Network Working Group
Request for Comments: 2261
Category: Standards Track

D. Harrington
Cabletron Systems, Inc.
R. Presuhn
BMC Software, Inc.
B. Wijnen
IBM T. J. Watson Research

January 1998

An Architecture for Describing SNMP Management Frameworks

Status of this Memo

Copyright Notice

Abstract

This document describes an architecture for describing SNMP Management Frameworks. The architecture is designed to be modular to allow the evolution of the SNMP protocol standards over time. The major portions of the architecture are an SNMP engine containing a Message Processing Subsystem, a Security Subsystem and an Access Control Subsystem, and possibly multiple SNMP applications which provide specific functional processing of management data.

Table of Contents

1. Introduction

1.1. Overview

This document defines a vocabulary for describing SNMP Management Frameworks, and an architecture for describing the major portions of SNMP Management Frameworks.

This document does not provide a general introduction to SNMP. Other documents and books can provide a much better introduction to SNMP. Nor does this document provide a history of SNMP. That also can be found in books and other documents.

Section 1 describes the purpose, goals, and design decisions of this architecture.

Section 2 describes various types of documents which define SNMP Frameworks, and how they fit into this architecture. It also provides a minimal road map to the documents which have previously defined SNMP frameworks.

Section 3 details the vocabulary of this architecture and its pieces. This section is important for understanding the remaining sections, and for understanding documents which are written to fit within this architecture.

Section 4 describes the primitives used for the abstract service interfaces between the various subsystems, models and applications within this architecture.

Section 5 defines a collection of managed objects used to instrument SNMP entities within this architecture.

Sections 6, 7, 8, and 9 are administrative in nature.

Appendix A contains guidelines for designers of Models which are expected to fit within this architecture.

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

1.2. SNMP

An SNMP management system contains:

- several (potentially many) nodes, each with an SNMP entity containing command responder and notification originator applications, which have access to management instrumentation (traditionally called agents);

- at least one SNMP entity containing command generator and/or notification receiver applications (traditionally called a manager) and,

- a management protocol, used to convey management information between the SNMP entities.

SNMP entities executing command generator and notification receiver applications monitor and control managed elements. Managed elements are devices such as hosts, routers, terminal servers, etc., which are monitored and controlled via access to their management information.

It is the purpose of this document to define an architecture which can evolve to realize effective management in a variety of configurations and environments. The architecture has been designed to meet the needs of implementations of:

- minimal SNMP entities with command responder and/or notification originator applications (traditionally called SNMP agents),

- SNMP entities with proxy forwarder applications (traditionally called SNMP proxy agents),

- command line driven SNMP entities with command generator and/or notification receiver applications (traditionally called SNMP command line managers),

- SNMP entities with command generator and/or notification receiver, plus command responder and/or notification originator applications (traditionally called SNMP mid-level managers or dual-role entities),

- SNMP entities with command generator and/or notification receiver and possibly other types of applications for managing a potentially very large number of managed nodes (traditionally called (network) management stations).

1.3. Goals of this Architecture

This architecture was driven by the following goals:

- Use existing materials as much as possible. It is heavily based on previous work, informally known as SNMPv2u and SNMPv2*.

- Address the need for secure SET support, which is considered the most important deficiency in SNMPv1 and SNMPv2c.

- Make it possible to move portions of the architecture forward in the standards track, even if consensus has not been reached on all pieces.

- Define an architecture that allows for longevity of the SNMP Frameworks that have been and will be defined.

- Keep SNMP as simple as possible.

- Make it relatively inexpensive to deploy a minimal conforming implementation.

- Make it possible to upgrade portions of SNMP as new approaches become available, without disrupting an entire SNMP framework.

- Make it possible to support features required in large networks, but make the expense of supporting a feature directly related to the support of the feature.

1.4. Security Requirements of this Architecture

Several of the classical threats to network protocols are applicable to the management problem and therefore would be applicable to any Security Model used in an SNMP Management Framework. Other threats are not applicable to the management problem. This section discusses principal threats, secondary threats, and threats which are of lesser importance.

The principal threats against which any Security Model used within this architecture SHOULD provide protection are:

Modification of Information
: The modification threat is the danger that some unauthorized SNMP entity may alter in-transit SNMP messages generated on behalf of an authorized principal in such a way as to effect unauthorized management operations, including falsifying the value of an object.

Masquerade
: The masquerade threat is the danger that management operations not authorized for some principal may be attempted by assuming the identity of another principal that has the appropriate authorizations.

Message Stream Modification
: The SNMP protocol is typically based upon a connectionless transport service which may operate over any subnetwork service. The re-ordering, delay or replay of messages can and does occur through the natural operation of many such subnetwork services. The message stream modification threat is the danger that messages

may be maliciously re-ordered, delayed or replayed to an extent which is greater than can occur through the natural operation of a subnetwork service, in order to effect unauthorized management operations.

Disclosure
The disclosure threat is the danger of eavesdropping on the exchanges between SNMP engines. Protecting against this threat may be required as a matter of local policy.

There are at least two threats against which a Security Model within this architecture need not protect.

Denial of Service
A Security Model need not attempt to address the broad range of attacks by which service on behalf of authorized users is denied. Indeed, such denial-of-service attacks are in many cases indistinguishable from the type of network failures with which any viable management protocol must cope as a matter of course.

Traffic Analysis
A Security Model need not attempt to address traffic analysis attacks. Many traffic patterns are predictable - entities may be managed on a regular basis by a relatively small number of management stations - and therefore there is no significant advantage afforded by protecting against traffic analysis.

1.5. Design Decisions

Various design decisions were made in support of the goals of the architecture and the security requirements:

- Architecture
  An architecture should be defined which identifies the conceptual boundaries between the documents. Subsystems should be defined which describe the abstract services provided by specific portions of an SNMP framework. Abstract service interfaces, as described by service primitives, define the abstract boundaries between documents, and the abstract services that are provided by the conceptual subsystems of an SNMP framework.

- Self-contained Documents
  Elements of procedure plus the MIB objects which are needed for processing for a specific portion of an SNMP framework should be defined in the same document, and as much as possible, should not be referenced in other documents. This allows pieces to be designed and documented as independent and self-contained

parts, which is consistent with the general SNMP MIB module approach. As portions of SNMP change over time, the documents describing other portions of SNMP are not directly impacted. This modularity allows, for example, Security Models, authentication and privacy mechanisms, and message formats to be upgraded and supplemented as the need arises. The self-contained documents can move along the standards track on different time-lines.

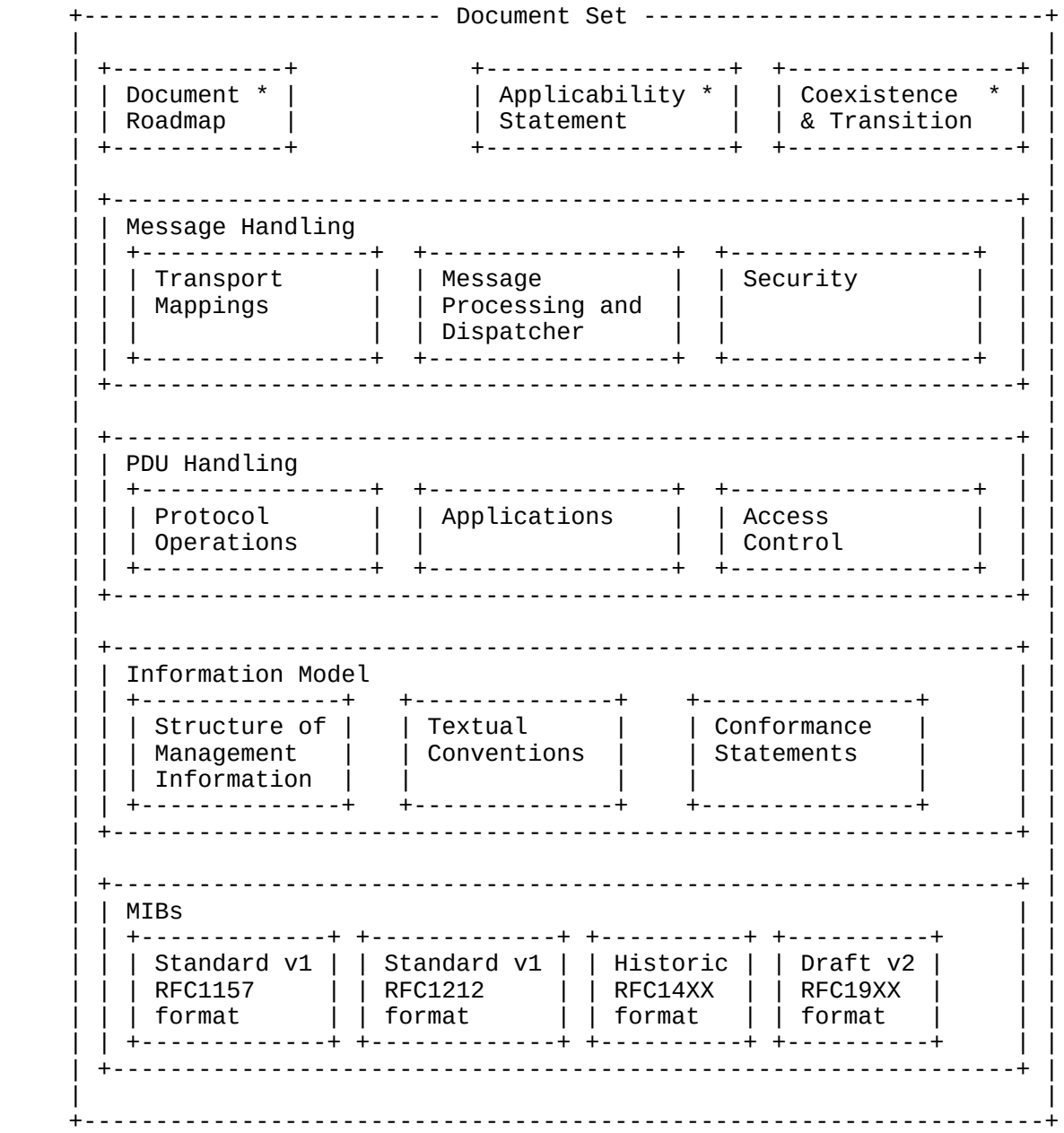
- Threats
  The Security Models in the Security Subsystem SHOULD protect against the principal threats: modification of information, masquerade, message stream modification and disclosure. They do not need to protect against denial of service and traffic analysis.

- Remote Configuration
  The Security and Access Control Subsystems add a whole new set of SNMP configuration parameters. The Security Subsystem also requires frequent changes of secrets at the various SNMP entities. To make this deployable in a large operational environment, these SNMP parameters must be able to be remotely configured.

- Controlled Complexity
  It is recognized that producers of simple managed devices want to keep the resources used by SNMP to a minimum. At the same time, there is a need for more complex configurations which can spend more resources for SNMP and thus provide more functionality. The design tries to keep the competing requirements of these two environments in balance and allows the more complex environments to logically extend the simple environment.

2. Documentation Overview

The following figure shows the set of documents that fit within the SNMP Architecture.



Note: RFC14XX means RFCs 1442, 1443, and 1444. RFC19XX means RFCs 1902, 1903, and 1904.

Those marked with an asterisk (*) are expected to be written in the future. Each of these documents may be replaced or supplemented. This Architecture document specifically describes how new documents fit into the set of documents in the area of Message and PDU handling.

2.1. Document Roadmap

One or more documents may be written to describe how sets of documents taken together form specific Frameworks. The configuration of document sets might change over time, so the "road map" should be maintained in a document separate from the standards documents themselves.

2.2. Applicability Statement

SNMP is used in networks that vary widely in size and complexity, by organizations that vary widely in their requirements of management. Some models will be designed to address specific problems of management, such as message security.

One or more documents may be written to describe the environments to which certain versions of SNMP or models within SNMP would be appropriately applied, and those to which a given model might be inappropriately applied.

2.3. Coexistence and Transition

The purpose of an evolutionary architecture is to permit new models to replace or supplement existing models. The interactions between models could result in incompatibilities, security "holes", and other undesirable effects.

The purpose of Coexistence documents is to detail recognized anomalies and to describe required and recommended behaviors for resolving the interactions between models within the architecture.

Coexistence documents may be prepared separately from model definition documents, to describe and resolve interaction anomalies between a model definition and one or more other model definitions.

Additionally, recommendations for transitions between models may also be described, either in a coexistence document or in a separate document.

2.4. Transport Mappings

SNMP messages are sent over various transports. It is the purpose of Transport Mapping documents to define how the mapping between SNMP and the transport is done.

2.5. Message Processing

A Message Processing Model document defines a message format, which is typically identified by a version field in an SNMP message header. The document may also define a MIB module for use in message processing and for instrumentation of version-specific interactions.

An SNMP engine includes one or more Message Processing Models, and thus may support sending and receiving multiple versions of SNMP messages.

2.6. Security

Some environments require secure protocol interactions. Security is normally applied at two different stages:

- in the transmission/receipt of messages, and
- in the processing of the contents of messages.

For purposes of this document, "security" refers to message-level security; "access control" refers to the security applied to protocol operations.

Authentication, encryption, and timeliness checking are common functions of message level security.

A security document describes a Security Model, the threats against which the model protects, the goals of the Security Model, the protocols which it uses to meet those goals, and it may define a MIB module to describe the data used during processing, and to allow the remote configuration of message-level security parameters, such as passwords.

An SNMP engine may support multiple Security Models concurrently.

2.7. Access Control

During processing, it may be required to control access to managed objects for operations.

An Access Control Model defines mechanisms to determine whether access to a managed object should be allowed. An Access Control Model may define a MIB module used during processing and to allow the remote configuration of access control policies.

2.8. Protocol Operations

SNMP messages encapsulate an SNMP Protocol Data Unit (PDU). It is the purpose of a Protocol Operations document to define the operations of the protocol with respect to the processing of the PDUs.

An application document defines which Protocol Operations documents are supported by the application.

2.9. Applications

An SNMP entity normally includes a number of applications. Applications use the services of an SNMP engine to accomplish specific tasks. They coordinate the processing of management information operations, and may use SNMP messages to communicate with other SNMP entities.

Applications documents describe the purpose of an application, the services required of the associated SNMP engine, and the protocol operations and informational model that the application uses to perform management operations.

An application document defines which set of documents are used to specifically define the structure of management information, textual conventions, conformance requirements, and operations supported by the application.

2.10. Structure of Management Information

Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules.

It is the purpose of a Structure of Management Information document to establish the syntax for defining objects, modules, and other elements of managed information.

2.11. Textual Conventions

When designing a MIB module, it is often useful to define new types similar to those defined in the SMI, but with more precise semantics, or which have special semantics associated with them. These newly defined types are termed textual conventions, and may defined in separate documents, or within a MIB module.

2.12. Conformance Statements

It may be useful to define the acceptable lower-bounds of implementation, along with the actual level of implementation achieved. It is the purpose of Conformance Statements to define the notation used for these purposes.

2.13. Management Information Base Modules

MIB documents describe collections of managed objects which instrument some aspect of a managed node.

2.13.1. SNMP Instrumentation MIBs

An SNMP MIB document may define a collection of managed objects which instrument the SNMP protocol itself. In addition, MIB modules may be defined within the documents which describe portions of the SNMP architecture, such as the documents for Message processing Models, Security Models, etc. for the purpose of instrumenting those Models, and for the purpose of allowing remote configuration of the Model.

2.14. SNMP Framework Documents

This architecture is designed to allow an orderly evolution of portions of SNMP Frameworks.

Throughout the rest of this document, the term "subsystem" refers to an abstract and incomplete specification of a portion of a Framework, that is further refined by a model specification.

A "model" describes a specific design of a subsystem, defining additional constraints and rules for conformance to the model. A model is sufficiently detailed to make it possible to implement the specification.

An "implementation" is an instantiation of a subsystem, conforming to one or more specific models.

SNMP version 1 (SNMPv1), is the original Internet-standard Network Management Framework, as described in RFCs 1155, 1157, and 1212.

SNMP version 2 (SNMPv2), is the SNMPv2 Framework as derived from the SNMPv1 Framework. It is described in RFCs 1902-1907. SNMPv2 has no message definition.

The Community-based SNMP version 2 (SNMPv2c), is an experimental SNMP Framework which supplements the SNMPv2 Framework, as described in RFC1901. It adds the SNMPv2c message format, which is similar to the

SNMPv1 message format.

SNMP version 3 (SNMPv3), is an extensible SNMP Framework which supplements the SNMPv2 Framework, by supporting the following:

- a new SNMP message format,
- Security for Messages, and
- Access Control.

Other SNMP Frameworks, i.e., other configurations of implemented subsystems, are expected to also be consistent with this architecture.

3. Elements of the Architecture

This section describes the various elements of the architecture and how they are named. There are three kinds of naming:

1) the naming of entities,

2) the naming of identities, and

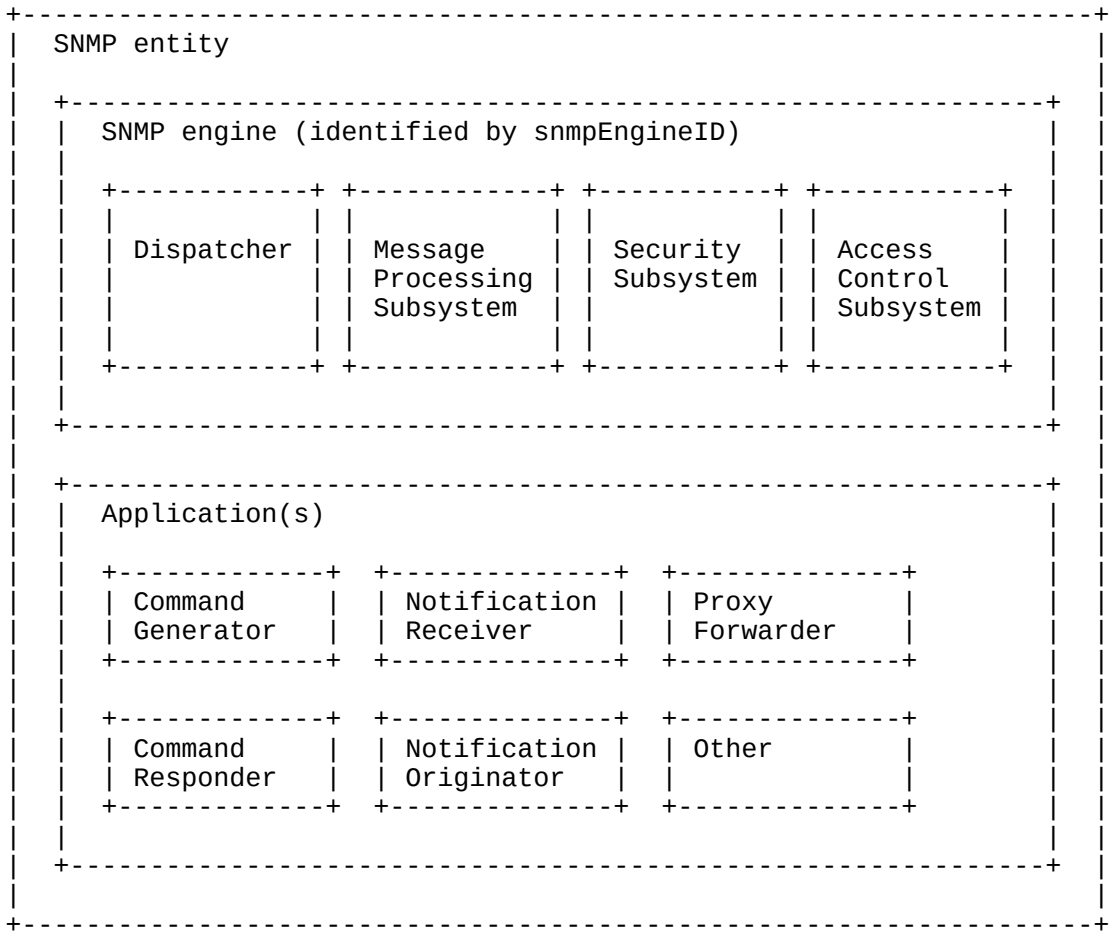
3) the naming of management information.

This architecture also defines some names for other constructs that are used in the documentation.

3.1. The Naming of Entities

An SNMP entity is an implementation of this architecture. Each such SNMP entity consists of an SNMP engine and one or more associated applications.

The following figure shows details about an SNMP entity and the components within it.



3.1.1. SNMP engine

An SNMP engine provides services for sending and receiving messages, authenticating and encrypting messages, and controlling access to managed objects. There is a one-to-one association between an SNMP engine and the SNMP entity which contains it.

The engine contains:

1) a Dispatcher,

2) a Message Processing Subsystem,

3) a Security Subsystem, and

4) an Access Control Subsystem.

3.1.1.1. snmpEngineID

Within an administrative domain, an snmpEngineID is the unique and unambiguous identifier of an SNMP engine. Since there is a one-to-one association between SNMP engines and SNMP entities, it also uniquely and unambiguously identifies the SNMP entity.

3.1.1.2. Dispatcher

There is only one Dispatcher in an SNMP engine. It allows for concurrent support of multiple versions of SNMP messages in the SNMP engine. It does so by:

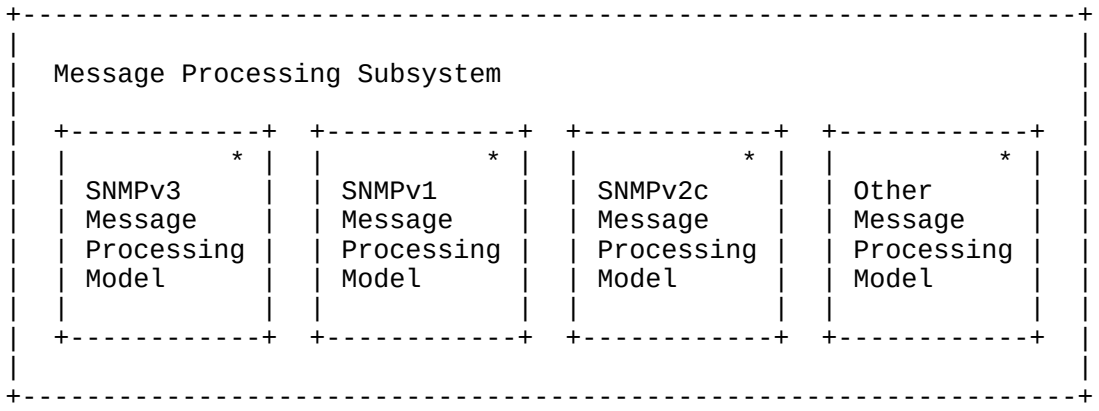
- sending and receiving SNMP messages to/from the network,
- determining the version of an SNMP message and interacting with the corresponding Message Processing Model,
- providing an abstract interface to SNMP applications for delivery of a PDU to an application.
- providing an abstract interface for SNMP applications that allows them to send a PDU to a remote SNMP entity.

3.1.1.3. Message Processing Subsystem

The Message Processing Subsystem is responsible for preparing messages for sending, and extracting data from received messages.

The Message Processing Subsystem potentially contains multiple Message Processing Models as shown in the next figure.

* One or more Message Processing Models may be present.



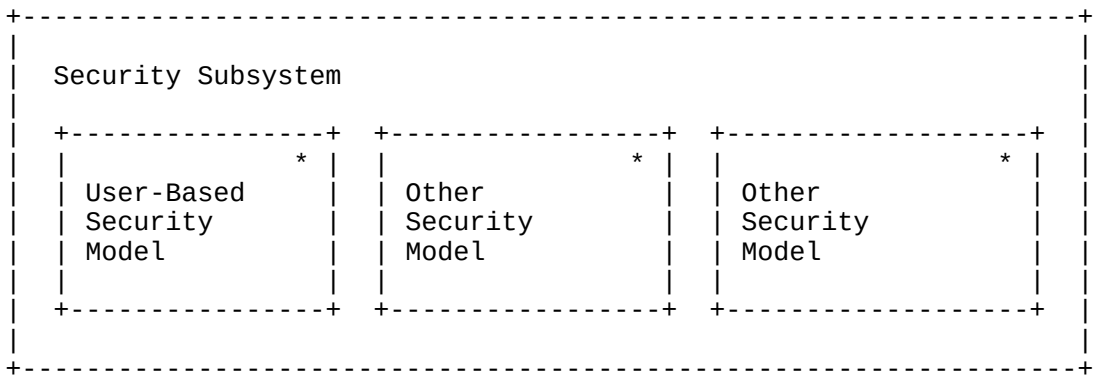
3.1.1.3.1. Message Processing Model

Each Message Processing Model defines the format of a particular version of an SNMP message and coordinates the preparation and extraction of each such version-specific message format.

3.1.1.4. Security Subsystem

The Security Subsystem provides security services such as the authentication and privacy of messages and potentially contains multiple Security Models as shown in the following figure

* One or more Security Models may be present.



3.1.1.4.1. Security Model

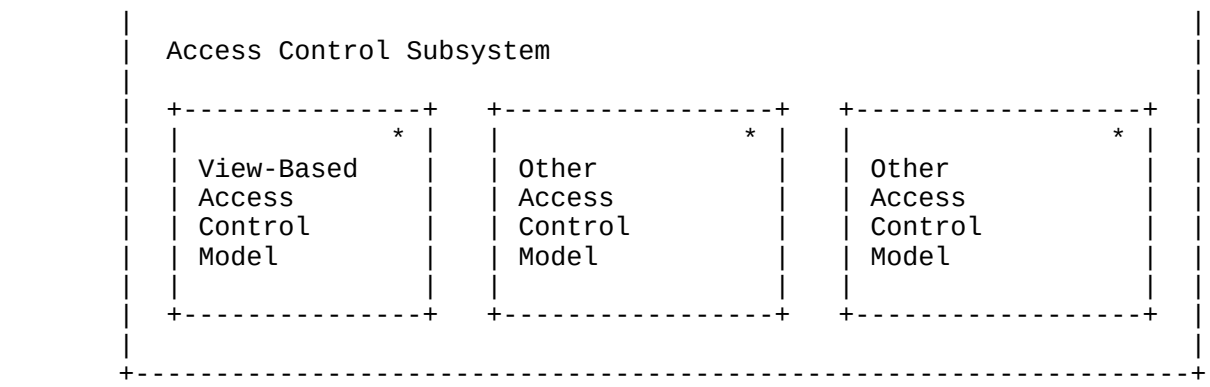
A Security Model defines the threats against which it protects, the goals of its services, and the security protocols used to provide security services such as authentication and privacy.

3.1.1.4.2. Security Protocol

A Security Protocol defines the mechanisms, procedures, and MIB data used to provide a security service such as authentication or privacy.

3.1.2. Access Control Subsystem

The Access Control Subsystem provides authorization services by means of one or more Access Control Models.



3.1.2.1. Access Control Model

An Access Control Model defines a particular access decision function in order to support decisions regarding access rights.

3.1.3. Applications

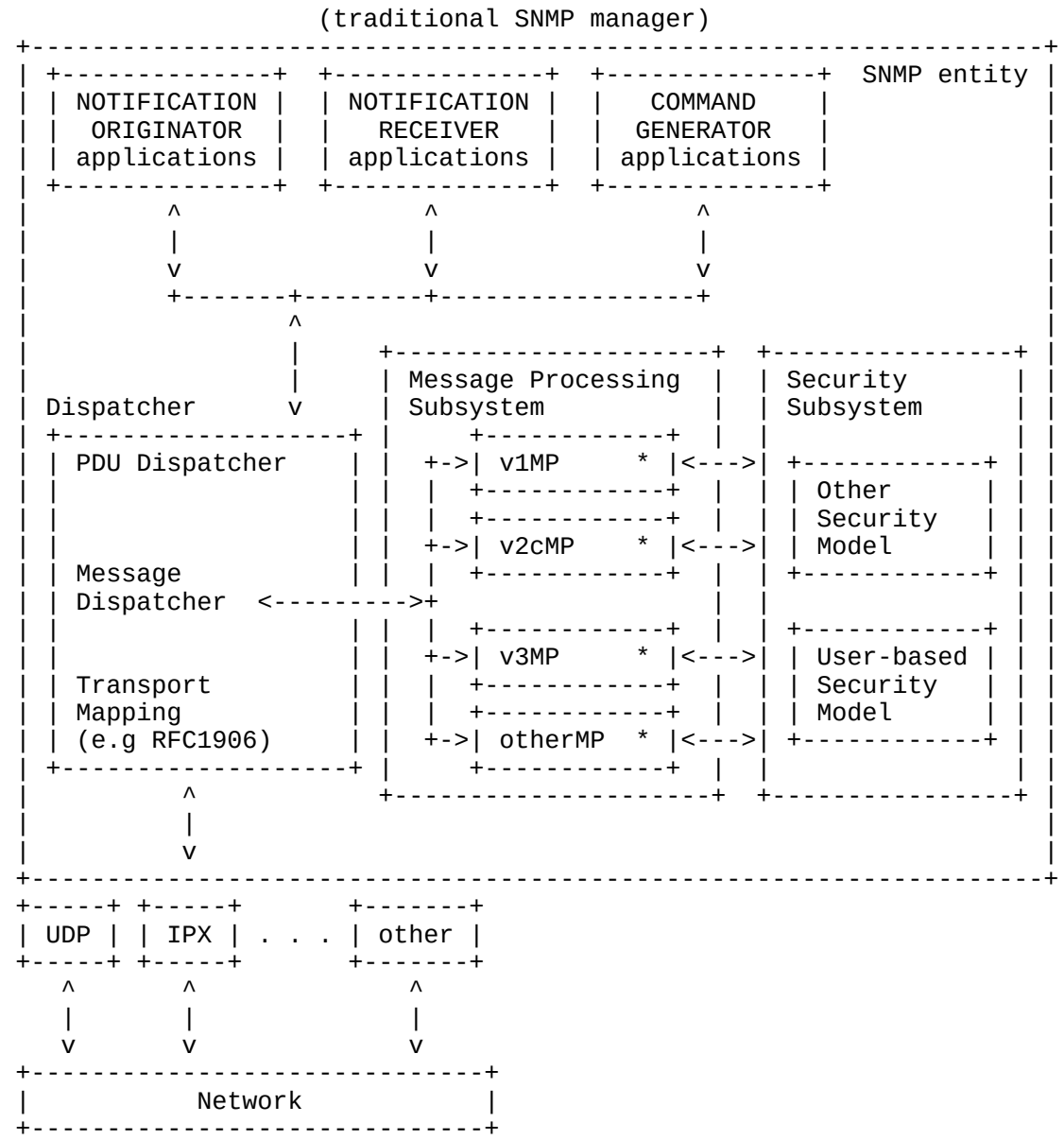
There are several types of applications, including:

- command generators, which monitor and manipulate management data,
- command responders, which provide access to management data,
- notification originators, which initiate asynchronous messages,
- notification receivers, which process asynchronous messages, and
- proxy forwarders, which forward messages between entities.

These applications make use of the services provided by the SNMP engine.

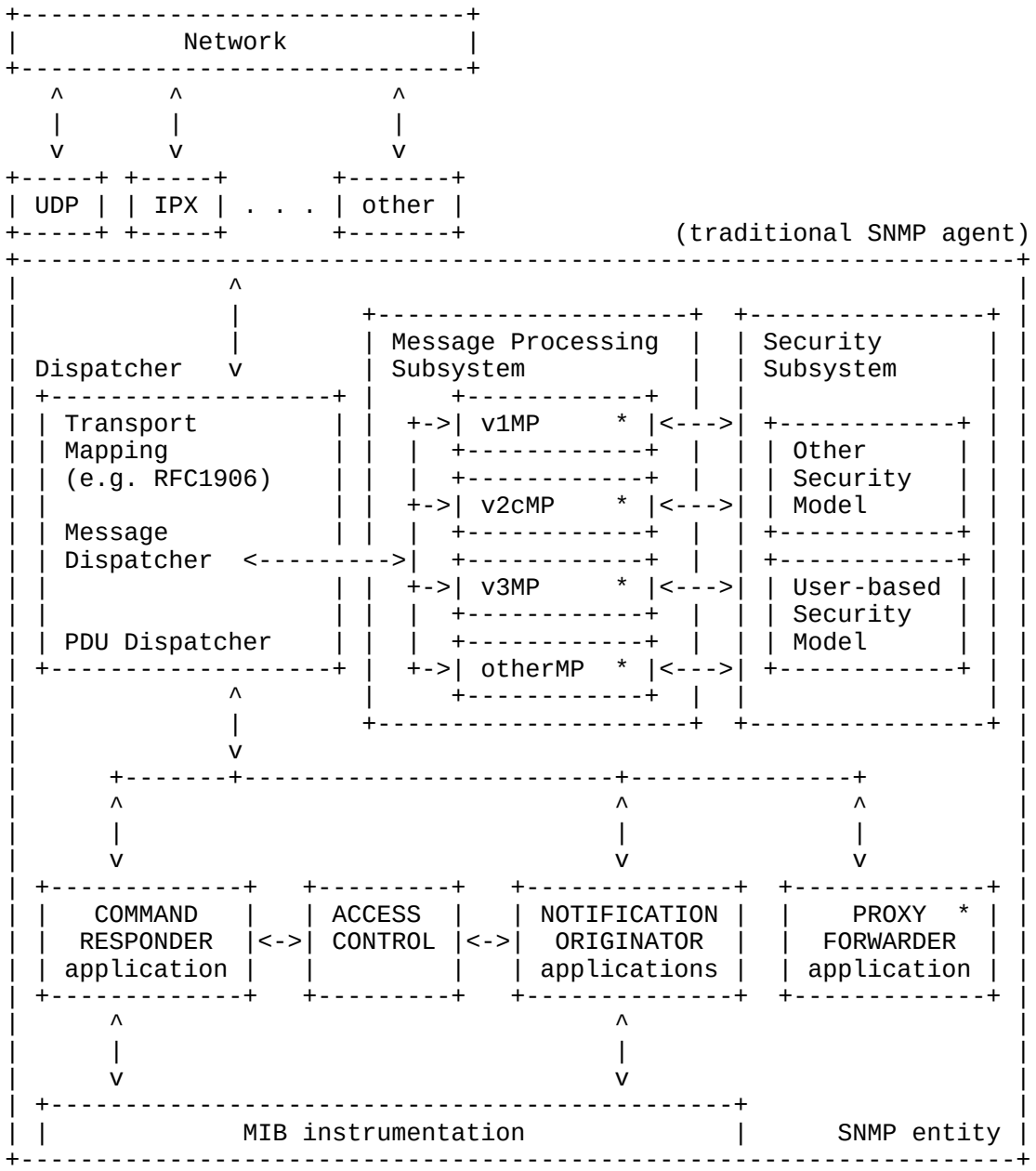
3.1.3.1. SNMP Manager

An SNMP entity containing one or more command generator and/or notification receiver applications (along with their associated SNMP engine) has traditionally been called an SNMP manager. * One or more models may be present.

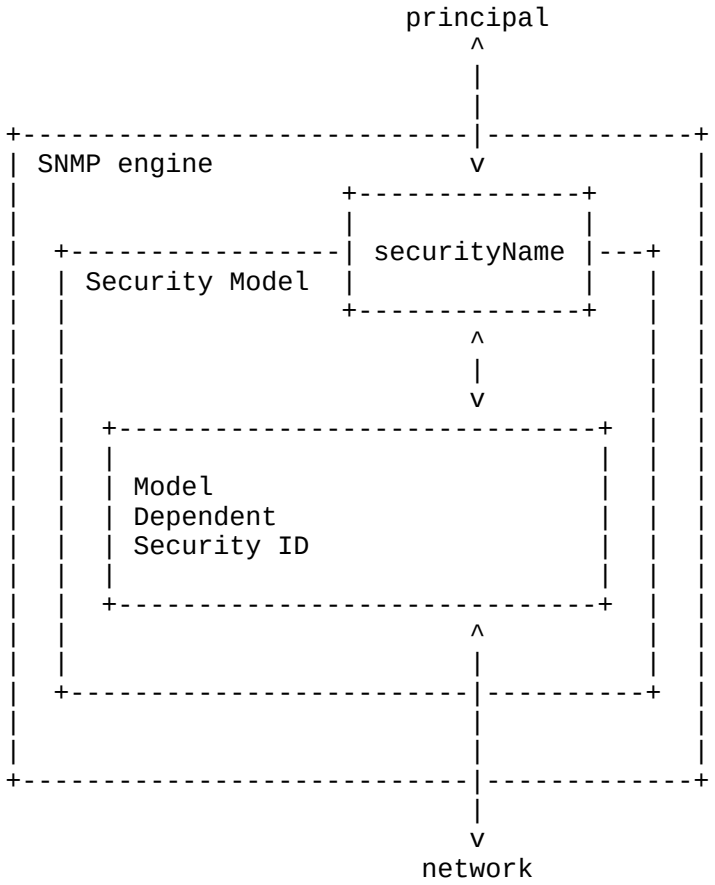


3.1.3.2. SNMP Agent

An SNMP entity containing one or more command responder and/or notification originator applications (along with their associated SNMP engine) has traditionally been called an SNMP agent.



3.2. The Naming of Identities



3.2.1. Principal

A principal is the "who" on whose behalf services are provided or processing takes place.

A principal can be, among other things, an individual acting in a particular role; a set of individuals, with each acting in a particular role; an application or a set of applications; and combinations thereof.

3.2.2. securityName

A securityName is a human readable string representing a principal. It has a model-independent format, and can be used outside a particular Security Model.

3.2.3. Model-dependent security ID

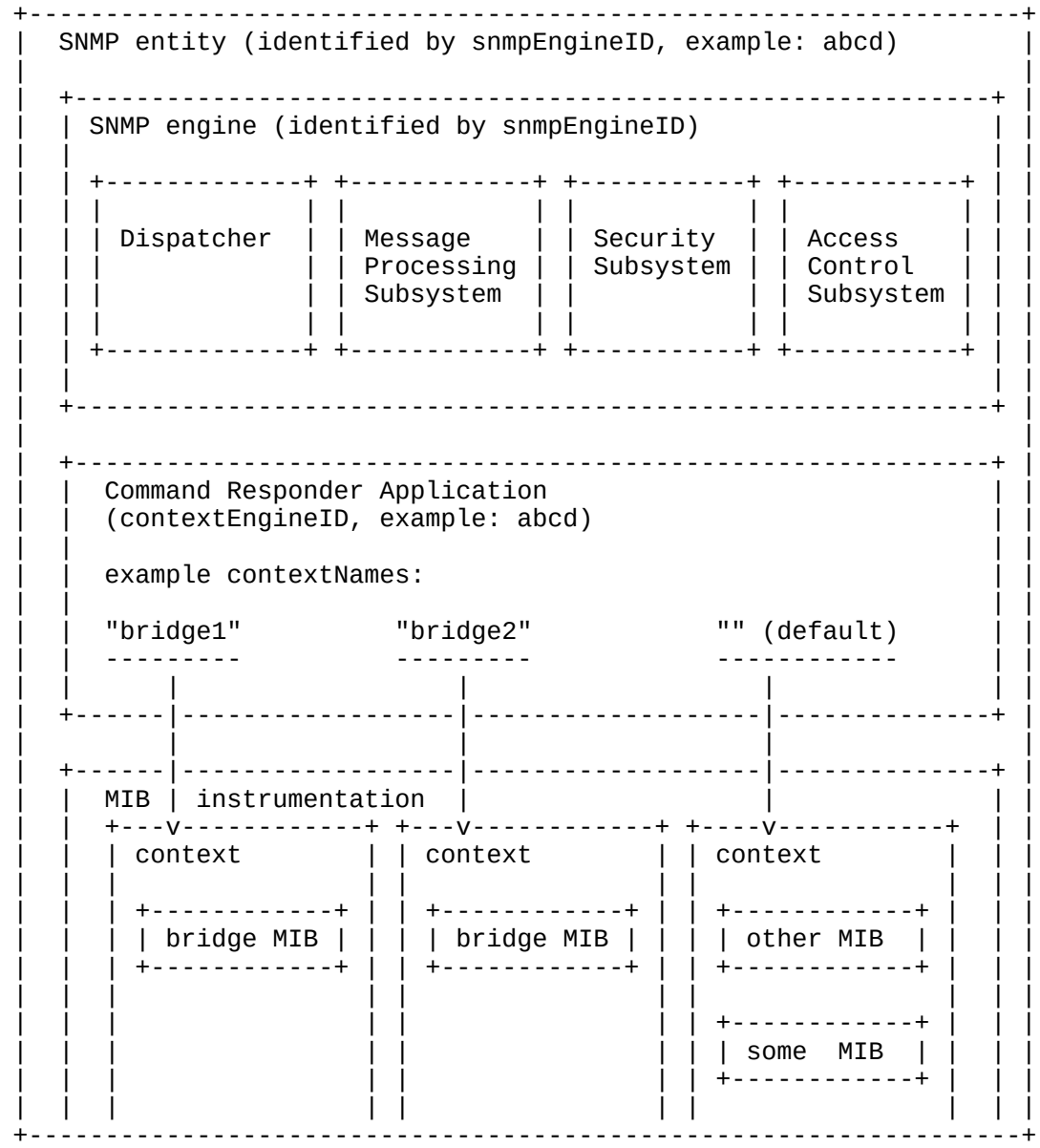
A model-dependent security ID is the model-specific representation of a securityName within a particular Security Model.

Model-dependent security IDs may or may not be human readable, and have a model-dependent syntax. Examples include community names, user names, and parties.

The transformation of model-dependent security IDs into securityNames and vice versa is the responsibility of the relevant Security Model.

3.3. The Naming of Management Information

Management information resides at an SNMP entity where a Command Responder Application has local access to potentially multiple contexts. This application uses a contextEngineID equal to the snmpEngineID of its associated SNMP engine.



3.3.1. An SNMP Context

An SNMP context, or just "context" for short, is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context. An SNMP entity potentially has access to many contexts.

Typically, there are many instances of each managed object type within a management domain. For simplicity, the method for identifying instances specified by the MIB module does not allow each instance to be distinguished amongst the set of all instances within a management domain; rather, it allows each instance to be identified only within some scope or "context", where there are multiple such contexts within the management domain. Often, a context is a physical device, or perhaps, a logical device, although a context can also encompass multiple devices, or a subset of a single device, or even a subset of multiple devices, but a context is always defined as a subset of a single SNMP entity. Thus, in order to identify an individual item of management information within the management domain, its contextName and contextEngineID must be identified in addition to its object type and its instance.

For example, the managed object type ifDescr [RFC1573], is defined as the description of a network interface. To identify the description of device-X's first network interface, four pieces of information are needed: the snmpEngineID of the SNMP entity which provides access to the management information at device-X, the contextName (device-X), the managed object type (ifDescr), and the instance ("1").

Each context has (at least) one unique identification within the management domain. The same item of management information can exist in multiple contexts. An item of management information may have multiple unique identifications. This occurs when an item of management information exists in multiple contexts, and this also occurs when a context has multiple unique identifications.

The combination of a contextEngineID and a contextName unambiguously identifies a context within an administrative domain; note that there may be multiple unique combinations of contextEngineID and contextName that unambiguously identify the same context.

3.3.2. contextEngineID

Within an administrative domain, a contextEngineID uniquely identifies an SNMP entity that may realize an instance of a context with a particular contextName.

3.3.3. contextName

A contextName is used to name a context. Each contextName MUST be unique within an SNMP entity.

3.3.4. scopedPDU

A scopedPDU is a block of data containing a contextEngineID, a contextName, and a PDU.

The PDU is an SNMP Protocol Data Unit containing information named in the context which is unambiguously identified within an administrative domain by the combination of the contextEngineID and the contextName. See, for example, RFC1905 for more information about SNMP PDUs.

3.4. Other Constructs

3.4.1. maxSizeResponseScopedPDU

The maxSizeResponseScopedPDU is the maximum size of a scopedPDU to be included in a response message. Note that the size of a scopedPDU does not include the size of the SNMP message header.

3.4.2. Local Configuration Datastore

The subsystems, models, and applications within an SNMP entity may need to retain their own sets of configuration information.

Portions of the configuration information may be accessible as managed objects.

The collection of these sets of information is referred to as an entity's Local Configuration Datastore (LCD).

3.4.3. securityLevel

This architecture recognizes three levels of security:

- without authentication and without privacy (noAuthNoPriv)
- with authentication but without privacy (authNoPriv)
- with authentication and with privacy (authPriv)

These three values are ordered such that noAuthNoPriv is less than authNoPriv and authNoPriv is less than authPriv.

Every message has an associated securityLevel. All Subsystems (Message Processing, Security, Access Control) and applications are required to either supply a value of securityLevel or to abide by the supplied value of securityLevel while processing the message and its contents.

4. Abstract Service Interfaces

Abstract service interfaces have been defined to describe the conceptual interfaces between the various subsystems within an SNMP entity.

These abstract service interfaces are defined by a set of primitives that define the services provided and the abstract data elements that are to be passed when the services are invoked. This section lists the primitives that have been defined for the various subsystems.

4.1. Dispatcher Primitives

The Dispatcher typically provides services to the SNMP applications via its PDU Dispatcher. This section describes the primitives provided by the PDU Dispatcher.

4.1.1. Generate Outgoing Request or Notification

The PDU Dispatcher provides the following primitive for an application to send an SNMP Request or Notification to another SNMP entity:

```
statusInformation =             -- sendPduHandle if success
                                -- errorIndication if failure
  sendPdu(
  IN   transportDomain          -- transport domain to be used
  IN   transportAddress         -- transport address to be used
  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
       )
```

4.1.2. Process Incoming Request or Notification PDU

The PDU Dispatcher provides the following primitive to pass an incoming SNMP PDU to an application:

```
processPdu(                     -- process Request/Notification PDU
  IN   messageProcessingModel   -- typically, SNMP version
  IN   securityModel            -- Security Model in use
  IN   securityName             -- 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
          )                         -- needed when sending a response
```

4.1.3. Generate Outgoing Response

The PDU Dispatcher provides the following primitive for an application to return an SNMP Response PDU to the PDU Dispatcher:

```
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
```

4.1.4. Process Incoming Response PDU

The PDU Dispatcher provides the following primitive to pass an incoming SNMP Response PDU to an application:

```
processResponsePdu(              -- process Response PDU
  IN   messageProcessingModel    -- typically, SNMP version
  IN   securityModel             -- Security Model in use
  IN   securityName              -- 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   statusInformation         -- success or errorIndication
  IN   sendPduHandle             -- handle from sendPdu
       )
```

4.1.5. Registering Responsibility for Handling SNMP PDUs

Applications can register/unregister responsibility for a specific contextEngineID, for specific pduTypes, with the PDU Dispatcher according to the following primitives. The list of particular pduTypes that an application can register for is determined by the Message Processing Model(s) supported by the SNMP entity that contains the PDU Dispatcher.

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

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

Note that realizations of the registerContextEngineID and unregisterContextEngineID abstract service interfaces may provide implementation-specific ways for applications to register/deregister responsiblity for all possible values of the contextEngineID or pduType parameters.

4.2. Message Processing Subsystem Primitives

The Dispatcher interacts with a Message Processing Model to process a specific version of an SNMP Message. This section describes the primitives provided by the Message Processing Subsystem.

4.2.1. Prepare Outgoing SNMP Request or Notification Message

The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Request or Notification Message:

```
statusInformation =              -- success or errorIndication
  prepareOutgoingMessage(
  IN   transportDomain           -- transport domain to be used
  IN   transportAddress          -- transport address to be used
  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
     IN   sendPduHandle             -- the handle for matching
                                    -- incoming responses
     OUT  destTransportDomain       -- destination transport domain
     OUT  destTransportAddress      -- destination transport address
     OUT  outgoingMessage           -- the message to send
     OUT  outgoingMessageLength     -- its length
          )
```

4.2.2. Prepare an Outgoing SNMP Response Message

The Message Processing Subsystem provides this service primitive for preparing an outgoing SNMP Response Message:

```
   result =                         -- SUCCESS or FAILURE
     prepareResponseMessage(
     IN   messageProcessingModel    -- typically, SNMP version
     IN   securityModel             -- same as on incoming request
     IN   securityName              -- same as on incoming request
     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
     OUT  destTransportDomain       -- destination transport domain
     OUT  destTransportAddress      -- destination transport address
     OUT  outgoingMessage           -- the message to send
     OUT  outgoingMessageLength     -- its length
          )
```

4.2.3. Prepare Data Elements from an Incoming SNMP Message

The Message Processing Subsystem provides this service primitive for preparing the abstract data elements from an incoming SNMP message:

```
   result =                         -- SUCCESS or errorIndication
     prepareDataElements(
     IN   transportDomain           -- origin transport domain
     IN   transportAddress          -- origin transport address
     IN   wholeMsg                  -- as received from the network
     IN   wholeMsgLength            -- as received from the network
     OUT  messageProcessingModel    -- typically, SNMP version
```

```
     OUT  securityModel             -- Security Model to use
     OUT  securityName              -- on behalf of this principal
     OUT  securityLevel             -- Level of Security requested
     OUT  contextEngineID           -- data from/at this entity
     OUT  contextName               -- data from/in this context
     OUT  pduVersion                -- the version of the PDU
     OUT  PDU                       -- SNMP Protocol Data Unit
     OUT  pduType                   -- SNMP PDU type
     OUT  sendPduHandle             -- handle for matched request
     OUT  maxSizeResponseScopedPDU  -- maximum size of the Response PDU
     OUT  statusInformation         -- success or errorIndication
                                    -- error counter OID/value if error
     OUT  stateReference            -- reference to state information
                                    -- to be used for possible Response
          )
```

4.3. Access Control Subsystem Primitives

Applications are the typical clients of the service(s) of the Access Control Subsystem.

The following primitive is provided by the Access Control Subsystem to check if access is allowed:

```
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
       )
```

4.4. Security Subsystem Primitives

The Message Processing Subsystem is the typical client of the services of the Security Subsystem.

4.4.1. Generate a Request or Notification Message

The Security Subsystem provides the following primitive to generate a Request or Notification message:

```
statusInformation =
  generateRequestMsg(
  IN   messageProcessingModel    -- typically, SNMP version
  IN   globalData                -- message header, admin data
```

```
     IN   maxMessageSize            -- of the sending SNMP entity
     IN   securityModel             -- for the outgoing message
     IN   securityEngineID          -- authoritative SNMP entity
     IN   securityName              -- on behalf of this principal
     IN   securityLevel             -- Level of Security requested
     IN   scopedPDU                 -- message (plaintext) payload
     OUT  securityParameters        -- filled in by Security Module
     OUT  wholeMsg                  -- complete generated message
     OUT  wholeMsgLength            -- length of the generated message
          )
```

4.4.2. Process Incoming Message

The Security Subsystem provides the following primitive to process an incoming message:

```
statusInformation =              -- errorIndication or success
                                 -- error counter OID/value if error
  processIncomingMsg(
  IN   messageProcessingModel    -- typically, SNMP version
  IN   maxMessageSize            -- of the sending SNMP entity
  IN   securityParameters        -- for the received message
  IN   securityModel             -- for the received message
  IN   securityLevel             -- Level of Security
  IN   wholeMsg                  -- as received on the wire
  IN   wholeMsgLength            -- length as received on the wire
  OUT  securityEngineID          -- identification of the principal
  OUT  securityName              -- identification of the principal
  OUT  scopedPDU,                -- message (plaintext) payload
  OUT  maxSizeResponseScopedPDU  -- maximum size of the Response PDU
  OUT  securityStateReference    -- reference to security state
       )                         -- information, needed for response
```

4.4.3. Generate a Response Message

The Security Subsystem provides the following primitive to generate a Response message:

```
statusInformation =
  generateResponseMsg(
  IN   messageProcessingModel    -- typically, SNMP version
  IN   globalData                -- message header, admin data
  IN   maxMessageSize            -- of the sending SNMP entity
  IN   securityModel             -- for the outgoing message
  IN   securityEngineID          -- authoritative SNMP entity
  IN   securityName              -- on behalf of this principal
  IN   securityLevel             -- for the outgoing message
  IN   scopedPDU                 -- message (plaintext) payload
```

```
     IN   securityStateReference    -- reference to security state
                                    -- information from original request
     OUT  securityParameters        -- filled in by Security Module
     OUT  wholeMsg                  -- complete generated message
     OUT  wholeMsgLength            -- length of the generated message
          )
```

4.5. Common Primitives

These primitive(s) are provided by multiple Subsystems.

4.5.1. Release State Reference Information

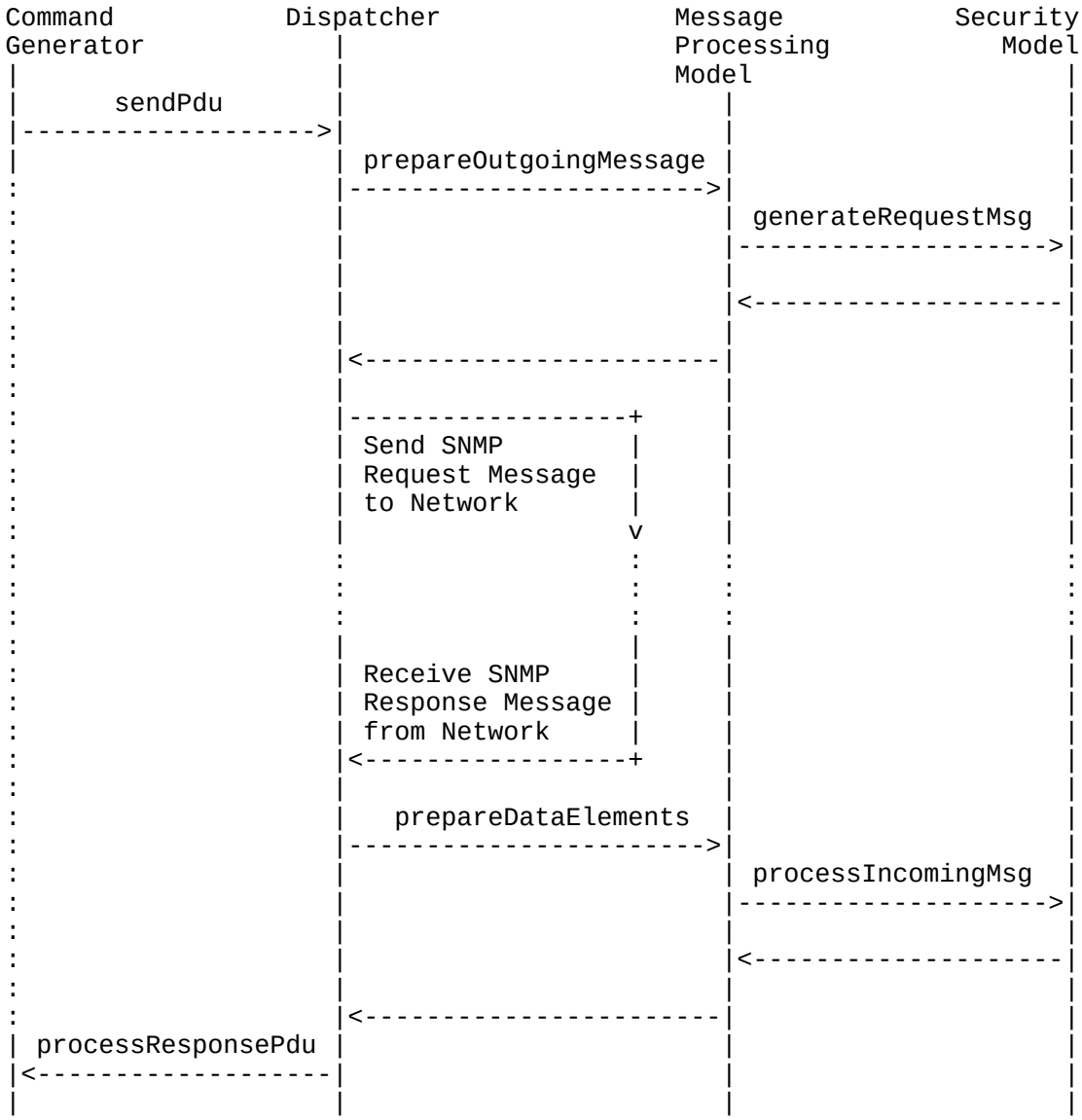
All Subsystems which pass stateReference information also provide a primitive to release the memory that holds the referenced state information:

```
stateRelease(
  IN   stateReference       -- handle of reference to be released
       )
```

4.6. Scenario Diagrams

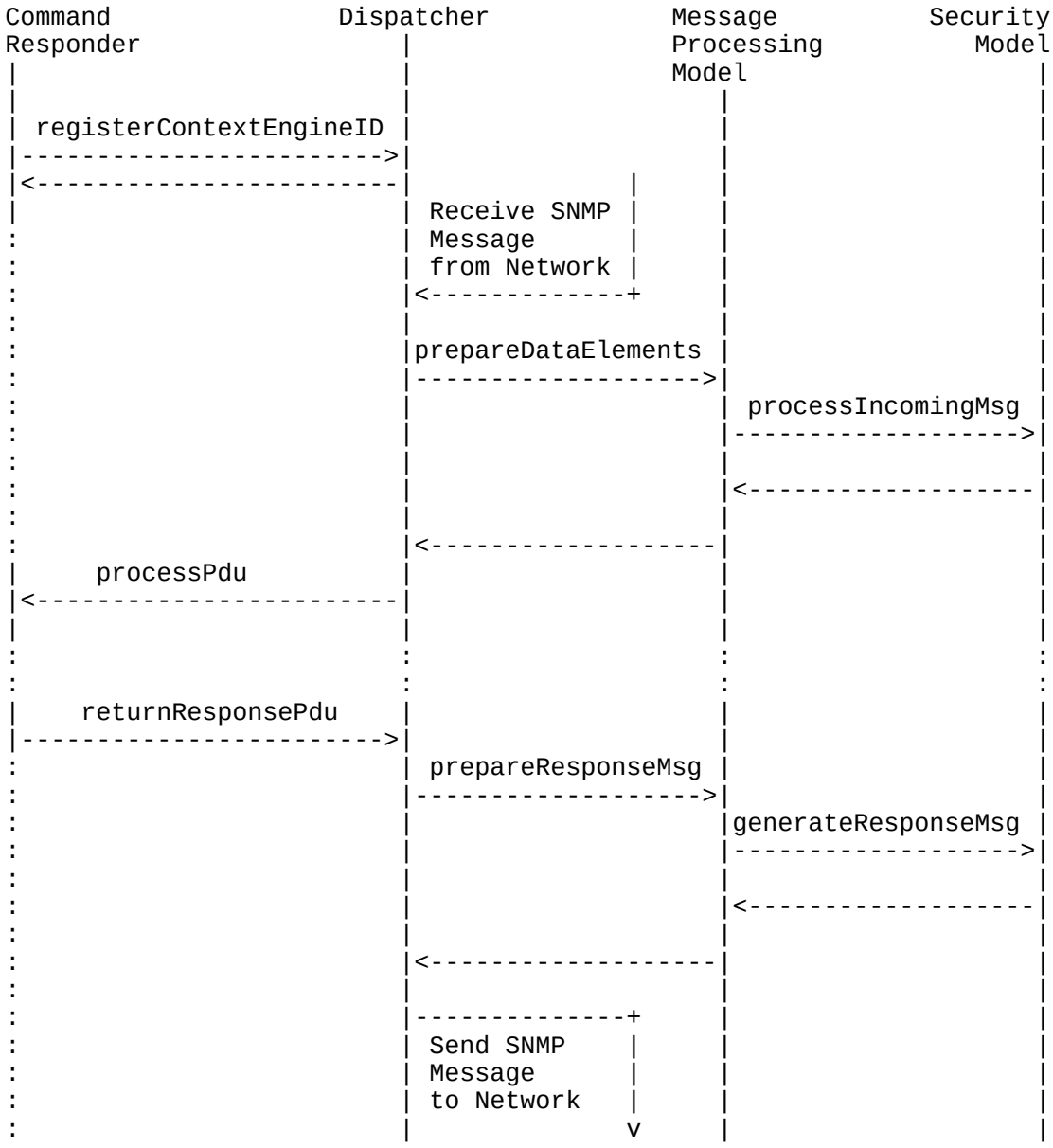
4.6.1. Command Generator or Notification Originator

This diagram shows how a Command Generator or Notification Originator application requests that a PDU be sent, and how the response is returned (asynchronously) to that application.



4.6.2. Scenario Diagram for a Command Responder Application

This diagram shows how a Command Responder or Notification Receiver application registers for handling a pduType, how a PDU is dispatched to the application after a SNMP message is received, and how the Response is (asynchronously) send back to the network.


Command Responder
Dispatcher
Message Processing Model
Security Model
registerContextEngineID
Receive SNMP Message from Network
prepareDataElements
processIncomingMsg
processPdu
returnResponsePdu
prepareResponseMsg
generateResponseMsg
Send SNMP Message to Network

5. Managed Object Definitions for SNMP Management Frameworks

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

   IMPORTS
       MODULE-IDENTITY, OBJECT-TYPE,
       OBJECT-IDENTITY,
       snmpModules                          FROM SNMPv2-SMI
       TEXTUAL-CONVENTION                   FROM SNMPv2-TC
       MODULE-COMPLIANCE, OBJECT-GROUP      FROM SNMPv2-CONF;

   snmpFrameworkMIB MODULE-IDENTITY
       LAST-UPDATED "9711200000Z"           -- 20 November 1997
       ORGANIZATION "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   Dave Harrington
                                 Cabletron Systems, Inc.
                     postal:     Post Office Box 5005
                                 Mail Stop: Durham
                                 35 Industrial Way
                                 Rochester, NH 03867-5005
                                 USA
                     email:      dbh@ctron.com
                     phone:      +1 603-337-7357

                     Co-editor   Randy Presuhn
                                 BMC Software, Inc.
                     postal:     1190 Saratoga Avenue
                                 Suite 130
                                 San Jose, CA 95129
                                 USA
                     email:      rpresuhn@bmc.com
                     phone:      +1 408-556-0720

                     Co-editor:  Bert Wijnen
                                 IBM T.J. Watson Research
                     postal:     Schagen 33
```

```
                                3461 GL Linschoten
                                Netherlands
                         email: wijnen@vnet.ibm.com
                         phone: +31 348-432-794
                        "
        DESCRIPTION  "The SNMP Management Architecture MIB"
        ::= { snmpModules 2 }

    -- Textual Conventions used in the SNMP Management Architecture ***

    SnmpEngineID ::= TEXTUAL-CONVENTION
        STATUS       current
        DESCRIPTION "An SNMP engine's administratively-unique identifier.

                     The value for this object may not be all zeros or
                     all 'ff'H or the empty (zero length) string.

                     The initial value for this object may be configured
                     via an operator console entry or via an algorithmic
                     function.  In the latter case, the following
                     example algorithm is recommended.

                     In cases where there are multiple engines on the
                     same system, the use of this algorithm is NOT
                     appropriate, as it would result in all of those
                     engines ending up with the same ID value.

                     1) The very first bit is used to indicate how the
                        rest of the data is composed.

                        0 - as defined by enterprise using former methods
                            that existed before SNMPv3. See item 2 below.

                        1 - as defined by this architecture, see item 3
                            below.

                        Note that this allows existing uses of the
                        engineID (also known as AgentID [RFC1910]) to
                        co-exist with any new uses.

                     2) The snmpEngineID has a length of 12 octets.

                        The first four octets are set to the binary
                        equivalent of the agent's SNMP management
                        private enterprise number as assigned by the
                        Internet Assigned Numbers Authority (IANA).
                        For example, if Acme Networks has been assigned
                        { enterprises 696 }, the first four octets would
```

be assigned '000002b8'H.

The remaining eight octets are determined via one or more enterprise-specific methods. Such methods must be designed so as to maximize the possibility that the value of this object will be unique in the agent's administrative domain. For example, it may be the IP address of the SNMP entity, or the MAC address of one of the interfaces, with each address suitably padded with random octets. If multiple methods are defined, then it is recommended that the first octet indicate the method being used and the remaining octets be a function of the method.

3) The length of the octet strings varies.

The first four octets are set to the binary equivalent of the agent's SNMP management private enterprise number as assigned by the Internet Assigned Numbers Authority (IANA). For example, if Acme Networks has been assigned { enterprises 696 }, the first four octets would be assigned '000002b8'H.

The very first bit is set to 1. For example, the above value for Acme Networks now changes to be '800002b8'H.

The fifth octet indicates how the rest (6th and following octets) are formatted. The values for the fifth octet are:

0 - reserved, unused.

1 - IPv4 address (4 octets)
lowest non-special IP address

2 - IPv6 address (16 octets)
lowest non-special IP address

3 - MAC address (6 octets)
lowest IEEE MAC address, canonical order

4 - Text, administratively assigned
Maximum remaining length 27

```
                        5     - Octets, administratively assigned
                                Maximum remaining length 27

                        6-127 - reserved, unused

                      127-255 - as defined by the enterprise
                                Maximum remaining length 27
                    "
        SYNTAX       OCTET STRING (SIZE(1..32))

    SnmpSecurityModel ::= TEXTUAL-CONVENTION
        STATUS       current
        DESCRIPTION "An identifier that uniquely identifies a
                     securityModel of the Security Subsystem within the
                     SNMP Management Architecture.

                     The values for securityModel are allocated as
                     follows:

                     - The zero value is reserved.
                     - Values between 1 and 255, inclusive, are reserved
                       for standards-track Security Models and are
                       managed by the Internet Assigned Numbers Authority
                       (IANA).
                     - Values greater than 255 are allocated to
                       enterprise-specific Security Models.  An
                       enterprise-specific securityModel value is defined
                       to be:

                       enterpriseID * 256 + security model within
                       enterprise

                       For example, the fourth Security Model defined by
                       the enterprise whose enterpriseID is 1 would be
                       260.

                     This scheme for allocation of securityModel
                     values allows for a maximum of 255 standards-
                     based Security Models, and for a maximum of
                     255 Security Models per enterprise.

                     It is believed that the assignment of new
                     securityModel values will be rare in practice
                     because the larger the number of simultaneously
                     utilized Security Models, the larger the
                     chance that interoperability will suffer.
                     Consequently, it is believed that such a range
                     will be sufficient.  In the unlikely event that
```

```
                    the standards committee finds this number to be
                    insufficient over time, an enterprise number
                    can be allocated to obtain an additional 255
                    possible values.

                    Note that the most significant bit must be zero;
                    hence, there are 23 bits allocated for various
                    organizations to design and define non-standard
                    securityModels.  This limits the ability to
                    define new proprietary implementations of Security
                    Models to the first 8,388,608 enterprises.

                    It is worthwhile to note that, in its encoded
                    form, the securityModel value will normally
                    require only a single byte since, in practice,
                    the leftmost bits will be zero for most messages
                    and sign extension is suppressed by the encoding
                    rules.

                    As of this writing, there are several values
                    of securityModel defined for use with SNMP or
                    reserved for use with supporting MIB objects.
                    They are as follows:

                        0  reserved for 'any'
                        1  reserved for SNMPv1
                        2  reserved for SNMPv2c
                        3  User-Based Security Model (USM)
                   "
        SYNTAX      INTEGER(0..2147483647)

    SnmpMessageProcessingModel ::= TEXTUAL-CONVENTION
        STATUS      current
        DESCRIPTION "An identifier that uniquely identifies a Message
                    Processing Model of the Message Processing
                    Subsystem within a SNMP Management Architecture.

                    The values for messageProcessingModel are
                    allocated as follows:

                    - Values between 0 and 255, inclusive, are
                      reserved for standards-track Message Processing
                      Models and are managed by the Internet Assigned
                      Numbers Authority (IANA).
                    - Values greater than 255 are allocated to
                      enterprise-specific Message Processing Models.
                      An enterprise messageProcessingModel value is
                      defined to be:
```

enterpriseID * 256 +
messageProcessingModel within enterprise

For example, the fourth Message Processing Model defined by the enterprise whose enterpriseID is 1 would be 260.

This scheme for allocation of securityModel values allows for a maximum of 255 standards-based Message Processing Models, and for a maximum of 255 Message Processing Models per enterprise.

It is believed that the assignment of new messageProcessingModel values will be rare in practice because the larger the number of simultaneously utilized Message Processing Models, the larger the chance that interoperability will suffer. It is believed that such a range will be sufficient. In the unlikely event that the standards committee finds this number to be insufficient over time, an enterprise number can be allocated to obtain an additional 256 possible values.

Note that the most significant bit must be zero; hence, there are 23 bits allocated for various organizations to design and define non-standard messageProcessingModels. This limits the ability to define new proprietary implementations of Message Processing Models to the first 8,388,608 enterprises.

It is worthwhile to note that, in its encoded form, the securityModel value will normally require only a single byte since, in practice, the leftmost bits will be zero for most messages and sign extension is suppressed by the encoding rules.

As of this writing, there are several values of messageProcessingModel defined for use with SNMP. They are as follows:

0 reserved for SNMPv1
1 reserved for SNMPv2c
2 reserved for SNMPv2u and SNMPv2*
3 reserved for SNMPv3

```
                    "
        SYNTAX       INTEGER(0..2147483647)

    SnmpSecurityLevel ::= TEXTUAL-CONVENTION
        STATUS       current
        DESCRIPTION "A Level of Security at which SNMP messages can be
                     sent or with which operations are being processed;
                     in particular, one of:

                       noAuthNoPriv - without authentication and
                                      without privacy,
                       authNoPriv   - with authentication but
                                      without privacy,
                       authPriv     - with authentication and
                                      with privacy.

                     These three values are ordered such that
                     noAuthNoPriv is less than authNoPriv and
                     authNoPriv is less than authPriv.
                    "
        SYNTAX       INTEGER { noAuthNoPriv(1),
                               authNoPriv(2),
                               authPriv(3)
                             }

    SnmpAdminString ::= TEXTUAL-CONVENTION
        DISPLAY-HINT "255a"
        STATUS       current
        DESCRIPTION "An octet string containing administrative
                     information, 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 transformation format
                     described in [RFC2044].

                     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.

                     When it is necessary to represent a newline,
                     the control code sequence CR LF should be used.
```

```
                    The use of leading or trailing white space should
                    be avoided.

                    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 encoding 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 of sub-identifiers for any object instance
                    does not exceed the limit of 128, as defined by
                    [RFC1905].
                   "
        SYNTAX      OCTET STRING (SIZE (0..255))


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

    snmpFrameworkAdmin
        OBJECT IDENTIFIER ::= { snmpFrameworkMIB 1 }
    snmpFrameworkMIBObjects
        OBJECT IDENTIFIER ::= { snmpFrameworkMIB 2 }
    snmpFrameworkMIBConformance
        OBJECT IDENTIFIER ::= { snmpFrameworkMIB 3 }

    -- the snmpEngine Group ******************************************

    snmpEngine OBJECT IDENTIFIER ::= { snmpFrameworkMIBObjects 1 }

    snmpEngineID     OBJECT-TYPE
        SYNTAX       SnmpEngineID
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "An SNMP engine's administratively-unique identifier.
                    "
        ::= { snmpEngine 1 }

    snmpEngineBoots  OBJECT-TYPE
        SYNTAX       INTEGER (1..2147483647)
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The number of times that the SNMP engine has
```

```
                    (re-)initialized itself since its initial
                    configuration.
                   "
        ::= { snmpEngine 2 }

    snmpEngineTime   OBJECT-TYPE
        SYNTAX       INTEGER (0..2147483647)
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The number of seconds since the SNMP engine last
                     incremented the snmpEngineBoots object.
                    "
        ::= { snmpEngine 3 }

    snmpEngineMaxMessageSize OBJECT-TYPE
        SYNTAX       INTEGER (484..2147483647)
        MAX-ACCESS   read-only
        STATUS       current
        DESCRIPTION "The maximum length in octets of an SNMP message
                     which this SNMP engine can send or receive and
                     process, determined as the minimum of the maximum
                     message size values supported among all of the
                     transports available to and supported by the engine.
                    "
        ::= { snmpEngine 4 }


    -- Registration Points for Authentication and Privacy Protocols **

    snmpAuthProtocols OBJECT-IDENTITY
        STATUS        current
        DESCRIPTION  "Registration point for standards-track
                      authentication protocols used in SNMP Management
                      Frameworks.
                     "
        ::= { snmpFrameworkAdmin 1 }

    snmpPrivProtocols OBJECT-IDENTITY
        STATUS        current
        DESCRIPTION  "Registration point for standards-track privacy
                      protocols used in SNMP Management Frameworks.
                     "
        ::= { snmpFrameworkAdmin 2 }

    -- Conformance information ****************************************

    snmpFrameworkMIBCompliances
                   OBJECT IDENTIFIER ::= {snmpFrameworkMIBConformance 1}
```

```
   snmpFrameworkMIBGroups
                  OBJECT IDENTIFIER ::= {snmpFrameworkMIBConformance 2}

   -- compliance statements

   snmpFrameworkMIBCompliance MODULE-COMPLIANCE
       STATUS       current
       DESCRIPTION "The compliance statement for SNMP engines which
                    implement the SNMP Management Framework MIB.
                   "
       MODULE    -- this module
           MANDATORY-GROUPS { snmpEngineGroup }

       ::= { snmpFrameworkMIBCompliances 1 }

   -- units of conformance

   snmpEngineGroup OBJECT-GROUP
       OBJECTS {
                 snmpEngineID,
                 snmpEngineBoots,
                 snmpEngineTime,
                 snmpEngineMaxMessageSize
               }
       STATUS       current
       DESCRIPTION "A collection of objects for identifying and
                    determining the configuration and current timeliness
                    values of an SNMP engine.
                   "
       ::= { snmpFrameworkMIBGroups 1 }

   END
```

6. Intellectual Property

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

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

7. Acknowledgements

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.)

8. Security Considerations

This document describes how an implementation can include a Security Model to protect management messages and an Access Control Model to control access to management information.

The level of security provided is determined by the specific Security Model implementation(s) and the specific Access Control Model implementation(s) used.

Applications have access to data which is not secured. Applications should take reasonable steps to protect the data from disclosure.

It is the responsibility of the purchaser of an implementation to ensure that:

1) an implementation complies with the rules defined by this architecture,

2) the Security and Access Control Models utilized satisfy the security and access control needs of the organization,

3) the implementations of the Models and Applications comply with the model and application specifications,

4) and the implementation protects configuration secrets from inadvertent disclosure.

9. References

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

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

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

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

[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.

[RFC1905] 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.

[RFC1906] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[RFC1907] 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] 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.

[RFC1909] McCloghrie, K., Editor, "An Administrative Infrastructure for SNMPv2", RFC 1909, February 1996.

[RFC1910] Waters, G., Editor, "User-based Security Model for SNMPv2", RFC 1910, February 1996.

[RFC2028] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[RFC2044] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2262] Case, J., Harrington, D., Presuhn, R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2262, January 1998.

[RFC2264] Blumenthal, U., and B. Wijnen, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2264, January 1998.

[RFC2265] Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2265, January 1998.

[RFC2263] Levi, D., Meyer, P., and B. Stewart, "SNMPv3 Applications", RFC 2263, January 1998.

10. Editors' Addresses

Bert Wijnen
IBM T.J. Watson Research
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31 348-432-794
EMail: wijnen@vnet.ibm.com

Dave Harrington
Cabletron Systems, Inc
Post Office Box 5005
Mail Stop: Durham
35 Industrial Way
Rochester, NH 03867-5005
USA

Phone: +1 603-337-7357
EMail: dbh@ctron.com

Randy Presuhn
BMC Software, Inc.
1190 Saratoga Avenue
Suite 130
San Jose, CA 95129
USA

Phone: +1 408-556-0720
EMail: rpresuhn@bmc.com

APPENDIX A

A. Guidelines for Model Designers

This appendix describes guidelines for designers of models which are expected to fit into the architecture defined in this document.

SNMPv1 and SNMPv2c are two SNMP frameworks which use communities to provide trivial authentication and access control. SNMPv1 and SNMPv2c Frameworks can coexist with Frameworks designed according to this architecture, and modified versions of SNMPv1 and SNMPv2c Frameworks could be designed to meet the requirements of this architecture, but this document does not provide guidelines for that coexistence.

Within any subsystem model, there should be no reference to any specific model of another subsystem, or to data defined by a specific model of another subsystem.

Transfer of data between the subsystems is deliberately described as a fixed set of abstract data elements and primitive functions which can be overloaded to satisfy the needs of multiple model definitions.

Documents which define models to be used within this architecture SHOULD use the standard primitives between subsystems, possibly defining specific mechanisms for converting the abstract data elements into model-usable formats. This constraint exists to allow subsystem and model documents to be written recognizing common borders of the subsystem and model. Vendors are not constrained to recognize these borders in their implementations.

The architecture defines certain standard services to be provided between subsystems, and the architecture defines abstract service interfaces to request these services.

Each model definition for a subsystem SHOULD support the standard service interfaces, but whether, or how, or how well, it performs the service is dependent on the model definition.

A.1. Security Model Design Requirements

A.1.1. Threats

A document describing a Security Model MUST describe how the model protects against the threats described under "Security Requirements of this Architecture", section 1.4.

A.1.2. Security Processing

Received messages MUST be validated by a Model of the Security Subsystem. Validation includes authentication and privacy processing if needed, but it is explicitly allowed to send messages which do not require authentication or privacy.

A received message contains a specified securityLevel to be used during processing. All messages requiring privacy MUST also require authentication.

A Security Model specifies rules by which authentication and privacy are to be done. A model may define mechanisms to provide additional security features, but the model definition is constrained to using (possibly a subset of) the abstract data elements defined in this document for transferring data between subsystems.

Each Security Model may allow multiple security protocols to be used concurrently within an implementation of the model. Each Security Model defines how to determine which protocol to use, given the securityLevel and the security parameters relevant to the message. Each Security Model, with its associated protocol(s) defines how the sending/receiving entities are identified, and how secrets are configured.

Authentication and Privacy protocols supported by Security Models are uniquely identified using Object Identifiers. IETF standard protocols for authentication or privacy should have an identifier defined within the snmpAuthProtocols or the snmpPrivProtocols subtrees. Enterprise specific protocol identifiers should be defined within the enterprise subtree.

For privacy, the Security Model defines what portion of the message is encrypted.

The persistent data used for security should be SNMP-manageable, but the Security Model defines whether an instantiation of the MIB is a conformance requirement.

Security Models are replaceable within the Security Subsystem. Multiple Security Model implementations may exist concurrently within an SNMP engine. The number of Security Models defined by the SNMP community should remain small to promote interoperability.

A.1.3. Validate the security-stamp in a received message

A Message Processing Model requests that a Security Model:
- verifies that the message has not been altered,
- authenticates the identification of the principal for whom the message was generated.
- decrypts the message if it was encrypted.

Additional requirements may be defined by the model, and additional services may be provided by the model, but the model is constrained to use the following primitives for transferring data between subsystems. Implementations are not so constrained.

A Message Processing Model uses the processMsg primitive as described in section 4.5.

A.1.4. Security MIBs

Each Security Model defines the MIB module(s) required for security processing, including any MIB module(s) required for the security protocol(s) supported. The MIB module(s) SHOULD be defined concurrently with the procedures which use the MIB module(s). The MIB module(s) are subject to normal access control rules.

The mapping between the model-dependent security ID and the securityName MUST be able to be determined using SNMP, if the model-dependent MIB is instantiated and if access control policy allows access.

A.1.5. Cached Security Data

For each message received, the Security Model caches the state information such that a Response message can be generated using the same security information, even if the Local Configuration Datastore is altered between the time of the incoming request and the outgoing response.

A Message Processing Model has the responsibility for explicitly releasing the cached data if such data is no longer needed. To enable this, an abstract securityStateReference data element is passed from the Security Model to the Message Processing Model.

The cached security data may be implicitly released via the generation of a response, or explicitly released by using the stateRelease primitive, as described in section 4.1.

A.2. Message Processing Model Design Requirements

An SNMP engine contains a Message Processing Subsystem which may contain multiple Message Processing Models.

The Message Processing Model MUST always (conceptually) pass the complete PDU, i.e., it never forwards less than the complete list of varBinds.

A.2.1. Receiving an SNMP Message from the Network

Upon receipt of a message from the network, the Dispatcher in the SNMP engine determines the version of the SNMP message and interacts with the corresponding Message Processing Model to determine the abstract data elements.

A Message Processing Model specifies the SNMP Message format it supports and describes how to determine the values of the abstract data elements (like msgID, msgMaxSize, msgFlags, msgSecurityParameters, securityModel, securityLevel etc). A Message Processing Model interacts with a Security Model to provide security processing for the message using the processMsg primitive, as described in section 4.5.

A.2.2. Sending an SNMP Message to the Network

The Dispatcher in the SNMP engine interacts with a Message Processing Model to prepare an outgoing message. For that it uses the following primitives:

- for requests and notifications: prepareOutgoingMessage, as described in section 4.4
- for response messages: prepareResponseMessage, as described in section 4.4

A Message Processing Model, when preparing an Outgoing SNMP Message, interacts with a Security Model to secure the message. For that it uses the following primitives:

- for requests and notifications: generateRequestMsg, as described in section 4.5.
- for response messages: generateResponseMsg as described in section 4.5.

Once the SNMP message is prepared by a Message Processing Model, the Dispatcher sends the message to the desired address using the appropriate transport.

A.3. Application Design Requirements

Within an application, there may be an explicit binding to a specific SNMP message version, i.e., a specific Message Processing Model, and to a specific Access Control Model, but there should be no reference to any data defined by a specific Message Processing Model or Access Control Model.

Within an application, there should be no reference to any specific Security Model, or any data defined by a specific Security Model.

An application determines whether explicit or implicit access control should be applied to the operation, and, if access control is needed, which Access Control Model should be used.

An application has the responsibility to define any MIB module(s) used to provide application-specific services.

Applications interact with the SNMP engine to initiate messages, receive responses, receive asynchronous messages, and send responses.

A.3.1. Applications that Initiate Messages

Applications may request that the SNMP engine send messages containing SNMP commands or notifications using the sendPdu primitive as described in section 4.2.

If it is desired that a message be sent to multiple targets, it is the responsibility of the application to provide the iteration.

The SNMP engine assumes necessary access control has been applied to the PDU, and provides no access control services.

The SNMP engine looks at the "expectResponse" parameter, and if a response is expected, then the appropriate information is cached such that a later response can be associated to this message, and can then be returned to the application. A sendPduHandle is returned to the application so it can later correspond the response with this message as well.

A.3.2. Applications that Receive Responses

The SNMP engine matches the incoming response messages to outstanding messages sent by this SNMP engine, and forwards the response to the associated application using the processResponsePdu primitive, as described in section 4.2.

A.3.3. Applications that Receive Asynchronous Messages

When an SNMP engine receives a message that is not the response to a request from this SNMP engine, it must determine to which application the message should be given.

An Application that wishes to receive asynchronous messages registers itself with the engine using the primitive registerContextEngineID as described in section 4.2.

An Application that wishes to stop receiving asynchronous messages should unregister itself with the SNMP engine using the primitive unregisterContextEngineID as described in section 4.2.

Only one registration per combination of PDU type and contextEngineID is permitted at the same time. Duplicate registrations are ignored. An errorIndication will be returned to the application that attempts to duplicate a registration.

All asynchronously received messages containing a registered combination of PDU type and contextEngineID are sent to the application which registered to support that combination.

The engine forwards the PDU to the registered application, using the processPdu primitive, as described in section 4.2.

A.3.4. Applications that Send Responses

Request operations require responses. An application sends a response via the returnResponsePdu primitive, as described in section 4.2.

The contextEngineID, contextName, securityModel, securityName, securityLevel, and stateReference parameters are from the initial processPdu primitive. The PDU and statusInformation are the results of processing.

A.4. Access Control Model Design Requirements

An Access Control Model determines whether the specified securityName is allowed to perform the requested operation on a specified managed object. The Access Control Model specifies the rules by which access control is determined.

The persistent data used for access control should be manageable using SNMP, but the Access Control Model defines whether an instantiation of the MIB is a conformance requirement.

The Access Control Model must provide the primitive isAccessAllowed.

B. Full Copyright Statement