = "addr1"	
<code>snmpTargetAddrTDomain</code>	= <code>snmpUDPDomain</code>
<code>snmpTargetAddrTAddress</code>	= 128.1.2.3:162
<code>snmpTargetAddrTagList</code>	= "group1"
<code>snmpTargetAddrParams</code>	= "AuthNoPriv joe"
<code>snmpTargetAddrStorageType</code>	= <code>readOnly(5)</code>
<code>snmpTargetAddrRowStatus</code>	= <code>active(1)</code>
* <code>snmpTargetAddrName</code> = "addr2"	
<code>snmpTargetAddrTDomain</code>	= <code>snmpUDPDomain</code>
<code>snmpTargetAddrTAddress</code>	= 128.2.4.6:162
<code>snmpTargetAddrTagList</code>	= "group1"
<code>snmpTargetAddrParams</code>	= "AuthNoPriv-joe"
<code>snmpTargetAddrStorageType</code>	= <code>readOnly(5)</code>
<code>snmpTargetAddrRowStatus</code>	= <code>active(1)</code>
* <code>snmpTargetAddrName</code> = "addr3"	
<code>snmpTargetAddrTDomain</code>	= <code>snmpUDPDomain</code>
<code>snmpTargetAddrTAddress</code>	= 128.1.2.3:162
<code>snmpTargetAddrTagList</code>	= "group2"
<code>snmpTargetAddrParams</code>	= "AuthPriv-bob"
<code>snmpTargetAddrStorageType</code>	= <code>readOnly(5)</code>
<code>snmpTargetAddrRowStatus</code>	= <code>active(1)</code>
* <code>snmpTargetParamsName</code> = "AuthNoPriv-joe"	
<code>snmpTargetParamsMPModel</code>	= 3

```

snmpTargetParamsSecurityModel      = 3 (USM)
snmpTargetParamsSecurityName       = "joe"
snmpTargetParamsSecurityLevel      = authNoPriv(2)
snmpTargetParamsStorageType        = readOnly(5)
snmpTargetParamsRowStatus          = active(1)

* snmpTargetParamsName              = "AuthPriv-bob"
  snmpTargetParamsMModel            = 3
  snmpTargetParamsSecurityModel     = 3 (USM)
  snmpTargetParamsSecurityName      = "bob"
  snmpTargetParamsSecurityLevel     = authPriv(3)
  snmpTargetParamsStorageType       = readOnly(5)
  snmpTargetParamsRowStatus         = active(1)

* snmpNotifyName                    = "group1"
  snmpNotifyTag                     = "group1"
  snmpNotifyType                    = trap(1)
  snmpNotifyStorageType             = readOnly(5)
  snmpNotifyRowStatus              = active(1)

* snmpNotifyName                    = "group2"
  snmpNotifyTag                     = "group2"
  snmpNotifyType                    = trap(1)
  snmpNotifyStorageType             = readOnly(5)
  snmpNotifyRowStatus              = active(1)

```

These entries define two groups of management targets. The first group contains two management targets:

	first target -----	second target -----
messageProcessingModel	SNMPv3	SNMPv3
securityModel	3 (USM)	3 (USM)
securityName	"joe"	"joe"
securityLevel	authNoPriv(2)	authNoPriv(2)
transportDomain	snmpUDPDomain	snmpUDPDomain
transportAddress	128.1.2.3:162	128.2.4.6:162

And the second group contains a single management target:

messageProcessingModel	SNMPv3
securityLevel	authPriv(3)
securityModel	3 (USM)
securityName	"bob"
transportDomain	snmpUDPDomain
transportAddress	128.1.5.9:162

B. Full Copyright Statement

Copyright (C) The Internet Society (1998). 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.

