Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

1. Abstract

This memo defines a portion of the Management Information Base (MIB) for use with network management protocols in TCP/IP-based internets. In particular, this MIB defines objects that enable policy-based configuration management of SNMP.
2. The SNMP Management Framework

The SNMP Management Framework presently consists of five major components:

- An overall architecture, described in RFC 2571 [1].
- Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in STD 16, RFC 1155 [2], STD 16, RFC 1212 [3] and RFC 1215 [4]. The second version, called SMIv2, is described in STD 58, RFC 2578 [5], RFC 2579 [6] and RFC 2580 [7].
- Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157 [8]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [9] and RFC 1906 [10]. The third version of the message protocol is called SNMPv3 and described in RFC 1906 [10], RFC 2572 [11] and RFC 2574 [12].
- Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in STD 15, RFC 1157 [8]. A second set of protocol operations and associated PDU formats is described in RFC 1905 [13].
- A set of fundamental applications described in RFC 2573 [14] and the view-based access control mechanism described in RFC 2575 [15].

A more detailed introduction to the current SNMP Management Framework can be found in RFC 2570 [18].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.
This memo specifies a MIB module that is compliant to the SMIv2. A MIB conforming to the SMIv1 can be produced through the appropriate translations. The resulting translated MIB must be semantically equivalent, except where objects or events are omitted because no translation is possible (use of Counter64). Some machine readable information in SMIv2 will be converted into textual descriptions in SMIv1 during the translation process. However, this loss of machine readable information is not considered to change the semantics of the MIB.
3. Overview

Large IT organizations have developed management strategies to cope with the extraordinarily large scale inherent in large networks. In particular, they try to manage the network as a whole by describing and implementing high-level business policies, rather than managing device by device, where orders of magnitude more decisions (and mistakes) may be made.

Following this management practice results in the following benefits:
- Reduced training needs (fewer details to learn)
- Reduced documentation costs (fewer details to document)
- Reduced impact of turnover (less ad-hoc knowledge goes out the door)
- Greater testability (a greater percentage of fielded configurations may be tested in the lab)
- Higher reliability (combination of factors above)
- Lower cost of changes (changes can be simpler and operate over a wider extent)
- Lower cost of corporate mergers (less knowledge to transfer; fewer policies to integrate)
- Lower cost of ownership (combination of factors above)

To illustrate the concept of "business policies", some examples are:
- All routers will run code version 6.2
- On-site contractors will all have special security restrictions on their ports
- All voice over cable ports in California must provide free local calling
- Apply special forwarding to all ports whose customers have paid for premium service.

Each of these policies could represent an action applied to hundreds of thousands of configuration variables.

In order to automate this practice, customers need software tools that will implement business policies across their network, as well as a standard protocol that will ensure that it can be applied to all of their devices, regardless of the vendor.

This practice is called Policy-Based Network Management. This document defines standard managed objects for the Simple Network Management Protocol that are used to distribute policies in a standard form throughout the network.
4. Policy-Based Management Architecture

Policy-based network management is the practice of applying management operations globally on all managed objects that share certain attributes.

Policies always express a notion of:
   if (an object has certain characteristics) then (apply operation to that object)

Policies take the following normal form:

   if (policyFilter) then (policyAction)

A policyFilter is an expression which results in a boolean to determine whether or not an object is a member of a set of objects upon which an action is to be performed.

A policyAction is an operation performed on a set of objects.

These policies are executed on managed devices, where the objects live (and thus their characteristics may be easily inspected), and where operations on those objects will be performed.

A management station is responsible for distributing an organization’s policies to all of the managed devices in the infrastructure. The pmPolicyTable provides managed objects for sending a policy to a managed device.

In this architecture, the objects that policies act on are called elements. An element is a group of related MIB variables such as all the variables for interface #7. This enables policies to be expressed more efficiently and concisely. Elements can also model circuits, CPUs, queues, processes, systems, etc.

The execution model for policies on a managed device is:

   foreach element for which policyFilter returns true
   execute policyAction on that element

For example:

   If (interface is fast ethernet)    then (apply full-duplex mode)
   If (interface is access)          then (apply security filters)
   If (gold service paid for on circuit) then (apply special queueing)
PolicyFilters have the capability of performing comparison operations on SNMP variables, logical expressions, and other functions. Many device characteristics are already defined in MIBs and are easy to include in policyFilter expressions (ifType == ethernet, frCircuitCommittedBurst < 128K, etc). However, there are important characteristics that aren’t currently in MIB objects, and worse, it is not current practice to store this information on managed devices. Therefore, this document defines MIB objects for this information. To meet today’s needs there are three missing areas: roles, capabilities and time.

Roles

A role is an abstract characteristic assigned to an element that expresses a notion, such as a political, financial, legal, geographical, or architectural attribute, typically not directly derivable from information stored on the managed system. For example, "paid for premium service" or "is plugged into a UPS" are examples of roles, whereas the percent utilization of a link would not be.

The types of information one would put into a role are:

- **political** - describes the role of a person or group of people, or of a service that a group of people use. Examples: executive, sales, outside-contracter, customer.
  If (attached user is executive) then (apply higher bandwidth)
  If (attached user is outside-contracter) then (restrict access)

- **financial/legal** - describes what financial consideration was received. Could also include contractual or legal considerations. Examples: paid, gold, free, trial, demo, lifeline
  (The lifeline example is supposed to model the RBOC’s legal obligation to provide dial tone to elderly/poor).
  If (gold service paid for) then (apply special queueing)

- **geographical** - describes the location of an element. Examples: California, Headquarters, insecure conduit.
  If (interface leaves the building) then (apply special security)

- **architectural** - describes the network architects "intent" for an element. For example: backup, trunk.
  If (interface is backup) then (set ifAdminStatus = down)
Collectively, these 4 classes of characteristics are called roles. Roles are human defined strings that can be referenced by a policyFilter. Multiple roles may be assigned to each element.

Capabilities

Some actions are inappropriate for certain elements or are simply unsupported. PolicyFilter’s must be able to be defined so that a policy can be applied only to elements that have the proper capability. The capabilities table provides MIB objects that describe the capabilities of the system.

Time

Managers may wish to define policies that are true for certain periods of time. This might mean that a policy is downloaded and is dormant for a period of time, becomes active, and then later becomes inactive. Sometimes these time periods will be regular (M-F 9-5) and sometimes ad-hoc. This MIB provides MIB objects that allow policies to be dependent on time.

5. Policy Based Management Execution Environment

There are several steps performed in order to execute policies in this environment:

- Element Discovery
- Element Filtering
- Policy Enforcement

5.1. Element Discovery

An element is a uniquely addressable entity on a managed device. Examples of elements include interfaces, circuits, queues, CPUs, and processes. Sometimes various attributes of an entity will be described through tables in several standard and proprietary MIBs - as long as the indexing is consistent between these tables, the entity can be modeled as 1 element. For example, the ifTable and the dot3Stats table both contain attributes of interfaces and share the same index (ifIndex), therefore they can be modeled as one element type.

The Element Type Registration table is used for the manager to
learn what element types are being managed by the system and to register new types if necessary. An element type is registered by providing the OID of an SNMP object (i.e., without the instance). Each SNMP instance that exists under that object is a distinct element. The address of the element is the index part of the discovered OID. This address will be supplied to policy filters and actions so that these scripts can inspect and configure the element.

For each element that is discovered, the policy filter is called with the element address as an argument to see if the element is a member of the set that the policy acts upon.

5.1.1. Implementation Notes

Note that while the external behavior of this registration process is defined in terms of the walking of MIB tables, implementation strategies may differ. For example, commonly-used element types (like interface) may have purpose-built element discovery capability built-in and advertised to managers through an entry in the pmElementTypeRegTable.

Before registering an element type, it is the responsibility of a manager to inspect the table and see if it is already registered (by the agent or another manager). Note that entries that differ only in the last OID (which specifies which object in an entry) are effectively duplicates and should be treated as such by the manager.

The system which implements the Policy-Based Management MIB may not have knowledge of the format of object identifiers in other MIBs. Therefore it is inappropriate for it to check these OIDs for errors. It is the responsibility of the management station to register well-formed object-identifiers. For example, if an extra sub-identifier is supplied when registering the ifTable, no elements will be discovered. Similarly, if a sub-identifier is missing, every element will be discovered numerous times (once per column) and none of the element addresses will be well-formed.
5.2. Element Filtering

The first step in executing a policy is to see which elements match the policy filter. To evaluate a policy, the policy filter is called once for each element and runs to completion. The element address is the only state that is passed to the script for each invocation (in particular, no state is remembered from the previous invocation of this element nor from the previous invocation of the policy filter). If any syntax or processing error occurs, the script will terminate immediately for this element. If the script returns non-zero, the corresponding policy action will be executed for this element.

5.2.1. Implementation Notes

It is an implementation-dependent matter as to how policy filters are scheduled. Each filter/element combination is conceptually its own process and can be scheduled sequentially or one or more could be run simultaneously.

Policy filters have no side-effects. Policy filter interpreters are encouraged to stop processing a filter as soon as it’s return value is known.

5.3. Policy Enforcement

For each element that has returned non-zero from the policy filter, the corresponding policy action is called. The element address is the only state that is passed to the script for each invocation (in particular, no state is remembered from the policy filter evaluation, nor from the previous filter/action invocation of this element nor from the previous invocation of the policy filter or action). If any syntax or processing error occurs, the script will terminate immediately for this element.

5.3.1. Implementation Notes

It is an implementation-dependent matter as to how policy actions are scheduled. Each filter/element combination is
conceptually its own process and can be scheduled sequentially or one or more could be run simultaneously.
6. Policy Based Management Expression Language

Policy filters and policy actions are expressed with the policy expression language. This expression language provides the power to make parenthesized logical and arithmetic comparisons and to call a number of pre-defined functions.

The policy expression language is a subset of the Perl and C languages that is limited to expressions. Some examples of the features that have been removed are: function definitions, local variables, assignments, pointers (except for constant pointers used for strings), arrays, structures, floating point and pre-processor functions.

Because there is no referenceable standard specification of the Perl language, this language is formally defined as a subset of ANSI C [19]. The policy expression language is defined in this standard by reference to ANSI C, but only allows those constructs that may be expressed in the BNF documented here. This is done because while BNF doesn’t fully specify syntactical rules (it allows constructs that are invalid) and doesn’t specify semantic rules, it can successfully be used to define the subset of ANSI C that is required for conformance to this standard.

The use of comments and newlines are allowed and encouraged where they will promote readability of expressions.

6.1. Formal Definition

The policy expression language follows the syntax and semantics of ANSI C [19], but expressions are limited to those that can be expressed in the following EBNF form:

\[
\begin{align*}
\text{identifier} & : \text{letter ( letter | digit )*} \\
\text{string} & : "" \text{ascii_char*}"" \\
\text{expression} & : \text{const_exp ( ';' const_exp )*} \\
\text{const_exp} & : \text{compound_exp | conditional_exp} \\
\text{conditional_exp} & : \text{const_exp }'?\text{ const_exp }':\text{ const_exp}
\end{align*}
\]
6.2. Accessor Functions

Several functions are available to provide access to information on the local system. No functions are available except for those described here.
6.2.1. getint()

The getint() function is used to retrieve the value of an SNMP MIB instance.

```c
int getint(char *objectIdentifier)
```

objectIdentifier is a NULL terminated string containing an ASCII dotted-decimal representation of an object identifier (e.g. 1.3.6.1.2.1.1.1.0).

The 2-character token "$1" can be used in place of any decimal sub-identifier. This token is expanded by the agent at execution time to contain the index value of each element the expression acts upon. For example, if the element is interface #7, and the objectIdentifier is "1.3.6.1.2.1.2.2.1.3.$1", it will be expanded to "1.3.6.1.2.1.2.2.1.3.7".

The agent will retrieve the instance in the same SNMP context in which the element resides. Note that no actual SNMP PDU needs to be generated and parsed when the policy MIB module resides on the same system as the managed elements.

If the queried object identifier value does not exist or is not an integer-valued object, execution of the containing expression on the current element will immediately terminate and the associated policyAction will not be executed on the current element.

This function returns the value of the integer-valued MIB instance.

It is recommended that NMS user interfaces display and allow input of MIB object names by their descriptor values followed by the index in dotted-decimal form (e.g., "ifType.7").

6.2.2. getstring()

The getstring() function is used to retrieve the value of an SNMP MIB instance.

```c
char * getstring(char *objectIdentifier)
```
objectIdentifier is a NULL terminated string containing an ASCII dotted-decimal representation of an object identifier (e.g. 1.3.6.1.2.1.1.1.0).

The 2-character token "$1" can be used in place of any decimal sub-identifier. This token is expanded by the agent at execution time to contain the index value of each element the expression acts upon. For example, if the element is interface #7, and the objectIdentifier is "1.3.6.1.2.1.2.2.1.3.$1", it will be expanded to "1.3.6.1.2.1.2.2.1.3.7".

The agent will retrieve the instance in the same SNMP context in which the element resides. Note that no actual SNMP PDU needs to be generated and parsed when the policy MIB module resides on the same system as the managed elements.

If the queried object identifier value does not exist or is not a string-valued or object-identifier-valued object, execution of the containing expression on the current element will immediately terminate and the associated policyAction will not be executed on the current element.

This function returns a string containing the value of string-valued MIB instances or the ASCII dotted-decimal representation of object-identifier-valued instances.

It is recommended that NMS user interfaces display and allow input of MIB object names by their descriptor values followed by the index in dotted-decimal form (e.g., "ifType.7").

6.2.3. exists()

The exists() function is used to verify the existence of an SNMP MIB instance.

int exists(char *objectIdentifier)

objectIdentifier is a NULL terminated string containing an ASCII dotted-decimal representation of an object identifier (e.g. 1.3.6.1.2.1.1.1.0).

The 2-character token "$1" can be used in place of any decimal sub-identifier. This token is expanded by the agent at execution time to contain the index value of each
element the expression acts upon. For example, if the element is interface #7, and the objectIdentifier is "1.3.6.1.2.1.2.2.1.3.$1", it will be expanded to "1.3.6.1.2.1.2.2.1.3.7".

The agent will retrieve the instance in the same SNMP context in which the element resides. Note that no actual SNMP PDU needs to be generated and parsed when the policy MIB module resides on the same system as the managed elements.

This function returns the value 1 if the SNMP instance exists and 0 if it doesn’t exist.

It is recommended that NMS user interfaces display and allow input of MIB object names by their descriptor values followed by the index in dotted-decimal form (e.g., "ifType.7").

6.2.4.  elementName()

The elementName() function is used to determine what the current element is and can be used to provide information about the type of element as well as how it is indexed.

char * elementName(void)

    elementName returns a NULL terminated string containing an ASCII dotted-decimal representation of an object identifier (e.g. 1.3.6.1.2.1.1.1.0). This object identifier identifies an instance of a MIB object that is an attribute of this element.

6.2.5.  strcmp()

The strcmp() function is used to compare two strings.

int strncmp(char *s1, char *s2, n)

    Arguments s1 and s2 are NULL terminated strings.

    Compares at most n bytes of s1 and s2, stopping at the end of either string.

    Returns -1 if s1 is lexicographically less than s2.
Returns 1 if s1 is lexicographically greater than s2.
Returns 0 if s1 and s2 are equal.

6.2.6. lc_strcmp()
The lc_strcmp() function is used to compare two strings.

    int lc_strcmp(char *s1, char *s2)

    Arguments s1 and s2 are NULL terminated strings that are
    translated into lower case strings lc1 and lc2.
    Compares at most n bytes of s1 and s2, stopping at the end of
    either string.
    Then:
    Returns -1 if lc1 is lexicographically less than lc2.
    Returns 1 if lc1 is lexicographically greater than lc2.
    Returns 0 if lc1 and lc2 are equal.

6.2.7. roleMatch()
The roleMatch() function is used to check to see if the
current element has been assigned a particular role.

    int roleMatch(char *roleString)

    Argument roleString is a NULL terminated string. If this
    exactly matches (content and length) any role assigned to the
    current element, the function returns 1. If no roles match,
    the function returns 0.

6.2.8. capMatch()
The capMatch() function is used to check to see if the current
element has a certain capability.

    int capMatch(char *capString)

    Argument capability is a NULL terminated string containing a
    ASCII dotted-decimal representation of an object identifier
    that describes a capability as would be found in the
    pmCapabilitiesTable.
If the current element has the capability described by capString, this function returns 1, otherwise it returns 0.

6.2.9. setint()

The setint() function is used to set a MIB instance to a certain integer value. The setint() function is only valid in policyActions. If when executing a policyFilter, the agent encounters a call to the setint() function, execution of the policyFilter for the current element will immediately terminate and the associated policyAction will not be executed on the current element.

int setint(char *objectIdentifier, int value)

objectIdentifier is a NULL terminated string containing an ASCII dotted-decimal representation of an object identifier (e.g. 1.3.6.1.2.1.1.1.0).

The 2-character token "$1" can be used in place of any decimal sub-identifier. This token is expanded by the agent at execution time to contain the index value of each element the expression acts upon. For example, if the element is interface #7, and the objectIdentifier is "1.3.6.1.2.1.2.2.1.3.$1", it will be expanded to "1.3.6.1.2.1.2.2.1.3.7".

The agent will set the variable specified by objectIdentifier to the value specified by value.

The agent will set the instance in the same SNMP context in which the element resides. Note that no actual SNMP PDU needs to be generated and parsed when the policy MIB module resides on the same system as the managed elements.

If the queried object identifier value does not exist and cannot be created or is not an integer-valued object, 0 is returned. If successful, 1 is returned.

It is recommended that NMS user interfaces display and allow input of MIB object names by their descriptor values followed by the index in dotted-decimal form (e.g., "ifType.7").
6.2.10. setstring()

The setstring() function is used to set a MIB instance to a certain string value. The setstring() function is only valid in policyActions. If when executing a policyFilter, the agent encounters a call to the setstring() function, execution of the policyFilter for the current element will immediately terminate and the associated policyAction will not be executed on the current element.

```
int setstring(char *objectIdentifier, char *value)
```

objectIdentifier is a NULL terminated string containing an ASCII dotted-decimal representation of an object identifier (e.g. 1.3.6.1.2.1.1.1.0).

The 2-character token "$1" can be used in place of any decimal sub-identifier. This token is expanded by the agent at execution time to contain the index value of each element the expression acts upon. For example, if the element is interface #7, and the objectIdentifier is "1.3.6.1.2.1.2.2.1.3.$1", it will be expanded to "1.3.6.1.2.1.2.2.1.3.7".

value is a NULL terminated string containing a sequence of bytes. The agent will set the variable specified by objectIdentifier to the value specified by value.

The agent will set the instance in the same SNMP context in which the element resides. Note that no actual SNMP PDU needs to be generated and parsed when the policy MIB module resides on the same system as the managed elements.

If the queried object identifier value does not exist and cannot be created or is not a string-valued object, 0 is returned. If successful, 1 is returned.

It is recommended that NMS user interfaces display and allow input of MIB object names by their descriptor values followed by the index in dotted-decimal form (e.g., "ifType.7").

6.2.11. setoid()

The setoid() function is used to set a MIB instance to a certain object identifier value. The setoid() function is only
valid in policyActions. If when executing a policyFilter, the agent encounters a call to the setoid() function, execution of the policyFilter for the current element will immediately terminate and the associated policyAction will not be executed on the current element.

```
int setoid(char *objectIdentifier, int value)
```

objectIdentifier is a NULL terminated string containing an ASCII dotted-decimal representation of an object identifier (e.g. 1.3.6.1.2.1.1.1.0).

The 2-character token "$1" can be used in place of any decimal sub-identifier. This token is expanded by the agent at execution time to contain the index value of each element the expression acts upon. For example, if the element is interface #7, and the objectIdentifier is "1.3.6.1.2.1.2.2.1.3.$1", it will be expanded to "1.3.6.1.2.1.2.2.1.3.7".

The agent will set the instance in the same SNMP context in which the element resides. Note that no actual SNMP PDU needs to be generated and parsed when the policy MIB module resides on the same system as the managed elements.

If the queried object identifier value does not exist and cannot be created or is not an object-identifier-valued object, 0 is returned. If successful, 1 is returned.

It is recommended that NMS user interfaces display and allow input of MIB object names by their descriptor values followed by the index in dotted-decimal form (e.g., "ifType.7").
7. Definitions

POLICY-MANAGEMENT-MIB DEFINITIONS ::= BEGIN

IMPORTS
   MODULE-IDENTITY, OBJECT-TYPE,
   Integer32, Gauge32, experimental
   FROM SNMPv2-SMI
   RowStatus, RowPointer
   FROM SNMPv2-TC
   MODULE-COMPLIANCE, OBJECT-GROUP
   FROM SNMPv2-CONF
   SnmpAdminString
   FROM SNMP-FRAMEWORK-MIB;

-- Policy-Based Management MIB

policyMgt MODULE-IDENTITY
   LAST-UPDATED "200007121500Z" -- July 12, 2000
   ORGANIZATION "IETF SNMP Configuration Working Group"
   CONTACT-INFO
      "Steve Waldbusser
      Phone: +1-650-948-6500
      Fax: +1-650-745-0671
      Email: waldbusser@nextbeacon.com

      Jon Saperia
      JDS Consulting, Inc.
      174 Chapman St.
      Watertown MA 02472-3063
      USA
      Phone: +1-617-744-1079
      Fax: +1-617-249-0874
      Email: saperia@jdscons.com

      Thippanna Hongal
      Riverstone Networks, Inc.
      5200 Great America Parkway
      Santa Clara, CA, 95054
      USA
      Phone: +1-408-878-6562
      Fax: +1-408-878-6501
      Email: hongal@riverstonenet.com"
   DESCRIPTION
      "The MIB module for rule-based configuration of SNMP
      infrastructures."

   REVISION "200007121500Z" -- July 12, 2000


DESCRIPTION
"The original version of this MIB, published as RFCXXXX."
::= { experimental 99 }

-- The policy group

pmPolicyTable OBJECT-TYPE
SYNTAX      SEQUENCE OF PmPolicyEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"The policy table. A policy is a pairing of a policyFilter and a policyAction which is used to apply the action to a selected set of elements."
::= { policyMgt 1 }

pmPolicyEntry OBJECT-TYPE
SYNTAX      PmPolicyEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"An entry in the policy table."
INDEX { pmPolicyIndex }
::= { pmPolicyTable 1 }

PmPolicyEntry ::= SEQUENCE {
    pmPolicyIndex              Integer32,
    pmPolicyFilter             OCTET STRING,
    pmPolicyCalendar           RowPointer,
    pmPolicyAction             OCTET STRING,
    pmPolicyFilterMaxLatency   Integer32,
    pmPolicyActionMaxLatency   Integer32,
    pmPolicyPrecedence         Integer32,
    pmPolicyGroup              SnmpAdminString,
    pmPolicyDescription        SnmpAdminString,
    pmPolicyMatches            Gauge32,
    pmPolicyStatus             RowStatus
}

pmPolicyIndex OBJECT-TYPE
SYNTAX      Integer32 (0..65535)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"A unique index for this policy entry."
pmPolicyFilter OBJECT-TYPE
SYNTAX OCTET STRING (SIZE (0..65535))
MAX-ACCESS read-create
STATUS current
DESCRIPTION
"A pmPolicyFilter is an expression which results in a boolean value which represents whether or not an element is a member of a set of elements upon which an action is to be performed.

The format of this expression is the policy expression language. Filter evaluation stops immediately when any error is detected without executing the policyAction.

The pmPolicyFilter is evaluated for various elements. Any element for which the pmPolicyFilter returns any nonzero value will match the filter and will have the associated policyAction executed on that element."

pmPolicyCalendar OBJECT-TYPE
SYNTAX RowPointer
MAX-ACCESS read-create
STATUS current
DESCRIPTION
"A pointer to an entry in the schedTable of the Scheduling MIB [20]. This policy is active when specified by the associated schedule entry.

If the value of this object is 0.0, this policy is always active."

pmPolicyAction OBJECT-TYPE
SYNTAX OCTET STRING
MAX-ACCESS read-create
STATUS current
DESCRIPTION
"A pmPolicyAction is an operation performed on a set of elements. The format of this expression is the policy expression language.

Action evaluation stops immediately when any error is
detected."

::= { pmPolicyEntry 4 }

pmPolicyFilterMaxLatency OBJECT-TYPE
SYNTAX      Integer32 (0..2147483647)
UNITS       "milliseconds"
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
"Every element under the control of this agent is re-checked periodically to see if it is under control of this policy by re-running the filter expression for this policy. This object lets the manager control the maximum amount of time that may pass before an element is re-checked.

In other words, in any given interval of this duration, all elements must be re-checked. Note that it is an implementation-dependent matter as to how the policy agent schedules the checking of various elements within this interval."

::= { pmPolicyEntry 5 }

pmPolicyActionMaxLatency OBJECT-TYPE
SYNTAX      Integer32 (0..2147483647)
UNITS       "milliseconds"
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
"Every element that matches this policy’s filter and is therefore under control of this policy will have this policy’s action executed periodically to ensure that the element remains in the state dictated by the policy. This object lets the manager control the maximum amount of time that may pass before an element has the action run on it.

In other words, in any given interval of this duration, all elements under control of this policy must have the action run on them. Note that it is an implementation-dependent matter as to how the policy agent schedules the policy action on various elements within this interval."

::= { pmPolicyEntry 6 }

pmPolicyPrecedence OBJECT-TYPE
SYNTAX      Integer32 (0..65535)
MAX-ACCESS read-create
STATUS current
DESCRIPTION
 "The order in which policies on the local system are evaluated. A policy with a higher precedence value will be evaluated after a policy with a lower precedence. For example, a policy with a precedence value of 999 will be evaluated after a policy with a precedence value of 998. These values must be unique on the local policy system that realizes this module. The value for a particular policy should be the same across an administrative domain, though that is not mandatory.

When the local policy system performs the evaluation in the pmPolicyFilter for the policy identified by this row it will also read the pmTrackingElementToPolicyStatus object for each object returned as a result of the policy evaluation. If that object is set to modified(3), then the pmPolicyAction shall not be taken on that element.

The value of precedence(4), of pmTrackingElementToPolicyStatus is an indication that when an evaluation was performed by another policy, the pmTrackingElementToPolicyStatus was found to have a value of on(1) and that policy had a higher precedence value than the policy that initially set the value of the pmTrackingElementToPolicyStatus to on(1). In this event, the pmTrackingElementToPolicyPrecedence object shall have the value of the pmPolicyIndex for the policy with the higher precedence value entered. If the policy identified by this row of the pmPolicyTable has a higher precedence value than the value found in pmTrackingElementToPolicyPrecedence then the pmPolicyAction should be performed on the element and the pmTrackingElementToPolicyPrecedence object updated with the value of the pmPolicyIndex for this policy. The only exception to these rules is when the policy that has the higher precedence value is not currently running, i.e., the schedule is off."

 ::= { pmPolicyEntry 7 }

pmPolicyGroup OBJECT-TYPE
SYNTAX SnmpAdminString
MAX-ACCESS read-create
STATUS current
DESCRIPTION
 "An administratively assigned string that is used to group
policies. Any combination is legal, the pmPolicyGroup object
does not constrain precedence. That is precedence is evaluated
independent of grouping though administrators might group
related policies together for clarity."
 ::= { pmPolicyEntry 8 }

pmPolicyDescription OBJECT-TYPE
SYNTAX        SnmpAdminString
MAX-ACCESS    read-create
STATUS        current
DESCRIPTION
 "A description of this rule and its significance, typically
 provided by a human."
 ::= { pmPolicyEntry 9 }

pmPolicyMatches OBJECT-TYPE
SYNTAX        Gauge32
UNITS         "elements"
MAX-ACCESS    read-create
STATUS        current
DESCRIPTION
 "The number of policies that are currently matched by the
 associated pmPolicyFilter."
 ::= { pmPolicyEntry 10 }

pmPolicyStatus OBJECT-TYPE
SYNTAX        RowStatus
MAX-ACCESS    read-create
STATUS        current
DESCRIPTION
 "The status of this pmPolicyEntry."
 ::= { pmPolicyEntry 11 }

-- Element Type Registration Table

-- The Element Type Registration table is used for the manager to
-- learn what element types are being managed by the system and to
-- register new types if necessary. An element type is registered by
-- providing the OID of an SNMP object (i.e., without the
-- instance). Each SNMP instance that exists under that object is a
-- distinct element. The address of the element is the index part of
-- the discovered OID. This address will be supplied to policy filters
-- and actions so that these scripts can inspect and configure the
-- element.
--
Before registering an element type, it is the responsibility of a manager to inspect the table and see if it is already registered (by the agent or another manager). Note that entries that differ only in the last OID (which specifies which object in an entry) are effectively duplicates and should be treated as such by the manager.

```
pmElementTypeRegTable OBJECT-TYPE
SYNTAX      SEQUENCE OF PmElementTypeRegEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
   "A registration table for element types managed by this system."
 ::= { policyMgt 2 }

pmElementTypeRegEntry OBJECT-TYPE
SYNTAX      PmElementTypeRegEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
   "A registration of an element type."
INDEX       { pmElementTypeRegIndex }
 ::= { pmElementTypeRegTable 1 }

PmElementTypeRegEntry ::= SEQUENCE {
   pmElementTypeRegIndex         Integer32,
   pmElementTypeRegOIDPrefix     OBJECT IDENTIFIER,
   pmElementTypeRegName          SnmpAdminString,
   pmElementTypeRegRowStatus     RowStatus
}

pmElementTypeRegIndex OBJECT-TYPE
SYNTAX      Integer32 (0..65535)
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
   "A unique index for this entry."
 ::= { pmElementTypeRegEntry 1 }

pmElementTypeRegOIDPrefix OBJECT-TYPE
SYNTAX      OBJECT IDENTIFIER
MAX-ACCESS  read-create
STATUS      current
DESCRIPTION
"An OBJECT IDENTIFIER subtree under which all instances of this element type may be found.

This OBJECT IDENTIFIER should be specified up to, but not including, any index objects. The agent will discover all instances in the system that are members of the specified subtree. It will then execute policy filters (and potentially policy actions) for each instance discovered.

Each invocation of the policy filter will be supplied with a parameter. This is derived by taking the last N sub-identifiers from the discovered instance, where N is:

\[
X = \text{number of sub-identifiers in pmElementTypeRegOIDPrefix} \\
Y = \text{number of sub-identifiers in discovered instance} \\
N = Y - X
\]

 ::= { pmElementTypeRegEntry 2 }

pmElementTypeRegName OBJECT-TYPE
   SYNTAX    SnmpAdminString
   MAX-ACCESS read-create
   STATUS    current
   DESCRIPTION
      "A description of this registered type."
 ::= { pmElementTypeRegEntry 3 }

pmElementTypeRegRowStatus OBJECT-TYPE
   SYNTAX    RowStatus
   MAX-ACCESS read-create
   STATUS    current
   DESCRIPTION
      "The status of this registration entry."
 ::= { pmElementTypeRegEntry 4 }

-- roleTable

-- The Role Table associates role strings to elements. It is the
-- responsibility of the agent to keep track of any re-indexing of the
-- underlying SNMP variables and to continue to associate role strings
-- with the element with which they were initially configured.
--
-- The agent must store role string associations in NVRAM.
--
-- The Role String table is visible through 2 SNMP tables. The
-- pmRoleESTable is a read-create table that organized role strings
-- sorted by element. This table is used to create and modify role
-- strings and their associations.
-- The pmRoleSETable is a read-only table that organizes role strings
-- sorted by string. This table is read-only.

pmRoleESTable OBJECT-TYPE
SYNTAX      SEQUENCE OF PmRoleESEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"The role string table with element as the major index."
::= { policyMgt 3 }

pmRoleESEntry OBJECT-TYPE
SYNTAX      PmRoleESEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"A role string entry associates a role string with an
individual element."
INDEX       { pmRoleESElement, pmRoleESString }
::= { pmRoleESTable 1 }

PmRoleESEntry ::= SEQUENCE {
    pmRoleESElement    RowPointer,
    pmRoleESString     SnmpAdminString,
    pmRoleESStatus     RowStatus
}

pmRoleESElement OBJECT-TYPE
SYNTAX      RowPointer
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"The element to which this role string is associated.

    If the agent assigns new indexes in the MIB table to
    represent the same underlying element (re-indexing), the
    agent will modify this value to contain the new index for the
    underlying element."
::= { pmRoleESEntry 1 }

pmRoleESString OBJECT-TYPE
SYNTAX SnmpAdminString
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "The role string that is associated with an element through this table."
::= { pmRoleESEntry 2 }

pmRoleESStatus OBJECT-TYPE
SYNTAX RowStatus
MAX-ACCESS read-create
STATUS current
DESCRIPTION "The status of this role string."
::= { pmRoleESEntry 3 }

pmRoleSETable OBJECT-TYPE
SYNTAX SEQUENCE OF PmRoleSEEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "A read-only version of the role string table with roleString as the major index. The purpose of this table is to make it easy to retrieve all elements that share a common string."
::= { policyMgt 4 }

PmRoleSEEntry OBJECT-TYPE
SYNTAX PmRoleSEEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "A role string entry associates a role string with an individual element."
INDEX { pmRoleSEString, pmRoleSEElement }
::= { pmRoleSETable 1 }

PmRoleSEEntry ::= SEQUENCE {
    pmRoleSEString SnmpAdminString,
    pmRoleSEElement RowPointer
}

pmRoleSEString OBJECT-TYPE
SYNTAX SnmpAdminString
MAX-ACCESS not-accessible
Internet Draft    Policy-Based Management MIB    July 12, 2000

STATUS      current
DESCRIPTION
"The role string that is associated with an element through
this table."
::= { pmRoleSEntry 1 }

pmRoleSEnElement OBJECT-TYPE
SYNTAX      RowPointer
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
"The element to which this role string is associated.

If the agent assigns new indexes in the MIB table to
represent the same underlying element (re-indexing), the
agent will modify this value to contain the new index for the
underlying element."
::= { pmRoleSEnEntry 2 }

-- Capabilities table

-- Note that with this table it is not necessary to list all OIDs that
-- a mechanism specific MIB Module supports, just the base OID if
-- the implementation is a fully compliant one. If the implementation
-- is not, then additional rows will exist in the table that list
-- the limitations or enhancements.

pmCapabilitiesTable OBJECT-TYPE
SYNTAX      SEQUENCE OF PmCapabilitiesEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"The pmCapabilitiesTable contains a description of
the inherent capabilities of the system."
::= { policyMgt 5 }

pmCapabilitiesEntry OBJECT-TYPE
SYNTAX      PmCapabilitiesEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
"The description of a capability or limitation of a
capability of the system. An entry will exist for each
domain and mechanism specific ability the system has. In
the case of a domain specific capability with no mechanism
specific parameters, the pmCapabilitiesSubType and all other
columns may be null. Entries will exist that contain
values for the pmCapabilitiesRestrictOID,
pmCapabilitiesRestrictType, pmCapabilitiesRestrictValue
and pmCapabilitiesRestrictString objects only when
an implementation is reporting a mechanism specific
restriction. Multiple entries are possible when more
than one restriction for a type or subtype are needed."

INDEX       { pmCapabilitiesIndex }
::= { pmCapabilitiesTable 1 }

PmCapabilitiesEntry ::= SEQUENCE {
    pmCapabilitiesIndex              Integer32,
    pmCapabilitiesType               OBJECT IDENTIFIER,
    pmCapabilitiesSubType            OBJECT IDENTIFIER,
    pmCapabilitiesModificationOID    OBJECT IDENTIFIER,
    pmCapabilitiesModificationType   INTEGER,
    pmCapabilitiesModificationValue  Integer32,
    pmCapabilitiesModificationString OCTET STRING
}

pmCapabilitiesIndex OBJECT-TYPE
SYNTAX     Integer32 (1..65535)
MAX-ACCESS not-accessible
STATUS      current
DESCRIPTION "A unique index for this entry."
::= { pmCapabilitiesEntry 1 }

pmCapabilitiesType OBJECT-TYPE
SYNTAX     OBJECT IDENTIFIER
MAX-ACCESS read-only
STATUS      current
DESCRIPTION "The type of the capability represented by this entry.
The IANA will publish the list of identifiers that are valid
values for this object."
::= { pmCapabilitiesEntry 2 }

pmCapabilitiesSubType OBJECT-TYPE
SYNTAX     OBJECT IDENTIFIER
MAX-ACCESS read-only
STATUS      current
DESCRIPTION "The sub type of capability is a pointer to a mechanism specific
 Internet Draft    Policy-Based Management MIB    July 12, 2000

        set of capabilities supporting a base technology. In the case of
        DIFFSERV, the OID value here would be the base OID of the
        Differentiated Services Policy MIB Module.

 ::= { pmCapabilitiesEntry 3 }

pmCapabilitiesModificationOID OBJECT-TYPE
SYNTAX      OBJECT IDENTIFIER
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "The OID of the object that is either not supported, supported
with one or more limitations, or expanded by an implementation
specific module. If this columnar object is other than null then
there must be at least an entry in pmCapabilitiesModificationType.
Note that this need not be a leaf node or scalar object. If
an entire table is not supported, this value can be the base OID
for the table."

 ::= { pmCapabilitiesEntry 4 }

pmCapabilitiesModificationType OBJECT-TYPE
SYNTAX      INTEGER {
  unsupported(0),
  restricted(1),
  additional(2),
  addvalue(3),
  maxlimit(4),
  minlimit(5)
}
MAX-ACCESS  read-only
STATUS      current
DESCRIPTION
  "An unsupported value indicates that the OID in
pmCapabilitiesModificationOID is not supported on
this system. A value of 1 indicates that the OID
is supported but with restricted values
These constraints are described in the
pmCapabilitiesModificationValue and
pmCapabilitiesModificationString objects. A value of
2 indicates a vendor specific extension to a standard.
The OID of the new object is pmCapabilitiesModificationOID.
For some implementations, additional functions may be
provided. addvalue indicates that this row of the table
describes an additional value that the object can take.
The specific value is in the pmCapabilitiesModificationValue.
The values of 4 and 5 indicate restrictions or the removal

Various Authors      Expires Jan 12, 2001            [Page 32]
of restrictions for the object identified.

::= { pmCapabilitiesEntry 5 }

pmCapabilitiesModificationValue OBJECT-TYPE
SYNTAX Integer32 (0..2147483647)
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"If the value of pmCapabilitiesModificationType is 0, this object will be null since 0 indicates no support for the object at all. A value of 1 in the pmCapabilitiesModificationType will be further modified by a single integer value in this object that corresponds to enumerated integer values that are not supported by the system for the object that is identified in this row. This value can also represent the limit values in the pmCapabilitiesModificationType object."

::= { pmCapabilitiesEntry 6 }

pmCapabilitiesModificationString OBJECT-TYPE
SYNTAX OCTET STRING
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"Any additional details or description or parameters needed."

::= { pmCapabilitiesEntry 7 }

-- Policy Tracking

pmTrackingPolicyToElementTable OBJECT-TYPE
SYNTAX SEQUENCE OF PmTrackingPolicyToElementEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The pmTrackingPolicyToElementTable describes what elements are under control of a policy."

::= { policyMgt 6 }

pmTrackingPolicyToElementEntry OBJECT-TYPE
SYNTAX PmTrackingPolicyToElementEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"An entry in the pmTrackingPolicyToElementTable."
INDEX { pmPolicyIndex, pmTrackingPolicyToElementElement }
::= { pmTrackingPolicyToElementTable 1 }

PmTrackingPolicyToElementEntry ::= SEQUENCE {
  pmTrackingPolicyToElementElement RowPointer,
  pmTrackingPolicyToElementStatus INTEGER
}

pmTrackingPolicyToElementElement OBJECT-TYPE
SYNTAX RowPointer
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The element this policy is configuring."
::= { pmTrackingPolicyToElementEntry 1 }

pmTrackingPolicyToElementStatus OBJECT-TYPE
SYNTAX INTEGER {
  off(0),
  on(1)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The status of this policy-element relationship. This value
will be 1 if the associated policyFilter returned 1 for this
element and if the calendar for the policy is active.

Entries will only exist in this table if their status is
on(1). Thus, on(1) is the only value of this object that can
be retrieved. This object exists so that it can serve as the
'payload' in the varbind instead of the
pmTrackingPolicyToElementElement object which is much longer
and is already in the index (it would otherwise be
duplicated)."
::= { pmTrackingPolicyToElementEntry 2 }

-- Element to Policy Table

pmTrackingElementToPolicyTable OBJECT-TYPE
SYNTAX SEQUENCE OF PmTrackingElementToPolicyEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"The pmTrackingElementToPolicyTable describes what policies
are controlling an element."
Internet Draft    Policy-Based Management MIB    July 12, 2000

::= { policyMgt 7 }

pmTrackingElementToPolicyEntry OBJECT-TYPE
SYNTAX      PmTrackingElementToPolicyEntry
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
   "An entry in the pmTrackingElementToPolicyTable."
INDEX       { pmTrackingElementToPolicyElement, pmPolicyIndex }
 ::= { pmTrackingElementToPolicyTable 1 }

PmTrackingElementToPolicyEntry ::= SEQUENCE {
   pmTrackingElementToPolicyElement          RowPointer,
   pmTrackingElementToPolicyStatus           INTEGER
}

pmTrackingElementToPolicyElement OBJECT-TYPE
SYNTAX      RowPointer
MAX-ACCESS  not-accessible
STATUS      current
DESCRIPTION
   "The element this policy is configuring."
 ::= { pmTrackingElementToPolicyEntry 1 }

pmTrackingElementToPolicyStatus OBJECT-TYPE
SYNTAX      INTEGER { 
   off(0),
   on(1),
   forceOff(2)
 }
MAX-ACCESS  read-write
STATUS      current
DESCRIPTION
   "The status of this policy-element relationship. This value
    will be 1 if the associated policyFilter returned 1 for this
    element and if the calendar for the policy is active.

    Entries will not exist in this table if their status would be
    off(0).

    A policy can be forcibly disabled on a particular element
    by setting this value to forceOff(2). The agent should then
    act as if the policyFilter failed for this element. The
    forceOff(2) state will persist (even across reboots) until
    this value is set to on(1) by a management request. Even if
the policyFilter later fails for this element, this value will stay in the forceOff(2) state.

::= { pmTrackingElementToPolicyEntry 2 }

-- Compliance Statements

pmConformance OBJECT IDENTIFIER ::= { policyMgt 20 }

pmCompliances OBJECT IDENTIFIER ::= { pmConformance 1 }

pmGroups OBJECT IDENTIFIER ::= { pmConformance 2 }

pmCompliance MODULE-COMPLIANCE

STATUS current

DESCRIPTION

"Describes the requirements for conformance to the Policy-Based Management MIB"

MODULE -- this module

MANDATORY-GROUPS { pmPolicyManagementGroup }

::= { pmCompliances 1 }

pmPolicyManagementGroup OBJECT-GROUP

OBJECTS { pmPolicyFilter, pmPolicyCalendar, pmPolicyAction,

pmPolicyFilterMaxLatency, pmPolicyActionMaxLatency,

pmPolicyPrecedence, pmPolicyGroup,

pmPolicyDescription, pmPolicyMatches, pmPolicyStatus,

pmElementTypeRegOIDPrefix,

pmElementTypeRegName, pmElementTypeRegRowStatus,

pmRoleESStatus, pmRoleSEEElement, pmCapabilitiesType,

pmCapabilitiesSubType, pmCapabilitiesModificationOID,

pmCapabilitiesModificationType,

pmCapabilitiesModificationValue,

pmCapabilitiesModificationString,

pmTrackingPolicyToElementStatus,

pmTrackingElementToPolicyStatus }

STATUS current

DESCRIPTION

"Objects that allow for the creation and management of configuration policies."

::= { pmGroups 1 }

END
8. Security Considerations

There are a number of management objects defined in this MIB that have a MAX-ACCESS clause of read-write and/or read-create. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations.

SNMPv1 by itself is not a secure environment. Even if the network itself is secure (for example by using IPSec), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB.

It is recommended that the implementors consider the security features as provided by the SNMPv3 framework. Specifically, the use of the User-based Security Model RFC 2574 [12] and the View-based Access Control Model RFC 2575 [15] is recommended.

It is then a customer/user responsibility to ensure that the SNMP entity giving access to an instance of this MIB, is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.
9. References


10. 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.

11. Full Copyright Statement

Copyright (C) The Internet Society (2000). 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.
# Table of Contents

1 Abstract .............................................. 1  
2 The SNMP Management Framework ....................... 2  
3 Overview ............................................. 4  
4 Policy-Based Management Architecture .................. 5  
5 Policy Based Management Execution Environment ....... 7  
5.1 Element Discovery .................................... 7  
5.1.1 Implementation Notes .............................. 8  
5.2 Element Filtering ..................................... 9  
5.2.1 Implementation Notes .............................. 9  
5.3 Policy Enforcement .................................. 9  
5.3.1 Implementation Notes .............................. 9  
6 Policy Based Management Expression Language ......... 11  
6.1 Formal Definition .................................... 11  
6.2 Accessor Functions .................................. 12  
6.2.1 getint() ......................................... 13  
6.2.2 getstring() ....................................... 13  
6.2.3 exists() .......................................... 14  
6.2.4 elementName() .................................... 15  
6.2.5 strcmp() ......................................... 15  
6.2.6 lc_strcmp() ....................................... 16  
6.2.7 roleMatch() ....................................... 16  
6.2.8 capMatch() ....................................... 16  
6.2.9 setint() ......................................... 17  
6.2.10 setstring() ...................................... 18  
6.2.11 setoid() ......................................... 18  
7 Definitions ........................................... 20  
8 Security Considerations ................................ 37  
9 References ............................................ 38  
10 Intellectual Property .................................. 39  
11 Full Copyright Statement .............................. 40