Network Working Group
Request for Comments: 4011
Category: Standards Track

S. Waldbusser
Nextbeacon
J. Saperia
JDS Consulting, Inc.
T. Hongal
Riverstone Networks, Inc.

March 2005

Policy Based Management MIB

Status of This Memo

Copyright Notice

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 monitoring and management of Simple Network Management Protocol (SNMP) infrastructures, a scripting language, and a script execution environment.

Table of Contents

1. The Internet-Standard Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to section 7 of RFC 3410 [16].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIv2, which is described in STD 58, RFC 2578 [2], STD 58, RFC 2579 [3], and STD 58, RFC 2580 [4].

2. Overview

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

The following are examples of "business policies":

- All routers will run code version 6.2.
- On-site contractors will only be connected to ports that are configured with special security restrictions.
- 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 variables.

To automate this practice, customers need software tools that will implement business policies across their networks, as well as standard protocols that will ensure that policies can be applied to all of their devices, regardless of the vendor.

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

3. Policy-Based Management Architecture

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

Policies are intended to express a notion of:

> if (an element has certain characteristics) then (apply an operation to that element)

Policies take the following normal form:

```
if (policyCondition) then (policyAction)
```

A policyCondition is a script that results in a boolean to determine whether an element is a member of a set of elements upon which an action is to be performed.

A policyAction is an operation performed on an element or a set of elements.

These policies are most often executed on or near managed devices where the elements live (and thus their characteristics may be easily inspected) and where operations on those elements will be performed.

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

An element is an instance of a physical or logical entity and is embodied by 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.

Conceptually, policies are executed in the following manner:

```
for each element for which policyCondition 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 (circuit w/gold service paid for)  then (apply special queuing)
```

Each unique combination of policy and element is called an execution context. Within a particular execution context, the phrase 'this element' is often used to refer to the associated element, as most policy operations will be applied to 'this element'. The address of 'this element' contains the object identifier of any attribute of the element, the SNMP context the element was discovered in, and the address of the system on which the element was discovered.

Policies can manage elements on the same system:

```
         -----------------------------------------------------
         |                                                   |
         |              Managed System                       |
         |                                                   |
         |                                                   |
         |   ------------------             Managed Elements |
         |   |                |               interfaces     |
         |   | Policy Manager | manages...    circuits       |
         |   |                |               queues         |
         |   ------------------               processes      |
         |                                    ...            |
         |                                                   |
         -----------------------------------------------------
```

or they can manage elements on other systems:

```
                                            --------------------------
                                            |  Managed System        |
     --------------------------             |    Managed Elements    |
     |                        |             |      interfaces        |
     |  Management Station or |             |      circuits          |
     |    Mid-Level Manager   |             |      ...               |
     |                        |             --------------------------
     |   ------------------   | manages...
     |   | Policy Manager |   |             --------------------------
     |   ------------------   |             |  Managed System        |
     |                        |             |    Managed Elements    |
     --------------------------             |      interfaces        |
                                            |      circuits          |
                                            |      ...               |
                                            --------------------------

                                            ...
```

PolicyConditions have the capability of performing comparison operations on SNMP variables, logical expressions, and other functions. Many device characteristics are already defined in MIB Modules and are easy to include in policyCondition 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 administratively specified characteristic of a managed element. As a selector for policies, it determines the applicability of the policy to a particular managed element.

Some examples of roles are political, financial, legal, geographical, or architectural characteristics, 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.

Some types of information one would put into a role include the following:

```
political - describes the role of a person or group of people, or of
            a service that a group of people uses.  Examples:
            executive, sales, outside-contractor, customer.
     If (attached user is executive) then (apply higher bandwidth)
     If (attached user is outside-contractor) 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.
     If (gold service paid for) then (apply special queuing)

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.  Examples: backup, trunk.
     If (interface is backup) then (set ifAdminStatus = down)
```

Roles in this model are human-defined strings that can be referenced by policy code. The role table in this MIB may be used to assign role strings to elements and to view all role string assignments. Implementation-specific mechanisms may also be used to assign role strings; however, these assignments must be visible in the role table. Multiple roles may be assigned to each element. Because policy code has access to data in MIB objects that represent the current state of the system and (in contrast) role strings are more static, it is recommended that role strings not duplicate information available in MIB objects. Role strings generally should be used to describe information not accessible in MIB objects.

Policy scripts may inspect role assignments to make decisions based on whether an element has a particular role assigned to it.

The pmRoleTable allows a management station to learn what roles exist on a managed system. The management station may choose not to install policies that depend on a role that does not exist on any elements in the system. The management station can then register for notifications of new roles. Upon receipt of a pmNewRoleNotification, it may choose to install new policies that make use of that new role.

Capabilities

The capabilities table allows a management station to learn what capabilities exist on a managed system. The management station may choose not to install policies that depend on a capability that does not exist on any elements in the system. The management station can then register for notifications of new capabilities. Upon receipt of a pmNewCapabilityNotification, it may choose to install new policies that make use of that new capability.

Time

Managers may wish to define policies that are intended to apply for certain periods of time. This might mean that a policy is installed and is dormant for a period of time, becomes ready, and then later goes dormant again. Sometimes these time periods will be regular (Monday-Friday 9-5), and sometimes ad hoc. This MIB provides a schedule table that can schedule when a policy is ready and when it is dormant.

A policy manager contains the following:

```
         -------------------------------------------------------
         | Policy Manager                                      |
         |                                                     |
         |   ----------------------------------------          |
         |   | Agent                                |          |
         |   |                                      |          |
         |   |  ---------------------------------   |          |
         |   |  | Policy Download and Control   |   |          |
         |   |  |   pmPolicyTable               |   |          |
         |   |  |   pmElementTypeRegTable       |   |          |
         |   |  |   pmSchedTable                |   |          |
         |   |  ---------------------------------   |          |
         |   |                                      |          |
         |   |  ---------------------------------   |          |
         |   |  | Policy Environment Control    |   |          |
         |   |  |   pmRoleTable                 |   |          |
         |   |  |   pmCapabilitiesTables        |   |          |
         |   |  ---------------------------------   |          |
         |   |                                      |          |
         |   |  ---------------------------------   |          |
         |   |  | Policy Monitoring             |   |          |
         |   |  |   pmTrackingTables            |   |          |
         |   |  |   pmDebuggingTable            |   |          |
         |   |  ---------------------------------   |          |
         |   ----------------------------------------          |
         |                                                     |
         |   --------------------------------                  |
         |   | Execution Environment        |                  |
         |   |                              |                  |
         |   |  -----------------------     |                  |
         |   |  | Policy Scheduler    |     |                  |
         |   |  -----------------------     |                  |
         |   |  -----------------------     |                  |
         |   |  | Language            |     |                  |
         |   |  -----------------------     |                  |
         |   |  -----------------------     |                  |
         |   |  | Function Library    |     |                  |
         |   |  -----------------------     |                  |
         |   --------------------------------                  |
         -------------------------------------------------------
```

4. Policy-Based Management Execution Environment

4.1. Terminology

Active Schedule - A schedule specifies certain times that it will be considered active. A schedule is active during those times.

Valid Policy - A valid policy is a policy that is fully configured and enabled to run. A valid policy may run unless it is linked to a schedule entry that says the policy is not currently active.

Ready Policy - A ready policy is a valid policy that either has no schedule or is linked to a schedule that is currently active.

Precedence Group - Multiple policies can be assigned to a precedence group with the resulting behavior that for each element, of the ready policies that match the condition, only the one with the highest precedence value will be active. For example, if there is a default bronze policy that applies to any interface and a special policy for gold interfaces, the higher precedence of the gold policy will ensure that it is run on gold ports and that the bronze policy isn’t.

Active Execution Context - An active execution context is a pairing of a ready policy with an element that matches the element type filter and the policy condition. If there are multiple policies in the precedence group, it is also necessary that no higher precedence policy in the group match the policy condition.

Run-Time Exception (RTE) - A run-time exception is a fatal error caused in language or function processing. If, during the invocation of a script, a run-time exception occurs, execution of that script is immediately terminated. If a policyCondition experiences a run-time exception while processing an element, the element is not matched by the condition and the associated action will not be run on that element. A run-time exception can cause an entry to be added to the pmDebuggingTable and will be reflected in the pmTrackingPEInfo object.

4.2. Execution Environment - Elements of Procedure

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

- Element Discovery
- Element Filtering
- Policy Enforcement

4.3. Element Discovery

An element is an instance of a physical or logical entity. 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 MIB Modules. As long as the indexing is consistent between these tables, the entity can be modeled as one 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 allows 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 index part of the discovered OID will be supplied to policy conditions and actions so that this code can inspect and configure the element. The agent can determine the index portion of discovered OIDs based on the length of the pmElementTypeRegOIDPrefix for the portion of the MIB that is being retrieved. For example, if the OIDPrefix is ’ifEntry’, which has 9 subids, the index starts on the 11th subid (skipping the subidentifier for the column; e.g., ifSpeed).

For each element that is discovered, the policy condition is called with the element’s name as an argument to see whether the element is a member of the set the policy acts upon.

Note that agents may automatically configure entries in this table for frequently used element types (interfaces, circuits, etc.). In particular, it may configure elements for which discovery is optimized in one or both of the following ways:

1. The agent may discover elements by scanning internal data structures as opposed to issuing local SNMP requests. It is possible to recreate the exact semantics described in this table even if local SNMP requests are not issued.

2. The agent may receive asynchronous notification of new elements (for example, "card inserted") and use that information to create elements instantly rather than through polling. A similar feature might be available for the deletion of elements.

Note that upon restart, the disposition of agent-installed entries is described by the pmPolicyStorageType object.

A special element type "0.0" represents the "system element". "0.0" represents the single instance of the system itself and provides an execution context for policies to operate on "the system" and on MIB objects modeled as scalars. For example, "0.0" gives an execution context for policy-based selection of the operating system code version (likely modeled as a scalar MIB object). The element type "0.0" always exists. As a consequence, no actual discovery will take place and the pmElementTypeRegMaxLatency object will have no effect for the "0.0" element type. However, if the "0.0" element type is not registered in the table, policies will not be executed on the "0.0" element.

If the agent is discovering elements by polling, it should check for new elements no less frequently than pmElementTypeRegMaxLatency would dictate. When an element is first discovered, all policyConditions are run immediately, and policyConditions that match will have the associated policyAction run immediately. Subsequently, the policyCondition will be run regularly for the element, with no more than pmPolicyConditionMaxLatency milliseconds elapsing between each invocation. Note that if an implementation has the ability to be alerted immediately when a particular type of element is created, it is urged to discover that type of element in this fashion rather than through polling, resulting in immediate configuration of the discovered element.

4.3.1. Implementation Notes

Note that although 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 (such as 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, a manager is responsible for inspecting the table to see whether it is already registered (either by the agent or by another manager). Note that entries that differ only in the last subid (which specifies which object is an entry) are effectively duplicates and should be treated as such by the manager.

The system that implements the Policy-Based Management MIB may not have knowledge of the format of object identifiers in other MIB Modules. 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 the ifTable is registered, 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.

4.4. Element Filtering

The first step in executing a policy is to see whether the policy is ready to run based on its schedule. If the pmPolicySchedule object is equal to zero, there is no schedule defined, and the policy is always ready. If the pmPolicySchedule object is non-zero, then the policy is ready only if the referenced schedule group contains at least one valid schedule entry that is active at the current time.

If the policy is ready, the next step in executing a policy is to see which elements match the policy condition. The policy condition is called once for each element and runs to completion. The element’s name is the only argument that is passed to the condition code for each invocation. No state is remembered within the policy script from the previous invocation of ’this element’ or from the previous invocation of the policy condition, except for state accessible through library functions. Two notable examples of these are the scratchpad functions, which explicitly provide for storing state, and the SNMP functions, which can store state in local or remote MIB objects. If any run-time exception occurs, the condition will terminate immediately for ’this element’. If the condition returns non-zero, the corresponding policy action will be executed for ’this element’.

If an element matches a condition and it had not matched that condition the last time it was checked (or if it is a newly discovered element), the associated policyAction will be executed immediately. If the element had matched the condition at the last check, it will remain in the set of elements whose policyAction will be run within the policyActionMaxLatency.

4.4.1. Implementation Notes

Whether policy conditions are multi-tasked is an implementation-dependent matter. Each condition/element combination is conceptually its own process and can be scheduled sequentially, or two or more could be run simultaneously.

4.5. Policy Enforcement

For each element that has returned non-zero from the policy condition, the corresponding policy action is called. The element’s name is the only argument that is passed to the policy action for each invocation. Except for state accessible from library functions, no state is remembered from the policy condition evaluation, or from the previous condition/action invocation of ’this element’ or from the previous invocation of the policy condition or action on any other element. If any run-time exception occurs, the action will terminate immediately for ’this element’.

4.5.1. Implementation Notes

How policy actions are multi-tasked is an implementation-dependent matter. Each condition/element combination is conceptually its own process and can be scheduled sequentially, or two or more could be run simultaneously.

5. The PolicyScript Language

Policy conditions and policy actions are expressed with the PolicyScript language. The PolicyScript language is designed to be a small interpreted language that is simple to understand and implement; it is designed to be appropriate for writing small scripts that make up policy conditions and actions.

PolicyScript is intended to be familiar to programmers that know one of several common languages, including Perl and C. Nominally, policyScript is a subset of the C language; however, it was desirable to have access to C++’s operator overloading (solely to aid in documenting the language). Therefore, PolicyScript is defined formally as a subset of the C++ language in which many of the operators are overloaded as part of the "var" class. Note, however, that a PolicyScript program cannot further overload operators, as the syntax to specify overloading is not part of the PolicyScript syntax. A subset was used to provide for easy development of low-cost interpreters of PolicyScript and to take away language constructs that are peculiar to the C/C++ languages. For example, it is expected that both C and Perl programmers will understand the constructs allowed in PolicyScript.

Some examples of the C/C++ features that are not available are function definitions, pointer variables, structures, enums, typedefs, floating point and pre-processor functions (except for comments).

This language is formally defined as a subset of ISO C++ [10] but only allows constructs that may be expressed in the Extended Backus-Naur Form (EBNF) documented here. This is because although EBNF 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 the language that is required for conformance to this specification. Unless explicitly described herein, the meaning of any construct expressed in the EBNF can be found by reference to the ISO C++ standard.

The use of comments and newlines are allowed and encouraged in order to promote readability of PolicyScript code. Comments begin with '/*' and end with '*/' or begin with '//' and go until the end of the line.

One subset is not expressible in the EBNF syntax: all variables within an instance of a PolicyScript script are within the same scope. In other words, variables defined in a block delimited with '{' and '}' are not in a separate scope from variables in the enclosing block.

PolicyScript code must be expressed in the ASCII character set.

In the EBNF used here, terminals are character set members (singly or in a sequence) that are enclosed between two single-quote characters or described as a phrase between '<' and '>' characters. Nonterminals are a sequence of letters and underscore characters. A colon (:) following a nonterminal introduces its definition, a production. In a production, a '|' character separates alternatives. The '(' and ')' symbols group the enclosed items. The '[' and ']' symbols indicate that the enclosed items are optional. A '?' symbol following an item indicates that the item is optional. A '*' symbol following an item indicates that the item is repeated zero, one, or more times. A '+' symbol following an item indicates that the item is repeated one or more times. The symbol '--' begins a comment that ends at the end of the line.

5.1. Formal Definition

The PolicyScript language follows the syntax and semantics of ISO C++ [10], but is limited to that which can be expressed in the EBNF below.

The following keywords are reserved words and cannot be used in any policy script. This prevents someone from using a common keyword in another language as an identifier in a script, thereby confusing the meaning of the script. The reserved words are:

> auto, case, char, const, default, do, double, enum, extern, float, goto, inline, int, long, register, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, and volatile.

Any syntax error, use of a reserved keyword, reference to an unknown identifier, improper number of function arguments, error in coercing an argument to the proper type, exceeding local limitations on string length, or exceeding local limitations on the total amount of storage used by local variables will cause an RTE.

PolicyScript permits comments using the comment delimiters, ’/*’ to ’*/’, or the start of comment symbol ’//’.

```
-- Lexical Grammar

   letter:       ’_’ | ’a’ | ’b’ | ’c’ | ’d’ | ’e’ | ’f’
               | ’g’ | ’h’ | ’i’ | ’j’ | ’k’ | ’l’ | ’m’
               | ’n’ | ’o’ | ’p’ | ’q’ | ’r’ | ’s’ | ’t’
               | ’u’ | ’v’ | ’w’ | ’x’ | ’y’ | ’z’
               | ’A’ | ’B’ | ’C’ | ’D’ | ’E’ | ’F’
               | ’G’ | ’H’ | ’I’ | ’J’ | ’K’ | ’L’ | ’M’
               | ’N’ | ’O’ | ’P’ | ’Q’ | ’R’ | ’S’ | ’T’
               | ’U’ | ’V’ | ’W’ | ’X’ | ’Y’ | ’Z’

   digit:        ’0’ | ’1’ | ’2’ | ’3’ | ’4’
               | ’5’ | ’6’ | ’7’ | ’8’ | ’9’

   non_zero:   ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’

   oct_digit:  ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’

   hex_digit:    digit | ’a’ | ’b’ | ’c’ | ’d’ | ’e’ | ’f’
                       | ’A’ | ’B’ | ’C’ | ’D’ | ’E’ | ’F’

   escape_seq:    ’\’’   |   ’\"’   |   ’\?’   |   ’\\’
                | ’\a’   |   ’\b’   |   ’\f’   |   ’\n’
                | ’\r’   |  ’\t’    |   ’\v’
                | ’\’ oct_digit+    | ’\x’ hex_digit+

   non_quote:  Any character in the ASCII character set
               except single quote (’), double quote ("),
               backslash (’\’), or newline.

   c_char:            non_quote | ’"’ | escape_seq

   string_literal:    ’"’ s_char* ’"’

   s_char:            non_quote | ’’’ | escape_seq

   char_constant:     ’’’ c_char ’’’

   decimal_constant:  non_zero digit*
```

```
   octal_constant:    ’0’ oct_digit*

   hex_constant:      ( ’0x’ | ’0X’ ) hex_digit+

   integer_constant:  decimal_constant | octal_constant | hex_constant

   identifier:        letter ( letter | digit )*

-- Phrase Structure Grammar

   -- Expressions

   primary_expr:      identifier | integer_constant | char_constant
                    | string_literal  |  ’(’ expression ’)’

   postfix_expr:      primary_expr
                    | identifier ’(’ argument_expression_list? ’)’
                    | postfix_expr ’++’
                    | postfix_expr ’--’
                    | postfix_expr ’[’ expression ’]’

   argument_expression_list:
                      assignment_expr
                    | argument_expression_list ’,’ assignment_expr

   unary_expr:        postfix_expr  |  unary_op unary_expr

   unary_op:          ’+’ | ’-’ | ’~’ | ’!’ | ’++’ | ’--’

   binary_expr:  unary_expr | binary_expr binary_op unary_expr

   binary_op:       ’||’ | ’&&’ | ’|’  | ’^’  | ’&’  | ’!=’
                  | ’==’ | ’>=’ | ’<=’ | ’>’  | ’<’  | ’>>’
                  | ’<<’ |  ’-’ | ’+’  | ’%’  | ’/’  |  ’*’

   assignment_expr:      binary_expr
                       | unary_expr assignment_op assignment_expr

   assignment_op:     ’=’ | ’*=’  | ’/=’ | ’%=’ | ’+=’ | ’-=’
                  | ’<<=’ | ’>>=’ | ’&=’ | ’^=’ | ’|=’

   expression:    assignment_expr | expression ’,’ assignment_expr

   -- Declarations

   declaration:       ’var’ declarator_list ’;’
```

```
   declarator_list:   init_declarator
                    | declarator_list ',' init_declarator

   init_declarator:   identifier [ '=' assignment_expr ]

   -- Statements

   statement:   declaration
              | compound_statement
              | expression_statement
              | selection_statement
              | iteration_statement
              | jump_statement

   compound_statement:    '{' statement* '}'

   expression_statement:  expression? ';'

   selection_statement:
           'if' '(' expression ')' statement
         | 'if' '(' expression ')' statement 'else' statement

   iteration_statement:
           'while' '(' expression ')' statement
         | 'for' '(' expression? ';' expression? ';' expression? ')'
               statement

   jump_statement:    'continue' ';'
                    | 'break' ';'
                    | 'return' expression? ';'

   -- Root production

   PolicyScript:     statement*
```

5.2. Variables

To promote shorter scripts and ease in writing them, PolicyScript provides a loosely typed data class, "var", that can store both integer and string values. The native C++ types (char, int, etc.) are thus unnecessary and have not been carried into the subset that comprises this language. The semantics of the "var" type are modeled after those of ECMAScript[17].

For example:

```
var number = 0, name = "IETF";
```

This language will be executed in an environment where the following typedef is declared. (Note that this typedef will not be visible in the policyCondition or policyAction code.)

```
typedef ... var;
```

Although this declaration is expressed here as a typedef, the 'typedef' keyword itself is not available to be used in PolicyScript code.

5.2.1. The Var Class

A value is an entity that takes on one of two types: string or integer.

The String type is the set of all finite ordered sequences of zero or more 8-bit unsigned integer values ("elements"). The string type can store textual data as well as binary data sequences. Each element is considered to occupy a position within the sequence. These positions are indexed with nonnegative integers. The first element (if any) is at position 0, the next element (if any) at position 1, and so on. The length of a string is the number of elements (i.e., 8-bit values) within it. The empty string has length zero and therefore contains no elements.

The integer type is the set of all integer values in the range -9223372036854775808 (-2^63) to 18446744073709551615 (2^64-1). If an integer operation would cause a (positive) overflow, then the result is returned modulo 2^64. If an integer operation would cause a (negative) underflow, then the result is undefined. Integer division rounds toward zero.

Prior to initialization, a var object has type String and a length of zero.

The policy script runtime system performs automatic type conversion as needed. To clarify the semantics of certain constructs it is useful to define a set of conversion operators: ToInteger(), ToString(), ToBoolean(), and Type(). These operators are not a part of the language; they are defined here to aid the specification of the semantics of the language. The conversion operators are polymorphic; that is, they can accept a value of any standard type.

ToInteger

The operator ToInteger converts its argument to a value of type Integer according to the following table:

Integer	The result equals the input argument (no conversion).
String	See grammar and note below.
integer_constant	The result equals the input argument (no conversion).
string_literal	See grammar and note below.
char_constant	See grammar and note below.

ToInteger Applied to Strings

ToInteger applied to the String Type string_literal and to char_constants applies the following grammar to the input. If the grammar cannot interpret the string as an expansion of numeric_string, then an RTE is generated. Note that a numeric_string that is empty or contains only white space is converted to 0.

```
 -- EBNF for numeric_string

   numeric_string : white_space* numeric? white_space*

   white_space :      <TAB> |  <SP> |  <NBSP> |  <FF> |  <VT>
                    | <CR>  |  <LF> |  <LS>   |  <PS> |  <USP>

   numeric :         signed_decimal |  hex_constant | octal_constant |
                     enum_decimal

   signed_decimal:  [ ’-’ | ’+’ ] decimal_constant

   enum_decimal:    [ letter | digit | ’-’ ]* ’(’ decimal_constant ’)’

   -- decimal_constant, hex_constant, and octal_constant are defined
   -- in the PolicyScript EBNF described earlier.
```

Note that when the enum_decimal form is converted, the sequence of characters before the parenthesis and the pair of parenthesis themselves are completely ignored, and the decimal_constant inside the parenthesis is converted. Thus, "frame-relay(32)" translates to the integer 32.

Although this will make the script more readable than using the constant "32", the burden is on the code writer to be accurate, as "ethernet-csmacd(32)" and "frame-relay(999)" will also be accepted.

ToString

The operator ToString converts its argument to a value of type String according to the following table:

Integer	Return the string containing the decimal representation of the input argument in the form of signed_decimal, except that no leading ’+’ will be used.
String	Return the input argument (no conversion)
integer_constant	Return the string containing the decimal representation of the input argument in the form of signed_decimal except that no leading ’+’ will be used.
string_literal	Return the input argument (no conversion)
char_constant	Return the string of length one containing the value of the input argument.

ToBoolean

The operator ToBoolean converts its argument to a value of type Integer according to the following table:

Integer	The result is 0 if the argument is 0. Otherwise the result is 1.
String	The results is 0 if the argument is the empty string. Otherwise the result is 1.
integer_constant	The result is 0 if the argument is 0. Otherwise the result is 1.
string_literal	The result is 0 if the argument is the empty string. Otherwise the result is 1.
char_constant	The result is 1.

Operators

The rules below specify the type conversion rules for the various operators.

```
A++:   A = ToInteger(A); A++;
A--:   A = ToInteger(A); A--;
++A:   A = ToInteger(A); ++A;
--A:   A = ToInteger(A); --A;
+A:    ToInteger(A);
-A:     -1 * ToInteger(A);
˜A:    ToInteger(A);
!A:    !ToBoolean(A);
A * B, A - B, A & B, A ^ B , A | B, A << B, A >> B:
       ToInteger(A) <operator> ToInteger(B)
```

```
      A / B, A % B:
             if (ToInteger(B) == 0)
               RTE, terminate;
             else
               ToInteger(A) <operator> ToInteger(B)
      A + B:
             if (Type(A) == String || Type(B) == String)
               ToString(A) concatenated with ToString(B)
             else
               A + B
      Compound Assignment (<operator>=):
              Simply follow rules above.  Note that type of LHS (Left
              Hand Side) may be changed as a result.

      A < B, A > B, A <= B, A >= B, A == B, A != B:
             if (Type(A) == String && Type(B) == String)
                 lexically compare strings with strcmp() logic
             else
                 ToInteger(A) <operator> ToInteger(B)
       A && B:
              if (ToBoolean(A))
                  ToBoolean(B);
              else
                  false;
       A || B:
              if (ToBoolean(A))
                  true;

              else
                  ToBoolean(B);

       if(A):
              if (ToBoolean(A))
       while(A):
              while(ToBoolean(A))
       for(...; A; ...):
             for(...; ToBoolean(A); ...)

       A[B] as a RHS (Right Hand Side) value:
             if (Type(A) != String
                  || ToInteger(B) >= strlen(A))
                RTE, terminate;
             A[ ToInteger(B) ]
             The contents are returned as a string of length one

        A[B] = C as a LHS value:
             if (Type(A) != String
                  || ToInteger(B) >= strlen(A))
```

```
                RTE, terminate;
             if (strlen(ToString(C)) == 0)
                RTE, terminate
             A[ ToInteger(B) ] = First octet of ToString(C)
```

Note that this is only applicable in a simple assignment.

For example, in the expression

"getVar("ifSpeed.1") < 128000"

getVar always returns a string and ’128000’ is implicitly an integer. The rules for ’<’ dictate that if either argument is an integer then a ’numeric less than’ is performed on ToInteger(A) and ToInteger(B).

If "getVar("ifSpeed.1")" returns "64000", the expression can be translated to:

```
ToInteger("64000") < ToInteger(128000); or,
64000 < 128000; or,
True
```

5.3. PolicyScript QuickStart Guide

PolicyScript is designed so that programmers fluent in other languages can quickly begin to write scripts.

One way to become familiar with a language is to see it in action. The following nonsensical script exercises most of the PolicyScript constructs (though it skips some usage options and many arithmetic operators).

```
var x, index = 7, str = "Hello World", oid = "ifSpeed.";

x = 0;
while(x < 10){
    if (str < "Goodbye") /* string comparison */
        continue;
    else
        break;
    x++;
}
if (oidlen(oid) == 10)
    oid += "." + index; // append index to oid
for(x = 0; x < 7; x++){
      str += "a";
```

```
      var y = 12;
      index = ((x * 7) + y) % 3;
      if (str[6] == 'W')
          return index;
}
return;
```

The following examples are more practical:

For a condition:

```
// Return 1 if this is an interface and it is tagged
// with the role "gold"
return (inSubtree(elementName(), "ifEntry")
    && roleMatch("gold"))
```

A condition/action pair:
First, register the Host Resources MIB hrSWRunEntry as a new element in the pmElementTypeRegTable. This will cause the policy to run for every process on the system. The token '$*' will be replaced by the script interpreter with a process index (see Section 7 for a definition of the '$*' token).

The condition:

```
// if it's a process and it's an application and it's
// consumed more than 5 minutes of CPU time
return (inSubtree(elementName(), "hrSWRunEntry")
        && getVar("hrSWRunType.$*") == 4  // app, not OS or driver
        && getVar("hrSWRunPerfCPU.$*") > 30000) // 300 seconds
```

The action:

```
// Kill it
setVar("hrSWRunStatus.$*", 4, Integer); // invalid(4) kills it
```

A more substantial action to start an RMON2 host table on interfaces that match the condition:

```
var pdu, index;

pdu = newPDU();
writeVar(pdu, 0, "hlHostControlDataSource.*",
         "ifIndex." + ev(0), Oid);
writeVar(pdu, 1, "hlHostControlNlMaxDesiredEntries.*", 1000,
         Integer);
writeVar(pdu, 2, "hlHostControlAlMaxDesiredEntries.*", 1000,
         Integer);
writeVar(pdu, 3, "hlHostControlOwner.*", "policy", String);
```

```
      writeVar(pdu, 4, "hlHostControlStatus.*", "active(1)", Integer);
      if (createRow(pdu, 5, 4, 20, 65535, index) == 0
          || index == -1)
          return;
```

Because PolicyScript is a least common denominator, it contains nothing that would astonish programmers familiar with C, C++, Perl, Tcl, JavaScript, or Python. Although a new programmer may attempt to use language constructs that aren't available in PolicyScript, s/he should be able to understand any existing PolicyScript and will likely know how to use anything that is valid in PolicyScript. The lists below quickly enumerate the changes of note for programmers coming from some particular languages. These lists won't describe the unavailable constructs, but it is easy to see from the definition above what is available.

5.3.1. Quickstart for C Programmers

- Character constants (i.e., 'c') are treated as one-character strings, not as integers. So operations such as ('M' - 'A') or (x + 'A') will not perform as expected.
- Functions can change the value of arguments even though they are not pointers (or called like '&arg').
- All variables are in the same scope.

5.3.2. Quickstart for Perl Programmers

- Comments are '/* comment */' and '// till end of line', not '#'.
- No need to put a '$' in front of variables.
- Strings are compared with ==, <=, <, etc. (details in Sec. 6.2.1).
- Strings are concatenated with '+' (details in Sec. 6.2.1).
- No variable substitution in "" strings. '' strings are 1 char only.
- Variables must be declared before use (but no type is necessary).
- All variables are in the same scope.

5.3.3. Quickstart for TCL Programmers

- Comments are '/* comment */' and '// till end of line', not '#'.
- No need to put a '$' in front of variables.
- Function calls are func-name(arg1, arg2, ...).
- Square braces [] don't interpret their contents.
- Double quotes "" surround a string, but no substitutions are performed ("" is like { } in TCL).
- Statements are terminated by a semicolon (;).
- Instead of "Set a b", use "b = a;".
- Strings are concatenated with '+' (details in Sec. 6.2.1).
- All variables are in the same scope.

5.3.4. Quickstart for Python Programmers

- Comments are '/* comment */' and '// till end of line', not '#'.
- Single quotes can be used only for single-character strings ('a').
- Indentation doesn't matter. Braces { } define blocks.
- Variables must be declared before use (but no type is necessary).
- The expressions for if and while are always surrounded by parenthesis, as in "if (x < 5)".
- 'for' syntax is "for(expression; expression; expression)" (see EBNF).
- All variables are in the same scope.

5.3.5. Quickstart for JavaScript/ECMAScript/JScript Programmers

- Variables must be declared before use.
- Functions can change the value of arguments.
- All variables are in the same scope.

5.4. PolicyScript Script Return Values

A PolicyScript script execution is normally ended by the execution of a return statement, or by having the flow of execution reach the end of the final statement in the script. A normal script execution always returns a Boolean value. If no explicit value is specified in the return statement, or if the flow of control proceeds through the end of the script, the return value is implicitly zero. If an expression is provided with the return statement, the expression is evaluated, and the result of the expression is implicitly converted with the ToBoolean operator before being returned to the script execution environment.

The return value of a policyCondition script is used to determine whether the associated policyAction script is executed. If the returned value is zero, the associated policyAction script is not executed. If the returned value is one, the associated policyAction script will be executed.

The return value of a policyAction script is ignored.

An RTE or invocation of the fail() function will cause the return value of the script to be set to zero. Note however, that execution of the defer() or fail() functions may set the defer attribute so that the lower precedence script may be executed. This is independent of the return value of the policy script execution.

6. Index Information for ’this element’

PolicyScript code needs a convenient way to get the components of the index for ’this element’ so that they can perform SNMP operations on it or on related elements.

Two mechanisms are provided.

1. For all OID input parameters to all SNMP Library Functions (but not OID utility functions), the token "$n" (’$’ followed by an integer between 0 and 128) can be used in place of any decimal sub-identifier. This token is expanded by the agent at execution time to contain the nth subid of the index for the current element. For example, if the element is interface 7, and the objectIdentifier is "1.3.6.1.2.1.2.2.1.3.$0", it will be expanded to "1.3.6.1.2.1.2.2.1.3.7". The special token "$*" is expanded to contain all of the subidentifiers of the index of the current element, separated by ’.’ characters.

   It is an RTE if a token is specified that is beyond the length of the index for the current element.

   Note that the "$n" convention is only active within strings.

2. The ec() and ev() functions allow access to the components of the index for ’this element’. ec() takes no argument and returns the number of index components that exist. ev() takes an integer argument specifying which component of the index (numbered starting at 0) and returns an integer containing the value of the n’th subidentifier. Refer to the Library functions section for the complete definition of ec() and ev().

```
For example, if 'this element' is frCircuitDLCI.5.57
                                  (ifIndex = 5, DLCI = 57)
      then ec()  returns 2
           ev(0) returns 5
           ev(1) returns 57
```

   This is helpful when one wishes to address a related element. Extending the previous example, to find the port speed of the port, the circuit (above) runs over:

```
portSpeed = getVar("ifSpeed." + ev(0));
```

   A script may check the type of ’this element’ by calling the elementName() function. Although it is possible to write a script that will work with different types of elements, many scripts will assume a particular element type and will work incorrectly if used on different element types.

7. Library Functions

Library functions are built-in functions available primarily to provide access to information on the local system or to manipulate this information more efficiently. A group of functions is organized into a library, the unit of conformance for function implementation. In order to claim conformance to a library, an implementation must implement all functions in a library to the specifications of the library.

In order for a management station or a condition or action to understand whether a certain library of functions is implemented, each library will have a name that it registers in the role table as a characteristic of the system element ("0.0") in the default SNMP context. Thus, conformance to a library can be tested with the roleMatch library function (in the base library) with the call roleMatch ("libraryName", "0.0").

Note that in the descriptions of these functions below, the function prototype describes the type of argument expected. Even though variables are not declared with a particular type, their contents must be appropriate for each function argument. If the type is variable, the keyword 'var' will be used. If only a string is appropriate, the keyword 'string' will be used. If only an integer is appropriate, the keyword 'integer' will be used. If the argument is declared as 'string' or 'integer' and a value of a different type is passed, the argument will be coerced with ToInteger() or ToString(). Any failure of this coercion will cause an RTE (in particular for ToInteger(), which will fail if its string-valued argument is not a well-formed integer).

In the function prototype, if the '&' character precedes the identifier for an argument, that argument may be modified by the function (e.g., "integer &result, ...)"). Arguments without the '&' character cannot be modified by the function. In a script, modifiable arguments don't have to be preceded by a '&'. It is an RTE if a constant is passed to a modifiable function argument (regardless of whether the function actually writes to the argument).

In the function prototype, the '[' and ']' characters surround arguments that are optional. In PolicyScript code, the optional argument may only be included if all optional arguments to the left of it are included. The function may place restrictions on when an optional argument must, or must not, be included.

In the function prototype, if a type is listed before the name of the function, the function returns a value of that type. If no type is listed, the function returns no value.

8. Base Function Library

A standard base library of functions is available to all systems that implement this specification. This library is registered with the name "pmBaseFunctionLibrary". Although the specification of this library is modularized into 4 separate sections, conformance to the library requires implementation of all functions in all sections.

The sections are:

- SNMP library functions
- Policy library functions
- Utility functions
- Library Functions

8.1. SNMP Library Functions

Two sets of SNMP Library functions are available with different situations in mind:

- Convenience SNMP Functions

  In an effort to keep simple things simple, these functions are easy to use and code that is easy to understand. These functions will suffice for the majority of situations, where a single variable is referenced and the desired error recovery is simply (and immediately) to give up (and move to the next policy-element combination). In more complex cases, the General SNMP Functions can be used at the cost of several times the code complexity.

  The convenience SNMP functions are getVar, exists, setVar, setRowStatus, createRow, counterRate, and searchColumn.

- General SNMP Functions

  The General SNMP functions allow nearly any legal SNMP Message to be generated, including those with multiple varbinds, getNext operations, notifications, and messages with explicit addressing or security specifications.

  The general SNMP functions are writeVar, readVar, snmpSend, readError, and writeBulkParameters.

8.1.1. SNMP Operations on Non-Local Systems

From time to time, a script may have to perform an operation on a different SNMP system than that on which ’this element’ resides. Scripts may also have to specify the use of alternate security parameters. In order to do this, the following optional arguments are provided for the SNMP library functions:

```
snmp-function(...[, integer mPModel,
                    string tDomain, string tAddress,
                    integer secModel, string secName,
                    integer secLevel, string contextEngineID
])
```

For example:

```
    getVar("sysDescr.0", "", SNMPv3, "transportDomainUdpIpv4",
           "192.168.1.1:161", USM, "joe", NoAuthNoPriv);
```

The use of these arguments is denoted in function definitions by the keyword ’NonLocalArgs’. The definitions of these arguments are as follows:

’mPModel’ is the integer value of the SnmpMessageProcessingModel to use for this operation.

’tDomain’ is a string containing an ASCII dotted-decimal object identifier representing the transport domain to use for this operation.

’tAddress’ is a string containing the transport address formatted according to the ’tDomain’ argument. The ASCII formats for various values of ’tDomain’ are defined by the DISPLAY-HINT for a TEXTUAL-CONVENTION that represents an address of that type. The DISPLAY-HINTs used are:

tDomain	Source of DISPLAY-HINT [5] [11]
transportDomainUdpIpv4	TransportAddressIPv4
transportDomainUdpIpv6	TransportAddressIPv6
transportDomainUdpDns	TransportAddressDns
snmpCLNSDomain	snmpOSIAddress
snmpCONSDomain	snmpOSIAddress
snmpDDPDomain	snmpNBPAddress
snmpIPXDomain	snmpIPXAddress
rfc1157Domain	snmpUDPAddress
Other	Use DISPLAY-HINT "1x:"

'secModel' is the integer value of the SnmpSecurityModel to use for this operation.

'secName' is a string value representing the SnmpSecurityName to use for this operation.

'secLevel' is the integer value of the SnmpSecurityLevel to use for this operation.

An SNMP operation will be sent to the target system by using security parameters retrieved from a local configuration datastore based on 'secModel', 'secName', and 'secLevel'. It is the responsibility of the agent to ensure that sensitive information in the local configuration datastore is used on behalf of the correct principals, as identified by the security credentials of the last entity to modify the pmPolicyAdminStatus for a policy.

To illustrate how this must be configured, consider an example in which 'joe' installs a policy on 'PMAgent' that will periodically configure objects on 'TargetAgent' with the credentials of 'Operator'. The following conditions must be true for this policy to execute with the proper privileges:

- 'Operator's security credentials for TargetAgent must be installed in PMAgent's local configuration datastore (e.g., usmUserTable [6]) indexed by TargetAgent's engineID and 'Operator'.
- VACM [9] must be configured on PMAgent so that 'joe' has access to the above entry in the appropriate MIB for the local configuration datastore (e.g., usmUserTable).
- 'joe' must be the last user to modify the pmPolicyAdminStatus object for the policy.

See the Security Considerations section for more information.

For convenience, constants for 'mPModel', 'secModel', and 'secLevel' are defined in the "Constants" section below.

'contextEngineID' is a string representing the contextEngineID of the SNMP entity targeted by this operation. It is encoded as a pair of hex digits (upper- and lowercase are valid) for each octet of the contextEngineID. If 'tDomain' and 'tAddress' are provided but 'contextEngineID' is not, then the operation will be directed to the SNMP entity reachable at 'tDomain' and 'tAddress'.

In order for PolicyScript code to use any of these arguments, all optional arguments to the left must be included. 'mPModel', 'tDomain', 'tAddress', 'secModel', 'secName', and 'secLevel' must be used as a group; if one is specified, they must all be. 'contextEngineID' may only be specified if all others are specified.

Note that a function that uses NonLocalArgs must provide a parameter for the contextName that will be required when the NonLocalArgs are present. Many functions will have the following logic:

ContextName Supplied	NonLocalArgs Supplied	
No	No	Addressed to default context on local system.
Yes	No	Addressed to named context on local system.
Yes	Yes	Addressed to named context on potentially remote system.
No	Yes	Not allowed.

8.1.2. Form of SNMP Values

Many of the library functions have input or output parameters that may be one of the many SMI data types. The actual type is not encoded in the value but is specified elsewhere, possibly by nature of the situation in which it is used. The exact usage for input and output is as follows:

Any Integer value
(INTEGER, Integer32, Counter32, Counter64, Gauge32, Unsigned32, TimeTicks, Counter64):

On input:
An Integer or a String that can be successfully coerced to an Integer with the ToInteger() operator. It is an RTE if a string is passed that cannot be converted by ToInteger() into an integer.

A string of the form

```
  enum_decimal: [ letter | digit | '-' ]* '(' decimal_constant
')'
```

will also be accepted. In this case the sequence of characters before the parentheses and the parentheses themselves are completely ignored, and the decimal_constant inside the parentheses is converted. Thus, "frame-relay(32)" translates to the integer 32.

On output:

An Integer containing the returned value.

Octet String

On input:

Either a String or an Integer. If an Integer, it will be coerced to a String with the ToString() function. This string will be used as an unencoded representation of the octet string value.

On output:

A String containing the unencoded value of the octet string.

Object Identifier

On input and on output:

A String containing a decimal ASCII encoded object identifier of the following form:

```
oid:       subid [ '.' subid ]* [ '.' ]
subid:     '0' | decimal_constant
```

It is an RTE if an Object Identifier argument is not in the form above. Note that a trailing '.' is acceptable and will simply be ignored. (Note, however, that a trailing dot could cause a strncmp() comparison of two otherwise-identical OIDs to fail; instead, use oidncmp().)

Note that ASCII descriptors (e.g., "ifIndex") are never used in these encodings "over the wire". They are never returned from library functions; nor are they ever accepted by them. NMS user interfaces are encouraged to allow humans to view object identifiers with ASCII descriptors, but they must translate those descriptors to dotted-decimal format before sending them in MIB objects to policy agents.

Null

On input:

The input is ignored.

On output:

A zero length string.

8.1.3. Convenience SNMP Functions

8.1.3.1. getVar()

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

`string getVar(string oid [, string contextName, NonLocalArgs])`

'oid' is a string containing an ASCII dotted-decimal representation of an object identifier (e.g., "1.3.6.1.2.1.1.1.0").

The optional 'contextName' argument contains the SNMP context on which to operate. If 'contextName' is not present, the contextName of 'this element' will be used. If 'contextName' is the zero-length string, the default context is used.

The optional 'NonLocalArgs' provide addressing and security information to perform an SNMP operation on a system different from that of 'this element'.

It is an RTE if the queried object identifier value does not exist.

This function returns a string containing the returned value, encoded according to the returned type. Note that no actual SNMP PDU has to be generated and parsed when the policy MIB agent resides on the same system as the managed elements.

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").

8.1.3.2. exists()

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

`integer exists(string oid [, string contextName, NonLocalArgs])`

'oid' is a string containing an ASCII dotted-decimal representation of an object identifier (e.g., "1.3.6.1.2.1.1.1.0").

The optional ’contextName’ argument contains the SNMP context on which to operate. If ’contextName’ is not present, the contextName of ’this element’ will be used. If ’contextName’ is the zero-length string, the default context is used.

The optional ’NonLocalArgs’ provide addressing and security information to perform an SNMP operation on a system different from that of ’this element’.

This function returns the value 1 if the SNMP instance exists and 0 if it doesn’t exist. Note that no actual SNMP PDU has to be generated and parsed when the policy MIB agent resides on the same system as the managed elements.

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").

8.1.3.3. setVar()

The setVar() function is used to set a MIB object instance to a certain value. The setVar() function is only valid in policyActions.

```
setVar(string oid, var value, integer type
       [, string contextName, NonLocalArgs] )
```

’oid’ is a string containing an ASCII dotted-decimal representation of an object identifier (e.g., "1.3.6.1.2.1.1.1.0").

’value’ is a string encoded in the format appropriate to the ’type’ parameter. The agent will set the variable specified by ’oid’ to the value specified by ’value’.

’type’ will be the type of the ’value’ parameter and will be set to one of the values for DataType Constants.

The optional ’contextName’ argument contains the SNMP context on which to operate.   If ’contextName’ is not present, the contextName of ’this element’ will be used. If ’contextName’ is the zero length string, the default context is used.

The optional ’NonLocalArgs’ provide addressing and security information to perform an SNMP operation on a system different from that of ’this element’. Note that no actual SNMP PDU has to be generated and parsed when the policy MIB agent resides on the same system as the managed elements.

It is an RTE if the set encounters any error.

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").

8.1.3.4. searchColumn()

```
integer searchColumn(string columnoid, string &oid,
                     string pattern, integer mode
                     [, string contextName, NonLocalArgs])
```

searchColumn performs an SNMP walk on a portion of the MIB searching for objects with values equal to the ’pattern’ parameter.

’columnoid’ constrains the search to those variables that share the same OID prefix (i.e., those that are beneath it in the OID tree).

A getnext request will be sent requesting the object identifier ’oid’. If ’oid’ is an empty string, the value of ’columnoid’ will be sent.

The value returned in each response packet will be transformed to a string representation of the value of the returned variable. The string representation of the value will be formed by putting the value in the form dictated by the "Form of SNMP Values" rules, and then by performing the ToString() function on this value, forming ’SearchString’.

The ’mode’ value controls what type of match to perform on this ’SearchString’ value. There are 6 possibilities for mode:

Mode	Search Action
ExactMatch	Case sensitive exact match of ’pattern’ and ’SearchString’.
ExactCaseMatch	Case insensitive exact match of ’pattern’ and ’SearchString’.
SubstringMatch	Case sensitive substring match, finding ’pattern’ in ’SearchString’.
SubstringCaseMatch	Case insensitive substring match, finding ’pattern’ in ’SearchString’.
RegexpMatch	Case sensitive regular expression match, searching ’SearchString’ for the regular expression given in ’pattern’.

RegexpCaseMatch	Case insensitive regular expression match, searching 'SearchString' for the regular expression given in 'pattern'.

Constants for the values of 'mode' are defined in the 'Constants' section below.

searchColumn uses the POSIX extended regular expressions defined in POSIX 1003.2.

The optional 'contextName' argument contains the SNMP context on which to operate. If 'contextName' is not present, the contextName of 'this element' will be used. If 'contextName' is the zero-length string, the default context is used.

The optional 'NonLocalArgs' provide addressing and security information to perform SNMP operations on a system different from that of 'this element'.

If a match is found, 'oid' is set to the OID of the matched value, and 1 is returned. If the search traverses beyond columnoid or returns an error without finding a match, zero is returned, and 'oid' isn't modified.

To find the first match, the caller should set 'oid' to the empty string. To find additional matches, subsequent calls to searchColumn should have 'oid' set to the OID of the last match, an operation that searchColumn performs automatically.

For example:

```
    To find an ethernet interface
    oid = "";
    searchColumn("ifType", oid, "6", 0);
```

This sends a getnext request for ifType and continues to walk the tree until a value matching 6 is found or a variable returns that is not in the 'ifType' subtree.

To find the next ethernet interface, assuming that interface 3 was discovered to be the first:

```
    oid = "ifType.3";
    searchColumn("ifType", oid, "6", 0);
```

In a loop to determine all the ethernet interfaces, this looks as follows:

```
oid = "";
while(searchColumn("ifType", oid, "6", 0)){
  /* Do something with oid */
}
```

Note that in the preceding examples, "ifType" is used as a notational convenience, and the actual code downloaded to the policy MIB agent must use the string "1.3.6.1.2.1.2.2.1.3" as there may be no MIB compiler (or MIB file) available on the policy MIB agent.

Note that if the value of ’columnoid’ is too short and thus references too much of the object identifier tree (e.g., "1.3.6"), ’columnoid’ could end up searching a huge number of variables (if the value was "1.3.6", it would search ALL variables on the agent). It is the responsibility of the caller to make sure that ’columnoid’ is set appropriately.

8.1.3.5. setRowStatus()

```
integer setRowStatus(string oid, integer maxTries
                     [, integer freeOnException , integer seed
                      , string contextName, NonLocalArgs])
```

setRowStatus is used to automate the process of finding an unused row in a read-create table that uses RowStatus whose index contains an arbitrary integer component for uniqueness.

’oid’ is a string containing an ASCII dotted-decimal representation of an object identifier, with one of the subids replaced with a ’*’ character (e.g., "1.3.6.1.3.1.99.1.2.1.9.*"). ’oid’ must reference an ’instance’ of the RowStatus object, and the ’*’ must replace any integer index item that may be set to some random value.

setRowStatus will come up with a number for the selected index item and will attempt to create the instance with the createAndWait state. If the attempt fails, it will retry with a different random index value. It will attempt this no more than ’maxTries’ times.

If the optional ’freeOnException’ argument is present and equal to 1, the agent will free this row by setting RowStatus to ’destroy’ if, later in the same script invocation, this script dies with a run-time exception or by a call to fail(). Note that this does not apply to exceptions experienced in subsequent invocations of the script.

If the optional 'seed' argument is present, the initial index will be set to 'seed'. Otherwise it will be random. 'seed' may not be present if the 'freeOnException' argument is not present.

The optional 'contextName' argument contains the SNMP context on which to operate. If 'contextName' is not present, the contextName of 'this element' will be used. If 'contextName' is the zero-length string, the default context is used.

The optional 'NonLocalArgs' provide addressing and security information to perform an SNMP operation on a system different from that of 'this element'.

setRowStatus returns the successful integer value for the index. If it is unsuccessful after 'maxTries', or if zero or more than one '*' is in OID, -1 will be returned.

The createRow function (below) can also be used when adding rows to tables. Although createRow has more functionality, setRowStatus may be preferable in certain situations (for example, to have the opportunity to inspect default values created by the agent).

8.1.3.6. createRow()

```
integer createRow(integer reqPDU, integer reqNumVarbinds,
                  integer statusColumn, integer maxTries,
                  integer indexRange,
                  integer &respPDU, integer &respNumVarbinds,
                  integer &index
                  [, integer freeOnException, string contextName,
                  NonLocalArgs])
```

createRow is used to automate the process of creating a row in a read-create table whose index contains an arbitrary integer component for uniqueness. In particular, it encapsulates the algorithm behind either the createAndWait or createAndGo mechanism and the algorithm for finding an unused row in the table. createRow is not useful for creating rows in tables whose indexes don't contain an arbitrary integer component.

createRow will perform the operation by sending 'reqPDU' and returning the results in 'respPDU'. Both 'reqPDU' and 'respPDU' must previously have been allocated with newPDU. 'reqPDU' and 'respPDU' may both contain the same PDU handle, in which case the 'reqPDU' is sent and then replaced with the contents of the received PDU.

'reqNumVarbinds' is an integer greater than zero that specifies which varbinds in the PDU will be used in this operation. The first 'reqNumVarbinds' in the PDU are used. Each such varbind must be of a special form in which the object name must have one of its subids replaced with a '*' character (e.g., "1.3.6.1.3.1.99.1.2.1.9.*"). The subid selected to be replaced will be an integer index item that may be set to some random value. The same subid should be selected in each varbind in the PDU.

'respNumVarbinds' will be modified to contain the number of varbinds received in the last response PDU.

'statusColumn' identifies which varbind in 'pdu' should be treated as the RowStatus column, where 0 identifies the 1st varbind.

createRow will come up with a random integer index value and will substitute that value in place of the '*' subid in each varbind. It will then set the value of the RowStatus column to select the 'createAndGo' mechanism and execute the set. If the attempt fails due to the unavailability of the 'createAndGo' mechanism, it will retry with the 'createAndWait' mechanism selected. If the attempt fails because the chosen index value is already in use, the operation will be retried with a different random index value. It will continue to retry different index values until it succeeds, until it has made 'maxTries' attempts, or until it encounters an error. The value of 'maxTries' should be chosen to be high enough to minimize the chance that as the table fills up an attempt to create a new entry will 'collide' too often and fail.

All random index values must be between 1 and 'indexRange', inclusive. This is so that values are not attempted for an index that fall outside of that index's restricted range (e.g., 1..65535).

If the optional 'freeOnException' argument is present and equal to 1, the agent will free this row by setting RowStatus to 'destroy' if, later in the same script invocation, this script dies with a run-time exception or by a call to fail(). Note that this does not apply to exceptions experienced in subsequent invocations of the script.

The optional 'contextName' argument contains the SNMP context on which to operate. If 'contextName' is not present, the contextName of 'this element' will be used. If 'contextName' is the zero-length string, the default context is used.

The optional 'NonLocalArgs' provide addressing and security information to perform an SNMP operation on a system different from that of 'this element'.

Note that no actual SNMP PDU has to be generated and parsed when the policy MIB agent resides on the same system as the managed elements. If no PDU is generated, the agent must correctly simulate the behavior of the SNMP Response PDU, particularly in case of an error.

This function returns zero unless an error occurs, in which case it returns the proper SNMP Error Constant. If an error occurred, respPDU will contain the last response PDU as received from the agent unless no response PDU was received, in which case respNumVarbinds will be 0. In any event, readError may be called on the PDU to determine error information for the transaction.

The 'index' parameter returns the chosen index. If successful, 'index' will be set to the successful integer index. If no SNMP error occurs but the operation does not succeed due to the following reasons, 'index' will be set to -1:

1) Unsuccessful after 'maxTries'.
2) An object name had no '*' in it.
3) An object name had more than one '*' in it.

For example, createRow() might be used as follows:

```
var index, pdu = newPDU(), nVars = 0;

writeVar(pdu, nVars++, "hlHostControlDataSource.*",
         "ifIndex." + ev(0), Oid);
writeVar(pdu, nVars++, "hlHostControlNlMaxDesiredEntries.*",
         1000, Integer);
writeVar(pdu, nVars++, "hlHostControlAlMaxDesiredEntries.*",
         1000, Integer);
writeVar(pdu, nVars++, "hlHostControlOwner.*", "policy",
         String);
writeVar(pdu, nVars++, "hlHostControlStatus.*", "active(1)",
         Integer);
if (createRow(pdu, nVars, 4, 20, 65535,
              pdu, nVars, index) != 0
```

```
            || index == -1)
            return;
        // index now contains index of new row
```

8.1.3.7. counterRate()

When a policy wishes to make a decision based on the rate of a counter, it faces a couple of problems:

1. It may have to run every X minutes but have to make decisions on rates calculated over at least Y minutes, where Y > X. This would require the complexity of managing a queue of old counter values.

2. The policy script has no control over exactly when it will run.

The counterRate() function is designed to surmount these problems easily.

```
integer counterRate(string oid, integer minInterval
                    [, integer 64bit,
                    string discOid, integer discMethod,
                    string contextName, NonLocalArgs])
```

'counterRate' retrieves the variable specified by oid once per invocation. It keeps track of timestamped values retrieved on previous invocations by this execution context so that it can calculate a rate over a period longer than that since the last invocation.

'oid' is the object identifier of the counter value that will be retrieved. The most recent previously saved value of the same object identifier that is at least 'minInterval' seconds old will be subtracted from the newly retrieved value, yielding a delta. If 'minInterval' is zero, this delta will be returned. Otherwise, this delta will be divided by the number of seconds elapsed between the two retrievals, and the integer-valued result will be returned (rounding down when necessary).

If there was no previously saved retrieval older than 'minInterval' seconds, then -1 will be returned. It is an RTE if the query returns noSuchName, noSuchInstance, or noSuchObject or an object that is not of type Counter32 or Counter64.

The delta calculation will allow for 32-bit counter semantics if it encounters rollover between the two retrievals, unless the optional argument ’64bit’ is present and equal to 1, in which case it will allow for 64-bit counter semantics.

’discOid’ and ’discMethod’ may only be present together. ’discOid’ contains an object identifier of a discontinuity indicator value that will be retrieved simultaneously with each counter value:

1. If ’discMethod’ is equal to 1 and the discontinuity indicator is less than the last one retrieved, then a discontinuity is indicated.
2. If ’discMethod’ is equal to 2 and the discontinuity indicated is different from the last one retrieved, then a discontinuity is indicated.

If this value indicates a discontinuity, this counter value (and its timestamp) will be stored, but all previously stored counter values will be invalidated and -1 will be returned.

The implementation will have to store a number of timestamped counter values. The implementation must keep all values that are newer than minInterval seconds, plus the newest value that is older than minInterval seconds. Other than this one value that is older than minInterval seconds, the implementation should discard any older values.

For example:

```
  Policy that executes every 60 seconds:
      rate = counterRate("ifInOctets.$*", 300);
      if (rate > 1000000)
          ...
```

Another example, with a discontinuity indicator:

```
  Policy that executes every 60 seconds:
      rate = counterRate("ifInOctets.$*", 300, 0,
                         "sysUpTime.0", 1);
      if (rate > 1000000)
          ...
```

Another example, with zero minInterval:

```
  Policy that executes every 60 seconds:
      delta = counterRate("ifInErrors.$*", 0);
      if (delta > 100)
          ...
```

The optional 'contextName' argument contains the SNMP context on which to operate. If 'contextName' is not present, the contextName of 'this element' will be used. If 'contextName' is the zero-length string, the default context is used.

8.1.4. General SNMP Functions

It is desirable that a general SNMP interface have the ability to perform SNMP operations on multiple variables at once and that it allow multiple varbind lists to exist at once. The newPdu, readVar, and writeVar functions exist to provide these facilities in a language without pointers, arrays, and memory allocators.

newPDU is called to allocate a PDU and return an integer handle to it. As PDUs are automatically freed when the script exits and can be reused during execution, there is no freePDU().

readVar and writeVar access a variable length varbind list for a PDU. The PDU handle and the index of the variable within that PDU are specified in every readVar and writeVar operation. Once a PDU has been fully specified by one or more calls to writeVar, it is passed to snmpSend (by referencing the PDU handle) and the number of varbinds to be included in the operation. When a response is returned, the contents of the response are returned in another PDU and may be read by one or more calls to readVar. Error information may be read from the PDU with the readError function. Because GetBulk PDUs send additional information in the SNMP header, the writeBulkParameters function is provided to configure these parameters.

Varbinds in this data store are created automatically whenever they are written by any writeVar or snmpSend operation.

For example:

```
  var pdu = newPDU();
  var nVars = 0, oid, type, value;

  writeVar(pdu, nVars++, "sysDescr.0", "", Null);
  writeVar(pdu, nVars++, "sysOID.0", "", Null);
  writeVar(pdu, nVars++, "ifNumber.0", "", Null);
  if (snmpSend(pdu, nVars, Get, pdu, nVars))
      return;
  readVar(pdu, 0, oid, value, type);
  readVar(pdu, 1, oid, value, type);
  readVar(pdu, 2, oid, value, type);
  ...
```

or,

```
  var pdu = newPDU();
  var nVars = 0, oid1, oid2;

  writeVar(pdu, nVars++, "ifIndex", "", Null);
  writeVar(pdu, nVars++, "ifType", "", Null);
  while(!done){
    if (snmpSend(pdu, nVars, Getnext, pdu, nVars))
        continue;
    readVar(pdu, 0, oid1, value, type);
    readVar(pdu, 1, oid2, value, type);
    /* leave OIDs alone, now PDU #0 is set up for next step
       in table walk. */
    if (oidncmp(oid1, "ifIndex", oidlen("ifIndex")))
      done = 0;
    ...
  }
```

Note that in the preceding examples, descriptors such as ifType and sysDescr are used in object identifiers solely as a notational convenience. The actual code downloaded to the policy MIB agent must use a dotted decimal notation only, as there may be no MIB compiler (or MIB file) available on the policy MIB agent.

To conform to this specification, implementations must allow each policy script invocation to allocate at least 5 PDUs with at least 64 varbinds per list. It is suggested that implementations limit the total number of PDUs per invocation to protect other script invocations from a malfunctioning script (e.g., a script that calls newPDU() in a loop).

8.1.4.1. newPDU()

```
integer newPDU()
```

newPDU will allocate a new PDU and return a handle to the PDU. If no PDU could be allocated, -1 will be returned. The PDU’s initial values of nonRepeaters and maxRepetitions will be zero.

8.1.4.2. writeVar()

```
writeVar(integer pdu, integer varBindIndex,
         string oid, var value, integer type)
```

writeVar will store ’oid’, ’value’, and ’type’ in the specified varbind.

’pdu’ is the handle to a PDU allocated by newPDU().

'varBindIndex' is a non-negative integer that identifies the varbind within the specified PDU modified by this call. The first varbind is number 0.

'oid' is a string containing an ASCII dotted-decimal representation of an object identifier (e.g., "1.3.6.1.2.1.1.1.0").

'value' is the value to be stored, of a type appropriate to the 'type' parameter.

'type' will be the type of the value parameter and will be set to one of the values for DataType Constants.

It is an RTE if any of the parameters don't conform to the rules above.

8.1.4.3. readVar()

```
readVar(integer pdu, integer varBindIndex, string &oid,
        var &value, integer &type)
```

readVar will retrieve the oid, the value, and its type from the specified varbind.

'pdu' is the handle to a PDU allocated by newPDU().

'varBindIndex' is a non-negative integer that identifies the varbind within the specified PDU read by this call. The first varbind is number 0.

The object identifier value of the referenced varbind will be copied into the 'oid' parameter, formatted in an ASCII dotted-decimal representation (e.g., "1.3.6.1.2.1.1.1.0").

'value' is the value retrieved, of a type appropriate to the 'type' parameter.

'type' is the type of the value parameter and will be set to one of the values for DataType Constants.

It is an RTE if 'pdu' doesn't reference a valid PDU or 'varBindIndex' doesn't reference a valid varbind.

8.1.4.4. snmpSend()

```
      integer snmpSend(integer reqPDU, integer reqNumVarbinds,
                       integer opcode,
                       integer &respPDU, integer &respNumVarbinds,
                       [, string contextName , NonLocalArgs] )
```

snmpSend will perform an SNMP operation by sending ’reqPDU’ and returning the results in ’respPDU’. Both ’reqPDU’ and ’respPDU’ must previously have been allocated with newPDU. ’reqPDU’ and ’respPDU’ may both contain the same PDU handle, in which case the ’reqPDU’ is sent and then replaced with the contents of the received PDU. If the opcode specifies a Trap or V2trap, ’respPDU’ will not be modified.

’reqNumVarbinds’ is an integer greater than zero that specifies which varbinds in the PDU will be used in this operation. The first ’reqNumVarbinds’ in the PDU are used. ’respNumVarbinds’ will be modified to contain the number of varbinds received in the response PDU, which, in the case of GetBulk or an error, may be substantially different from reqNumVarbinds.

’opcode’ is the type of SNMP operation to perform and must be one of the values for SNMP Operation Constants listed in the ’Constants’ section below.

The optional ’contextName’ argument contains the SNMP context on which to operate. If ’contextName’ is not present, the contextName of ’this element’ will be used. If ’contextName’ is the zero-length string, the default context is used.

Note that no actual SNMP PDU has to be generated and parsed when the policy MIB agent resides on the same system as the managed elements. If no PDU is generated, the agent must correctly simulate the behavior of the SNMP Response PDU, particularly in case of an error.

This function returns zero unless an error occurs, in which case it returns the proper SNMP Error Constant. If an error occurred, respPDU will contain the response PDU as received from the agent, unless no response PDU was received, in which case respNumVarbinds will be 0. In any event, readError may be called on the PDU to determine error information for the transaction.

If an SNMP Version 1 trap is requested (the opcode is Trap(4)), then SNMP Version 2 trap parameters are supplied and converted according to the rules of RFC 3584 [8], section 3.2. The first variable binding must be sysUpTime.0, and the second must be snmpTrapOID.0, as per RFC 3416 [7], section 4.2.6. Subsequent variable bindings are copied to the SNMP Version 1 trap PDU in the usual fashion.

8.1.4.5. readError()

```
readError(integer pdu, integer numVarbinds, integer &errorStatus,
          integer &errorIndex, integer &hasException)
```

Returns the error information in a PDU.

'errorStatus' contains the error-status field from the response PDU or a local error constant if the error was generated locally. If no error was experienced or no PDU was ever copied into this PDU, this value will be 0.

'errorIndex' contains the error-index field from the response PDU. If no PDU was ever copied into this PDU, this value will be 0.

'hasException' will be 1 if any of the first 'numVarbinds' varbinds in the PDU contain an exception (Nosuchobject, Nosuchinstance, Endofmibview); otherwise it will be 0.

It is an RTE if 'pdu' does not reference a valid PDU or if 'numVarbinds' references varbinds that aren't valid.

8.1.4.6. writeBulkParameters()

```
writeBulkParameters(integer pdu, integer nonRepeaters,
                    integer maxRepetitions)
```

Modifies the parameters in a PDU in any subsequent GetBulk operation sent by the PDU. 'nonRepeaters' will be copied into the PDU's non-repeaters field, and 'maxRepetitions' into the max-repetitions field.

This function may be called before or after writeVar is called to add varbinds to the PDU, but it must be called before the PDU is sent; otherwise, it will have no effect. A new PDU is initialized with nonRepeaters set to zero and maxRepetitions set to zero. If a Bulk PDU is sent before writeBulkParameters is called, these default values will be used. If writeBulkParameters is called to modify a PDU, it is acceptable if this PDU is later sent as a type other than bulk. The writeBulkParameters call will only affect subsequent sends of Bulk PDUs. If a PDU is used to receive the contents of a response, the values of nonRepeaters and maxRepetitions are never modified.

8.1.5. Constants for SNMP Library Functions

The following constants are defined for use with all SNMP Library Functions. Policy code will be executed in an environment where the following constants are declared. (Note that the constant declarations below will not be visible in the policyCondition or policyAction code.) These constants are reserved words and cannot be used for any variable or function name.

Although these declarations are expressed here as C ’const’s, the ’const’ construct itself is not available to be used in policy code.

```
// Datatype Constants

// From RFC 2578 [2]
const integer Integer       = 2;
const integer Integer32     = 2;
const integer String        = 4;
const integer Bits          = 4;
const integer Null          = 5;
const integer Oid           = 6;
const integer IpAddress     = 64;
const integer Counter32     = 65;
const integer Gauge32       = 66;
const integer Unsigned32    = 66;
const integer TimeTicks     = 67;
const integer Opaque        = 68;
const integer Counter64     = 70;

// SNMP Exceptions from RFC 3416 [7]
const integer NoSuchObject         = 128;
const integer NoSuchInstance       = 129;
const integer EndOfMibView         = 130;

// SNMP Error Constants from RFC 3416 [7]
const integer NoError              = 0;
const integer TooBig               = 1;
const integer NoSuchName           = 2;
const integer BadValue             = 3;
const integer ReadOnly             = 4;
const integer GenErr               = 5;
const integer NoAccess             = 6;
const integer WrongType            = 7;
const integer WrongLength          = 8;
const integer WrongEncoding        = 9;
```

```
   const integer WrongValue           = 10;
   const integer NoCreation           = 11;
   const integer InconsistentValue    = 12;
   const integer ResourceUnavailable  = 13;
   const integer CommitFailed         = 14;
   const integer UndoFailed           = 15;
   const integer AuthorizationError   = 16;
   const integer NotWritable          = 17;
   const integer InconsistentName     = 18;

   // "Local" Errors
   // These are also possible choices for errorStatus returns
   // For example: unknown PDU, maxVarbinds is bigger than number
   // written with writeVar, unknown opcode, etc.
   const integer BadParameter         = 1000;

   // Request would have created a PDU larger than local limitations
   const integer TooLong              = 1001;

   // A response to the request was received but errors were encountered
   // when parsing it.
   const integer ParseError           = 1002;

   // Local system has complained of an authentication failure
   const integer AuthFailure          = 1003;

   // No valid response was received in a timely fashion
   const integer TimedOut             = 1004;

   // General local failure including lack of resources
   const integer GeneralFailure       = 1005;

   // SNMP Operation Constants from RFC 3416 [7]
   const integer Get                  = 0;
   const integer Getnext              = 1;
   const integer Set                  = 3;
   const integer Trap                 = 4;
   const integer Getbulk              = 5;
   const integer Inform               = 6;
   const integer V2trap               = 7;

   // Constants from RFC 3411 [1] for SnmpMessageProcessingModel
   const integer SNMPv1              = 0;
   const integer SNMPv2c             = 1;
   const integer SNMPv3              = 3;
```

```
// Constants from RFC 3411 [1] for SnmpSecurityModel
const integer SNMPv1              = 1;
const integer SNMPv2c             = 2;
const integer USM                 = 3;

// SnmpSecurityLevel Constants from RFC 3411 [1]
const integer NoAuthNoPriv        = 1;
const integer AuthNoPriv          = 2;
const integer AuthPriv            = 3;

// Constants for use with searchColumn
const integer ExactMatch          = 0;
const integer ExactCaseMatch      = 1;
const integer SubstringMatch      = 2;
const integer SubstringCaseMatch  = 3;
const integer RegexpMatch         = 4;
const integer RegexpCaseMatch     = 5;
```

8.2. Policy Library Functions

Policy Library Functions provide access to information specifically related to the execution of policies.

8.2.1. 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 and how it is indexed.

```
string elementName()
```

elementName returns a 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’.

8.2.2. elementAddress()

```
elementAddress(&tDomain, &tAddress)
```

elementAddress finds a domain/address pair that can be used to access ’this element’ and returns the values in ’tDomain’ and ’tAddress’.

8.2.3. elementContext()

string elementContext()

elementContext() returns a string containing the SNMP contextName of 'this element'.

8.2.4. ec()

The ec() (element count) and ev() (element value) functions provide convenient access to the components of the index for 'this element'. Typical uses will be in creating the index to other, related elements.

integer ec()

ec() returns an integer count of the number of index subidentifiers that exist in the index for 'this element'.

8.2.5. ev()

integer ev(integer n)

ev() returns the value of the nth subidentifier in the index for 'this element'. The first subidentifier is indexed at 0. It is an RTE if n specifies a subidentifier beyond the last subidentifier.

8.2.6. roleMatch()

The roleMatch() function is used to check whether an element has been assigned a particular role.

```
integer roleMatch(string roleString [, string element,
                  string contextName, string contextEngineID])
```

'roleString' is a string. The optional argument 'element' contains the OID name of an element, defaulting to the current element if 'element' is not supplied. If roleString exactly matches (content and length) any role assigned to the specified element, the function returns 1. If no roles match, the function returns 0.

The optional 'contextName' argument contains the SNMP context on which to operate. If 'contextName' is not present, the contextName of 'this element' will be used. If 'contextName' is the zero-length string, the default context is used.

’contextEngineID’ contains the contextEngineID of the remote system on which ’element’ resides. It is encoded as a pair of hex digits (upper- and lowercase are valid) for each octet of the contextEngineID. If ’contextEngineID’ is not present, the contextEngineID of ’this element’ will be used. ’contextEngineID’ may only be present if the ’element’ and ’context’ arguments are present.

8.2.7. Scratchpad Functions

Every maxLatency time period, every policy runs once for each element. When the setScratchpad function executes, it stores a value named by a string that can be retrieved with getScratchpad() even after this policy execution code exits. This allows sharing of data between a condition and an action, two conditions executing on different elements, or even different policies altogether.

The value of ’scope’ controls which policy/element combinations can retrieve this ’varName’/’value’ pair. The following are options for ’scope’:

Global

The ’varName’/’value’ combination will be available in the condition or action of any policy while it is executing on any element. Note that any information placed here will be visible to all other scripts on this system regardless of their authority. Sensitive information should not be placed in global scratchpad variables.

Policy

The ’varName’/’value’ combination will be available in any future execution of the condition or action of the current policy (regardless of what element the policy is executing on). If a policy is ever deleted, or if its condition or action code is modified, all values in its ’Policy’ scope will be deleted.

PolicyElement

The ’varName’/’value’ combination will be available in future executions of the condition or action of the current policy, but only when the policy is executing on the current element. If a policy is ever deleted, or if its condition or action code is modified, all values in its ’PolicyElement’ scope will be deleted. The agent may also periodically delete values in a ’PolicyElement’ scope if the corresponding element does not exist (in other words, if an element disappears for a period and reappears, values in its ’PolicyElement’ scope may or may not be deleted).

setScratchpad’s ’storageType’ argument allows the script to control the lifetime of a variable stored in the scratchpad. If the storageType is equal to the constant ’volatile’, then this variable must be deleted on a reboot. If it is equal to ’nonVolatile’, then this variable should be stored in non-volatile storage, where it will be available after a reboot. If the ’storageType’ argument is not present, the variable will be volatile and will be erased on reboot.

If the optional ’freeOnException’ argument is present and equal to 1, the agent will free this variable if, later in the same script invocation, this script dies with a run-time exception or by a call to fail(). (Note that this does not apply to exceptions experienced in subsequent invocations of the script.)

Note that there may be implementation-specific limits on the number of scratchpad variables that can be allocated. The limit of unique scratchpad variables may be different for each scope or storageType. It is suggested that implementations limit the total number of scratchpad variables per script to protect other scripts from a malfunctioning script. In addition, compliant implementations must support at least 50 Global variables, 5 Policy variables per policy, and 5 PolicyElement variables per policy-element pair.

Scratchpad Usage Examples

Policy	Element	Action
A	ifIndex.1	setScratchpad(Global, "foo", "55")
A	ifIndex.1	getScratchpad(Global, "foo", val) --> 55
A	ifIndex.2	getScratchpad(Global, "foo", val) --> 55
B	ifIndex.2	getScratchpad(Global, "foo", val) --> 55
B	ifIndex.2	setScratchpad(Global, "foo", "16")
A	ifIndex.1	getScratchpad(Global, "foo", val) --> 16

Policy	Element	Action
A	ifIndex.1	setScratchpad(Policy, "bar", "75")
A	ifIndex.1	getScratchpad(Policy, "bar", val) --> 75
A	ifIndex.2	getScratchpad(Policy, "bar", val) --> 75
B	ifIndex.1	getScratchpad(Policy, "bar", val) not found
B	ifIndex.1	setScratchpad(Policy, "bar", "20")
A	ifIndex.2	getScratchpad(Policy, "bar", val) --> 75
B	ifIndex.2	getScratchpad(Policy, "bar", val) --> 20

Policy	Element	Action
A	ifIndex.1	setScratchpad(PolicyElement, "baz", "43")
A	ifIndex.1	getScratchpad(PolicyElement, "baz", val) --> 43
A	ifIndex.2	getScratchpad(PolicyElement, "baz", val) not found
B	ifIndex.1	getScratchpad(PolicyElement, "baz", val) not found
A	ifIndex.2	setScratchpad(PolicyElement, "baz", "54")

```
B       ifIndex.1  setScratchpad(PolicyElement, "baz", "65")
A       ifIndex.1  getScratchpad(PolicyElement, "baz", val) --> 43
A       ifIndex.2  getScratchpad(PolicyElement, "baz", val) --> 54
B       ifIndex.1  getScratchpad(PolicyElement, "baz", val) --> 65
```

```
Policy  Element    Action
A       ifIndex.1  setScratchpad(PolicyElement, "foo", "11")
A       ifIndex.1  setScratchpad(Global,        "foo", "22")
A       ifIndex.1  getScratchpad(PolicyElement, "foo", val) --> 11
A       ifIndex.1  getScratchpad(Global,        "foo", val) --> 22
```

Constants

The following constants are defined for use with the scratchpad functions. Policy code will be executed in an environment where the following constants are declared. (Note that these constant declarations will not be visible in the policyCondition or policyAction MIB objects.)

Although these declarations are expressed here as C ’const’s, the ’const’ construct itself is not available to be used inside of policy code.

```
// Scratchpad Constants

// Values of scope
const integer Global           = 0;
const integer Policy           = 1;
const integer PolicyElement    = 2;

// Values of storageType
const integer Volatile         = 0;
const integer NonVolatile      = 1;
```

8.2.8. setScratchpad()

```
setScratchpad(integer scope, string varName [, string value,
              integer storageType, integer freeOnException ])
```

The setScratchpad function stores a value that can be retrieved even after this policy execution code exits.

The value of ’scope’ controls which policy/element combinations can retrieve this ’varName’/’value’ pair. The options for ’scope’ are Global, Policy, and PolicyElement.

’varName’ is a string used to identify the value. Subsequent retrievals of the same ’varName’ in the proper scope will return the value stored. Note that the namespace for ’varName’ is distinct for each scope. ’varName’ is case sensitive.

’value’ is a string containing the value to be stored. ToString(value) is called on ’value’ to convert it to a string before storage.

If the ’value’ argument is missing, the ’varName’ in scope ’scope’ will be deleted if it exists.

If the optional ’storageType’ argument is present and is equal to the constant ’Volatile’, then this variable must be deleted on a reboot. If it is equal to ’NonVolatile’, then this variable should be stored in non-volatile storage, where it will be available after a reboot. If the ’storageType’ argument is not present, the variable will be volatile and will be erased on reboot. ’storageType’ may not be present if the ’value’ argument is not present. If the variable already existed, its previous storageType is updated according to the current ’storageType’ argument.

If the optional ’freeOnException’ argument is present and equal to 1, the agent will free this variable if, later in the same script invocation, this script dies with a run-time exception or by a call to fail(). (Note that this does not apply to exceptions experienced in subsequent invocations of the script.)

8.2.9. getScratchpad()

```
integer getScratchpad(integer scope, string varName,
                      string &value)
```

The getScratchpad function allows the retrieval of values that were stored previously in this execution context or in other execution contexts. The value of ’scope’ controls which execution contexts can pass a value to this execution context. The options for ’scope’ are Global, Policy, and PolicyElement.

’varName’ is a string used to identify the value. Subsequent retrievals of the same ’varName’ in the proper scope will return the value stored. Note that the namespace for varName is distinct for each scope. As a result, getScratchpad cannot force access to a variable in an inaccessible scope; it can only retrieve variables by referencing the proper scope in which they were set. ’varName’ is case sensitive.

On successful return, 'value' will be set to the value that was previously stored; otherwise, 'value' will not be modified.

This function returns 1 if a value was previously stored and 0 otherwise.

8.2.10. signalError()

The signalError() function is used by the script to indicate to a management station that it is experiencing abnormal behavior. signalError() turns on the conditionUserSignal(3) or actionUserSignal(5) bit in the associated pmTrackingPEInfo object (subsequent calls to signalError() have no additional effect). This bit is initially cleared at the beginning of each execution. If, upon a subsequent execution, the script finishes without calling signalError, the bit will be cleared.

```
signalError()
```

The signalException function takes no arguments and returns no value.

8.2.11. defer()

Precedence groups enforce the rule that for each element, of the ready policies that match the condition, only the one with the highest precedence value will be active. Unfortunately, once the winning policy has been selected and the action begins running, situations can occur in which the policy script determines that it cannot complete its task. In many such cases, it is desirable that the next runner-up ready policy be executed. In the previous example, it would be desirable that at least bronze behavior be configured if gold is appropriate but gold isn't possible.

When a policy defers, it exits, and the ready, condition-matching policy with the next-highest precedence is immediately run. Because this might also defer, the execution environment must remember where it is in the precedence chain so that it can continue going down the chain until an action completes without deferring, or until no policies are left in the precedence group. Once a policy finishes successfully, the next iteration will begin at the top of the precedence chain.

There are two ways to defer. A script can exit by calling fail() and specify that it should defer immediately. Alternately, a script can instruct the execution environment to defer automatically in the event of a run-time exception.

defer(integer deferOnRTE)

The defer function changes the run-time exception behavior of a script. By default, a script will not defer when it encounters an RTE. If defer(1) is called, the exit behavior is changed so that the script will defer when it is terminated due to an RTE. If defer(0) is called, the script is reset to its default behavior and will not defer.

Note that calling defer doesn’t cause the script to exit. Defer only changes the default behavior if an RTE occurs later in this invocation.

8.2.12. fail()

fail(integer defer, integer free [, string message])

The fail function causes the script to optionally perform certain functions and then exit.

If ’defer’ is 1, this script will defer to the next lower precedence ready policy in the same precedence group whose condition matches. If ’defer’ isn’t 1, it will not defer. Note that if a condition defers, it is functionally equivalent to the condition returning false.

If ’free’ is 1, certain registered resources will be freed. If, earlier in this script invocation, any rows were created by createRow with the ’freeOnException’ option, the execution environment will set the RowStatus of each row to ’destroy’ to delete the row. Further, if earlier in this script invocation any scratchpad variables were created or modified with the ’freeOnException’ option, they will be deleted.

If the optional ’message’ argument is present, it will be logged to the debugging table if pmPolicyDebugging is turned on for this policy.

This function does not return. Instead, the script will terminate.

8.2.13. getParameters()

From time to time, policy scripts may be parameterized so that they are supplied with one or more parameters (e.g., site-specific constants). These parameters may be installed in the pmPolicyParameters object and are accessible to the script via the getParameters() function. If it is necessary for multiple parameters to be passed to the script, the script can choose whatever encoding/delimiting mechanism is most appropriate so that the multiple parameters can be stored in the associated instance of pmPolicyParameters.

```
string getParameters()
```

The getParameters function takes no arguments. It returns a string containing the value of the pmPolicyParameters object for the running policy.

For example, if a policy is to apply to "slow speed interfaces" and the cutoff point for slow speed should be parameterized, the policy filter should be:

```
getVar("ifSpeed.$*") == getParameters()
```

In this example, one can store the string "128000" in the policy’s pmPolicyParameters object to cause this policy to act on all 128 Kbps interfaces.

8.3. Utility Library Functions

Utility Library Functions are provided to enable more efficient policy scripts.

8.3.1. regexp()

```
integer regexp(string pattern, string str,
               integer case [, string &match])
```

regexp searches ’str’ for matches to the regular expression given in ‘pattern‘. regexp uses the POSIX extended regular expressions defined in POSIX 1003.2.

If ‘case‘ is 0, the search will be case insensitive; otherwise, it will be case sensitive.

If a match is found, 1 is returned, otherwise 0 is returned.

If the optional argument ’match’ is provided and a match is found, ’match’ will be replaced with the text of the first substring of ’str’ that matches ’pattern’. If no match is found, it will be unchanged.

8.3.2. regexpReplace()

```
string regexpReplace(string pattern, string replacement,
                     string str, integer case)
```

regexpReplace searches ’str’ for matches to the regular expression given in ’pattern’, replacing each occurrence of matched text with ’replacement’. regexpReplace uses the POSIX extended regular expressions defined in POSIX 1003.2.

If ‘case‘ is 0, the search will be case insensitive; otherwise, it will be case sensitive.

The modified string is returned (it would be the same as the original string if no matches were found).

8.3.3. oidlen()

```
integer oidlen(string oid)
```

oidlen returns the number of subidentifiers in ’oid’. ’oid’ is a string containing an ASCII dotted-decimal representation of an object identifier (e.g., "1.3.6.1.2.1.1.1.0").

8.3.4. oidncmp()

```
integer oidncmp(string oid1, string oid2, integer n)
```

Arguments ’oid1’ and ’oid2’ are strings containing ASCII dotted-decimal representations of object identifiers (e.g., "1.3.6.1.2.1.1.1.0").

oidcmp compares not more than n subidentifiers of ’oid1’ and ’oid2’ and returns -1 if ’oid1’ is less than ’oid2’, 0 if they are equal, and 1 if ’oid1’ is greater than ’oid2’.

8.3.5. inSubtree()

```
integer inSubtree(string oid, string prefix)
```

Arguments ’oid’ and ’prefix’ are strings containing ASCII dotted-decimal representations of object identifiers (e.g., "1.3.6.1.2.1.1.1.0").

inSubtree returns 1 if every subidentifier in ’prefix’ equals the corresponding subidentifier in ’oid’, otherwise it returns 0. The is equivalent to oidncmp(oid1, prefix, oidlen(prefix)) is provided because this is an idiom and because it avoids evaluating ’prefix’ twice if it is an expression.

8.3.6. subid()

```
integer subid(string oid, integer n)
```

subid returns the value of the nth (starting at zero) subidentifier of ’oid’. ’oid’ is a string containing an ASCII dotted-decimal representation of an object identifier (e.g., "1.3.6.1.2.1.1.1.0").

If n specifies a subidentifier beyond the length of ’oid’, a value of -1 is returned.

8.3.7. subidWrite()

```
integer subidWrite(string oid, integer n, integer subid)
```

subidWrite sets the value of the nth (starting at zero) subidentifier of ’oid’ to ’subid’. ’oid’ is a string containing an ASCII dotted-decimal representation of an object identifier (e.g., "1.3.6.1.2.1.1.1.0").

If n specifies a subidentifier beyond the length of ’oid’, a value of -1 is returned. Note that appending subidentifiers can be accomplished with the string concatenation ’+’ operator. If no error occurs, zero is returned.

8.3.8. oidSplice()

```
string oidSplice(string oid1, integer offset, integer len, string
   oid2)
```

oidSplice returns an OID formed by replacing ’len’ subidentifiers in ’oid1’ with all of the subidentifiers from ’oid2’, starting at ’offset’ in ’oid1’ (the first subidentifier is at offset 0). The OID length will be extended, if necessary, if ’offset’ + ’len’ extends beyond the end of ’oid1’. If ’offset’ is larger than the length of oid1, then an RTE will occur.

The resulting OID is returned.

For example:

```
    oidSplice("1.3.6.1.2.1", 5, 1, "7")     => "1.3.6.1.2.7"
    oidSplice("1.3.6.1.2.1", 4, 2, "7.7")   => "1.3.6.1.7.7"
    oidSplice("1.3.6.1.2.1", 4, 3, "7.7.7") => "1.3.6.1.7.7.7"
```

8.3.9. parseIndex()

ParseIndex is provided to make it easy to pull index values from OIDs into variables.

```
var parseIndex(string oid, integer &index, integer type,
               integer len)
```

parseIndex pulls values from the instance identification portion of ’oid’, encoded as per Section 7.7, "Mapping of the INDEX Clause", of the SMIv2 [2].

’oid’ is the OID to be parsed.

’index’ describes which subid to begin parsing at. ’index’ will be modified to indicate the subid after the last one parsed (even if this points past the last subid). The first subid is index 0. If any error occurs, ’index’ will be set to -1 on return. If the input index is less than 0 or refers past the end of the OID, ’index’ will be set to -1 on return and the function will return 0.

If ’type’ is Integer, ’len’ will not be consulted. The return value is the integer value of the next subid.

If ’type’ is String and ’len’ is greater than zero, ’len’ subids will be parsed. For each subid parsed, the chr() value of the subid will be appended to the returned string. If any subid is greater than 255, ’index’ will be set to -1 on return, and an empty string will be returned. If there are fewer than ’len’ subids left in ’oid’, ’index’ will be set to -1 on return, but a string will be returned containing a character for each subid that was left.

If ’type’ is String and ’len’ is zero, the next subid will be parsed to find N, the length of the string. Then, that many subids will be parsed. For each subid parsed, the chr() value of the subid will be appended to the returned string. If any subid is greater than 255, ’index’ will be set to -1 on return, and an empty string will be returned. If there are fewer than N subids left in ’oid’, ’index’ will be set to -1 on return, but a string will be returned containing a character for each subid that was left.

If ’type’ is String and ’len’ is -1, subids will be parsed until the end of ’oid’. For each subid parsed, the chr() value of the subid will be appended to the returned string. If any subid is greater than 255, 'index' will be set to -1 on return, and an empty string will be returned.

If 'type' is Oid and 'len' is greater than zero, 'len' subids will be parsed. For each subid parsed, the decimal-encoded value of the subid will be appended to the returned string, with a '.' character appended between each output subid, but not after the last subid. If there are fewer than 'len' subids left in 'oid', 'index' will be set to -1 on return, but a string will be returned containing an encoding for each subid that was left.

If 'type' is Oid and 'len' is zero, the next subid will be parsed to find N, the number of subids to parse. For each subid parsed, the decimal-encoded value of the subid will be appended to the returned string, with a '.' character appended between each output subid but not after the last subid. If there are fewer than N subids left in 'oid', 'index' will be set to -1 on return, but a string will be returned containing an encoding for each subid that was left.

If 'type' is Oid and 'len' is -1, subids will be parsed until the end of 'oid'. For each subid parsed, the decimal-encoded value of the subid will be appended to the returned string, with a '.' character appended between each output subid, but not after the last subid.

For example, to decode the index component of an instance of the ipForward table:

```
oid = "ipForwardIfIndex.0.0.0.0.13.0.192.168.1.1";
index = 11;
dest   =  parseIndex(oid, index, String, 4);
proto  =  parseIndex(oid, index, Integer, 0);
policy =  parseIndex(oid, index, Integer, 0);
nextHop = parseIndex(oid, index, String, 4);
// proto and policy now contain integer values
// dest and nextHop now contain 4 byte IP addresses.  Use
// stringToDotted to get them to dotted decimal notation:
// e.g.: stringToDotted(nextHop) => "192.168.1.1"
```

8.3.10. stringToDotted()

stringToDotted() is provided to encode strings suitable for the index portion of an OID or to convert the binary encoding of an IP address to a dotted-decimal encoding.

string stringToDotted(string value)

If ’value’ is the zero-length string, the zero-length string is returned.

The decimal encoding of the first byte of ’value’ is appended to the output string. Then, for each additional byte in ’value’, a ’.’ is appended to the output string, followed by the decimal encoding of the additional byte.

8.3.11. integer()

integer integer(var input)

integer converts ’input’ into an integer by using the rules specified for ToInteger(), returning the integer-typed results.

8.3.12. string()

string string(var input)

string converts ’input’ into a string by using the rules specified for ToString(), returning the string-typed results.

8.3.13. type()

string type(var variable)

type returns the type of its argument as either the string ’String’ or the string ’Integer’.

8.3.14. chr()

string chr(integer char)

Returns a one-character string containing the character specified by the ASCII code contained in ’char’.

8.3.15. ord()

integer ord(string str)

Returns the ASCII value of the first character of ’str’. This function complements chr().

8.3.16. substr()

```
string substr(string &str, integer offset
              [, integer len, string replacement])
```

Extracts a substring out of 'str' and returns it. The first octet is at offset 0. If the offset is negative, the returned string starts that far from the end of 'str'. If 'len' is positive, the returned string contains up to 'len' octets, up to the end of the string. If 'len' is omitted, the returned string includes everything to the end of 'str'. If 'len' is negative, abs(len) octets are left off the end of the string.

If a substring is specified that is partly outside the string, the part within the string is returned. If the substring is totally outside the string, a zero-length string is produced.

If the optional 'replacement' argument is included, 'str' is modified. 'offset' and 'len' act as above to select a range of octets in 'str'. These octets are replaced with octets from 'replacement'. If the replacement string is shorter or longer than the number of octets selected, 'str' will shrink or grow, respectively. If 'replacement' is included, the 'len' argument must also be included.

Note that to replace everything from offset to the end of the string, substr() should be called as follows:

```
substr(str, offset, strlen(str) - offset, replacement)
```

8.4. General Functions

The following POSIX standard library functions are provided:

```
strncmp()
strncasecmp()
strlen()
random()
sprintf()
sscanf()
```

9. International String Library

This library is optional for systems that wish to have support for collating (sorting) and verifying equality of international strings in a manner that will be least surprising to humans. International strings are encoded in the UTF-8 transformation format described in [14]. This library is registered with the name "pmInternationalStringLibrary".

When verifying equality of international strings in the Unicode character set, it is recommended to normalize the strings with the stringprep() function before checking for equality.

When attempting to sort international strings in the Unicode character set, normalization should also be performed, but note that the result is highly context dependent and hard to implement correctly. Just ordering by Unicode Codepoint Value is in many cases not what the end user expects. See Unicode technical note 9 for more information about sorting.

9.1. stringprep()

integer stringprep(string utf8Input, string &utf8Output)

Performs the Stringprep [13] transformation for appropriate comparison of internationalized strings. The transformation is performed on ’utf8Input’; if the transformation finishes without error, the resulting string is written to utf8Output. The stringprep profile used is specified below in Section 9. If it is successful, the function returns 1.

If the stringprep transformation encounters an error, 0 is returned, and the utf8Output parameter remains unchanged.

For example, to compare UTF8 strings ’one’ and ’two’:

```
if (stringprep(one, a) && stringprep(two, b)){
    if (a == b){
       // strings are identical
    } else {
       // strings are different
    }
} else {
    // strings couldn’t be transformed for comparison
}
```

See Stringprep [13] for more information.

9.1.1. Stringprep Profile

The Stringprep specification [13] describes a framework for preparing Unicode text strings in order to increase the likelihood that string input and string comparison work in ways that make sense for typical users throughout the world. Specifications that specify stringprep (as this one does) are required to fully specify stringprep’s processing options by documenting a stringprep profile.

This profile defines the following, as required by Stringprep:

- The intended applicability of the profile: internationalized network management information.

- The character repertoire that is the input and output to stringprep: Unicode 3.2, as defined in Stringprep [13], Appendix A.1.

- The mapping tables used: Table B.1 from Stringprep [13].

- Any additional mapping tables specific to the profile: None.

- The Unicode normalization used: Form KC, as described in Stringprep [13].

- The characters that are prohibited as output: As specified in the following tables from Stringprep [13]:

  Table C.2
  Table C.3
  Table C.4
  Table C.5
  Table C.6
  Table C.7
  Table C.8
  Table C.9

- Bidirectional character handling: not performed.

- Any additional characters that are prohibited as output: None.

9.2. utf8Strlen()

```
integer utf8Strlen(string str)
```

Returns the number of UTF-8 characters in ’str’, which may be less than the number of octets in ’str’ if one or more characters are multi-byte characters.

9.3. utf8Chr()

```
string utf8Chr(integer utf8)
```

Returns a one-character string containing the character specified by the UTF-8 code contained in 'utf8'. Although it contains only 1 UTF-8 character, the resulting string may be more than 1 octet in length.

9.4. utf8Ord()

```
integer utf8Ord(string str)
```

Returns the UTF-8 code-point value of the first character of 'str'. Note that the first UTF-8 character in 'str' may be more than 1 octet in length. This function complements chr().

9.5. utf8Substr()

```
string utf8Substr(string &str, integer offset
              [, integer len, string replacement])
```

Extracts a substring out of 'str' and returns it, keeping track of UTF-8 character boundaries and using them, instead of octets, as the basis for offset and length calculations. The first character is at offset 0. If offset is negative, the returned string starts that far from the end of 'str'. If 'len' is positive, the returned string contains up to 'len' characters, up to the end of the string. If 'len' is omitted, the returned string includes everything to the end of 'str'. If 'len' is negative, abs(len) characters are left off the end of the string.

If you specify a substring that is partly outside the string, the part within the string is returned. If the substring is totally outside the string, a zero-length string is produced.

If the optional 'replacement' argument is included, 'str' is modified. 'offset' and 'len' act as above to select a range of characters in 'str'. These characters are replaced with characters from 'replacement'. If the replacement string is shorter or longer than the number of characters selected, 'str' will shrink or grow, respectively. If 'replacement' is included, the 'len' argument must also be included.

Note that to replace everything from offset to the end of the string, substr() should be called as follows:

```
substr(str, offset, strlen(str) - offset, replacement)
```

10. Schedule Table

This table is an adapted form of the policyTimePeriodCondition class defined in the Policy Core Information Model, RFC 3060 [18]. Some of the objects describing a schedule are expressed in formats defined in the iCalendar specification [15].

The policy schedule table allows control over when a valid policy will be ready, based on the date and time.

A policy’s pmPolicySchedule variable refers to a group of one or more schedules in the schedule table. At any given time, if any of these schedules are active, the policy will be ready (assuming that it is enabled and thus valid), and its conditions and actions will be executed, as appropriate. At times when none of these schedules are active, the policy will not be ready and will have no effect. A policy will always be ready if its pmPolicySchedule variable is 0. If a policy has a non-zero pmPolicySchedule that doesn’t refer to a group that includes an active schedule, then the policy will not be ready, even if this is due to a misconfiguration of the pmPolicySchedule object or the pmSchedTable.

A policy that is controlled by a schedule group immediately executes its policy condition (and conditionally the policyAction) when the schedule group becomes active, periodically re-executing these scripts as appropriate until the schedule group becomes inactive (i.e., all schedules are inactive).

An individual schedule item is active at those times that match all the variables that define the schedule: pmSchedTimePeriod, pmSchedMonth, pmSchedDay, pmSchedWeekDay, and pmSchedTimeOfDay. It is possible to specify multiple values for each schedule item. This provides a mechanism for defining complex schedules. For example, a schedule that is active the entire workday each weekday could be defined.

Months, days, and weekdays are specified by using the objects pmSchedMonth, pmSchedDay, and pmSchedWeekDay of type BITS. Setting multiple bits in these objects causes an OR operation. For example, setting the bits monday(1) and friday(5) in pmSchedWeekDay restricts the schedule to Mondays and Fridays.

The matched times for pmSchedTimePeriod, pmSchedMonth, pmSchedDay pmSchedWeekDay, and pmSchedTimeOfDay are ANDed together to determine the time periods when the schedule will be active; in other words, the schedule is only active for those times when ALL of these schedule attributes match. For example, a schedule with an overall validity range of January 1, 2000, through December 31, 2000; a month mask that selects March and April; a day-of-the-week mask that selects Fridays; and a time-of-day range of 0800 through 1600 would represent the following time periods:

```
   Friday, March  5, 2000, from 0800 through 1600
   Friday, March 12, 2000, from 0800 through 1600
   Friday, March 19, 2000, from 0800 through 1600
   Friday, March 26, 2000, from 0800 through 1600
   Friday, April  2, 2000, from 0800 through 1600
   Friday, April  9, 2000, from 0800 through 1600
   Friday, April 16, 2000, from 0800 through 1600
   Friday, April 23, 2000, from 0800 through 1600
   Friday, April 30, 2000, from 0800 through 1600
```

Wildcarding of schedule attributes of type BITS is achieved by setting all bits to one.

It is possible to define schedules that will never cause a policy to be activated. For example, one can define a schedule that should be active on February 31st.

11. Definitions

```
POLICY-BASED-MANAGEMENT-MIB DEFINITIONS ::= BEGIN
IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, NOTIFICATION-TYPE,
    Counter32, Gauge32, Unsigned32,
    mib-2                                        FROM SNMPv2-SMI
    RowStatus, RowPointer, TEXTUAL-CONVENTION,
    DateAndTime, StorageType                     FROM SNMPv2-TC
    MODULE-COMPLIANCE, OBJECT-GROUP,
    NOTIFICATION-GROUP                           FROM SNMPv2-CONF
    SnmpAdminString                              FROM SNMP-FRAMEWORK-MIB;

--  Policy-Based Management MIB

pmMib MODULE-IDENTITY
    LAST-UPDATED "200502070000Z"  -- February 7, 2005
    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 (WG Co-chair)
        JDS Consulting, Inc.
        84 Kettell Plain Road.
        Stow MA 01775
        USA
        Phone: +1-978-461-0249
        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

        David Partain (WG Co-chair)
        Postal: Ericsson AB
                P.O. Box 1248
                SE-581 12 Linkoping
                Sweden
        Tel: +46 13 28 41 44
        E-mail: David.Partain@ericsson.com

        Any questions or comments about this document can also be
        directed to the working group at snmpconf@snmp.com."
    DESCRIPTION
        "The MIB module for policy-based configuration of SNMP
        infrastructures.

        Copyright (C) The Internet Society (2005).  This version of
        this MIB module is part of RFC 4011; see the RFC itself for
        full legal notices."

    REVISION "200502070000Z"    -- February 7, 2005
    DESCRIPTION
        "The original version of this MIB, published as RFC4011."
    ::= { mib-2 124 }
```

```
PmUTF8String ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
        "An octet string containing information typically in
        human-readable form.

        To facilitate internationalization, this
        information is represented by using the ISO/IEC
        IS 10646-1 character set, encoded as an octet
        string using the UTF-8 transformation format
        described in RFC 3629.

        As 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
        0x10FFFF.  Byte sequences that do not
        correspond to the valid UTF-8 encoding of a
        code point or that are outside this range are
        prohibited.

        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.

        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.

        UTF-8 may require multiple bytes to represent a
        single character/code point; thus, the length
        of this object in octets may be different from
        the number of characters encoded.  Similarly,
        size constraints refer to the number of encoded
        octets, not the number of characters represented
        by an encoding.

        Note that when this TC is used for an object
        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
```

```
        RFC 3416.

        Note that the size of PmUTF8String object is
        measured in octets, not characters."
       SYNTAX       OCTET STRING (SIZE (0..65535))

-- The policy table

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

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

PmPolicyEntry ::= SEQUENCE {
    pmPolicyAdminGroup            PmUTF8String,
    pmPolicyIndex                 Unsigned32,
    pmPolicyPrecedenceGroup       PmUTF8String,
    pmPolicyPrecedence            Unsigned32,
    pmPolicySchedule              Unsigned32,
    pmPolicyElementTypeFilter     PmUTF8String,
    pmPolicyConditionScriptIndex  Unsigned32,
    pmPolicyActionScriptIndex     Unsigned32,
    pmPolicyParameters            OCTET STRING,
    pmPolicyConditionMaxLatency   Unsigned32,
    pmPolicyActionMaxLatency      Unsigned32,
    pmPolicyMaxIterations         Unsigned32,
    pmPolicyDescription           PmUTF8String,
    pmPolicyMatches               Gauge32,
    pmPolicyAbnormalTerminations  Gauge32,
    pmPolicyExecutionErrors       Counter32,
    pmPolicyDebugging             INTEGER,
    pmPolicyAdminStatus           INTEGER,
    pmPolicyStorageType           StorageType,
    pmPolicyRowStatus             RowStatus
```

```
}

pmPolicyAdminGroup OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE(0..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An administratively assigned string that can be used to group
        policies for convenience, for readability, or to simplify
        configuration of access control.

        The value of this string does not affect policy processing in
        any way.  If grouping is not desired or necessary, this object
        may be set to a zero-length string."
    ::= { pmPolicyEntry 1 }

pmPolicyIndex OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "A unique index for this policy entry, unique among all
         policies regardless of administrative group."
    ::= { pmPolicyEntry 2 }

pmPolicyPrecedenceGroup OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (0..32))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "An administratively assigned string that is used to group
        policies.  For each element, only one policy in the same
        precedence group may be active on that element.  If multiple
        policies would be active on an element (because their
        conditions return non-zero), the execution environment will
        only allow the policy with the highest value of
        pmPolicyPrecedence to be active.

        All values of this object must have been successfully
        transformed by Stringprep RFC 3454.  Management stations
        must perform this translation and must only set this object to
        string values that have been transformed."
    ::= { pmPolicyEntry 3 }

pmPolicyPrecedence OBJECT-TYPE
    SYNTAX      Unsigned32 (0..65535)
    MAX-ACCESS  read-create
    STATUS      current
```

```
    DESCRIPTION
        "If, while checking to see which policy conditions match an
        element, 2 or more ready policies in the same precedence group
        match the same element, the pmPolicyPrecedence object provides
        the rule to arbitrate which single policy will be active on
        'this element'.  Of policies in the same precedence group, only
        the ready and matching policy with the highest precedence
        value (e.g., 2 is higher than 1) will have its policy action
        periodically executed on 'this element'.

        When a policy is active on an element but the condition ceases
        to match the element, its action (if currently running) will
        be allowed to finish and then the condition-matching ready
        policy with the next-highest precedence will immediately
        become active (and have its action run immediately).  If the
        condition of a higher-precedence ready policy suddenly begins
        matching an element, the previously-active policy's action (if
        currently running) will be allowed to finish and then the
        higher precedence policy will immediately become active.  Its
        action will run immediately, and any lower-precedence matching
        policy will not be active anymore.

        In the case where multiple ready policies share the highest
        value, it is an implementation-dependent matter as to which
        single policy action will be chosen.

        Note that if it is necessary to take certain actions after a
        policy is no longer active on an element, these actions should
        be included in a lower-precedence policy that is in the same
        precedence group."
    ::= { pmPolicyEntry 4 }

pmPolicySchedule OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "This policy will be ready if any of the associated schedule
         entries are active.

         If the value of this object is 0, this policy is always
         ready.

         If the value of this object is non-zero but doesn't
         refer to a schedule group that includes an active schedule,
         then the policy will not be ready, even if this is due to a
         misconfiguration of this object or the pmSchedTable."
    ::= { pmPolicyEntry 5 }
```

```
pmPolicyElementTypeFilter OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (0..128))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object specifies the element types for which this policy
        can be executed.

        The format of this object will be a sequence of
        pmElementTypeRegOIDPrefix values, encoded in the following
        BNF form:

        elementTypeFilter:   oid [ ';' oid ]*
                      oid:   subid [ '.' subid ]*
                    subid:   '0' | decimal_constant

        For example, to register for the policy to be run on all
        interface elements, the 'ifEntry' element type will be
        registered as '1.3.6.1.2.1.2.2.1'.

        If a value is included that does not represent a registered
        pmElementTypeRegOIDPrefix, then that value will be ignored."
    ::= { pmPolicyEntry 6 }

pmPolicyConditionScriptIndex OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
         "A pointer to the row or rows in the pmPolicyCodeTable that
         contain the condition code for this policy.  When a policy
         entry is created, a pmPolicyCodeIndex value unused by this
         policy's adminGroup will be assigned to this object.

         A policy condition is one or more PolicyScript statements
         that result(s) in a boolean value that represents whether
         an element is a member of a set of elements upon which an
         action is to be performed.  If a policy is ready and the
         condition returns true for an element of a proper element
         type, and if no higher-precedence policy should be active,
         then the policy is active on that element.

         Condition evaluation stops immediately when any run-time
         exception is detected, and the policyAction is not executed.

         The policyCondition is evaluated for various elements.  Any
         element for which the policyCondition returns any nonzero value
         will match the condition and will have the associated
```

```
            policyAction executed on that element unless a
            higher-precedence policy in the same precedence group also
            matches 'this element'.

            If the condition object is empty (contains no code) or
            otherwise does not return a value, the element will not be
            matched.

            When this condition is executed, if SNMP requests are made to
            the local system and secModel/secName/secLevel aren't
            specified, access to objects is under the security
            credentials of the requester who most recently modified the
            associated pmPolicyAdminStatus object.  If SNMP requests are
            made in which secModel/secName/secLevel are specified, then
            the specified credentials are retrieved from the local
            configuration datastore only if VACM is configured to
            allow access to the requester who most recently modified the
            associated pmPolicyAdminStatus object.  See the Security
            Considerations section for more information."
       ::= { pmPolicyEntry 7 }

   pmPolicyActionScriptIndex OBJECT-TYPE
       SYNTAX      Unsigned32 (1..4294967295)
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
            "A pointer to the row or rows in the pmPolicyCodeTable that
            contain the action code for this policy.  When a policy entry
            is created, a pmPolicyCodeIndex value unused by this policy's
            adminGroup will be assigned to this object.

            A PolicyAction is an operation performed on a
            set of elements for which the policy is active.

            Action evaluation stops immediately when any run-time
            exception is detected.

            When this condition is executed, if SNMP requests are made to
            the local system and secModel/secName/secLevel aren't
            specified, access to objects is under the security
            credentials of the requester who most recently modified the
            associated pmPolicyAdminStatus object.  If SNMP requests are
            made in which secModel/secName/secLevel are specified, then
            the specified credentials are retrieved from the local
            configuration datastore only if VACM is configured to
            allow access to the requester who most recently modified the
            associated pmPolicyAdminStatus object.  See the Security
            Considerations section for more information."
```

```
    ::= { pmPolicyEntry 8 }

pmPolicyParameters OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0..65535))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "From time to time, policy scripts may seek one or more
        parameters (e.g., site-specific constants).  These parameters
        may be installed with the script in this object and are
        accessible to the script via the getParameters() function.  If
        it is necessary for multiple parameters to be passed to the
        script, the script can choose whatever encoding/delimiting
        mechanism is most appropriate."
    ::= { pmPolicyEntry 9 }

pmPolicyConditionMaxLatency OBJECT-TYPE
    SYNTAX      Unsigned32 (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 whether it is under control
        of this policy by re-running the condition 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 how the policy agent
        schedules the checking of various elements within this
        interval is an implementation-dependent matter.
        Implementations may wish to re-run a condition more
        quickly if they note a change to the role strings for an
        element."
    ::= { pmPolicyEntry 10 }

pmPolicyActionMaxLatency OBJECT-TYPE
    SYNTAX      Unsigned32 (0..2147483647)
    UNITS       "milliseconds"
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Every element that matches this policy’s condition 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 how the policy agent schedules the policy
            action on various elements within this interval is an
            implementation-dependent matter."
        ::= { pmPolicyEntry 11 }

    pmPolicyMaxIterations OBJECT-TYPE
        SYNTAX      Unsigned32
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
            "If a condition or action script iterates in loops too many
            times in one invocation, the execution environment may
            consider it in an infinite loop or otherwise not acting
            as intended and may be terminated by the execution
            environment.  The execution environment will count the
            cumulative number of times all 'for' or 'while' loops iterated
            and will apply a threshold to determine when to terminate the
            script.  What threshold the execution environment uses is an
            implementation-dependent manner, but the value of
            this object SHOULD be the basis for choosing the threshold for
            each script.  The value of this object represents a
            policy-specific threshold and can be tuned for policies of
            varying workloads.  If this value is zero, no
            threshold will be enforced except for any
            implementation-dependent maximum.  Regardless of this value,
            the agent is allowed to terminate any script invocation that
            exceeds a local CPU or memory limitation.

            Note that the condition and action invocations are tracked
            separately."
        ::= { pmPolicyEntry 12 }

    pmPolicyDescription OBJECT-TYPE
        SYNTAX      PmUTF8String
        MAX-ACCESS  read-create
        STATUS      current
        DESCRIPTION
             "A description of this rule and its significance, typically
             provided by a human."
        ::= { pmPolicyEntry 13 }

    pmPolicyMatches OBJECT-TYPE
        SYNTAX      Gauge32
```

```
    UNITS       "elements"
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
         "The number of elements that, in their most recent execution
         of the associated condition, were matched by the condition."
    ::= { pmPolicyEntry 14 }

pmPolicyAbnormalTerminations OBJECT-TYPE
    SYNTAX      Gauge32
    UNITS       "elements"
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
         "The number of elements that, in their most recent execution
         of the associated condition or action, have experienced a
         run-time exception and terminated abnormally.  Note that if a
         policy was experiencing a run-time exception while processing
         a particular element but runs normally on a subsequent
         invocation, this number can decline."
    ::= { pmPolicyEntry 15 }

pmPolicyExecutionErrors OBJECT-TYPE
    SYNTAX      Counter32
    UNITS       "errors"
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
         "The total number of times that execution of this policy's
         condition or action has been terminated due to run-time
         exceptions."
    ::= { pmPolicyEntry 16 }

pmPolicyDebugging OBJECT-TYPE
    SYNTAX      INTEGER {
                    off(1),
                    on(2)
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "The status of debugging for this policy.  If this is turned
         on(2), log entries will be created in the pmDebuggingTable
         for each run-time exception that is experienced by this
         policy."
    DEFVAL { off }
    ::= { pmPolicyEntry 17 }
```

```
pmPolicyAdminStatus OBJECT-TYPE
    SYNTAX      INTEGER {
                    disabled(1),
                    enabled(2),
                    enabledAutoRemove(3)
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "The administrative status of this policy.

         The policy will be valid only if the associated
         pmPolicyRowStatus is set to active(1) and this object is set
         to enabled(2) or enabledAutoRemove(3).

         If this object is set to enabledAutoRemove(3), the next time
         the associated schedule moves from the active state to the
         inactive state, this policy will immediately be deleted,
         including any associated entries in the pmPolicyCodeTable.

         The following related objects may not be changed unless this
         object is set to disabled(1):
             pmPolicyPrecedenceGroup, pmPolicyPrecedence,
             pmPolicySchedule, pmPolicyElementTypeFilter,
             pmPolicyConditionScriptIndex, pmPolicyActionScriptIndex,
             pmPolicyParameters, and any pmPolicyCodeTable row
             referenced by this policy.
         In order to change any of these parameters, the policy must
         be moved to the disabled(1) state, changed, and then
         re-enabled.

         When this policy moves to either enabled state from the
         disabled state, any cached values of policy condition must be
         erased, and any Policy or PolicyElement scratchpad values for
         this policy should be removed.  Policy execution will begin by
         testing the policy condition on all appropriate elements."
    ::= { pmPolicyEntry 18 }

pmPolicyStorageType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object defines whether this policy and any associated
         entries in the pmPolicyCodeTable are kept in volatile storage
         and lost upon reboot or if this row is backed up by
         non-volatile or permanent storage.
```

```
         If the value of this object is 'permanent', the values for
         the associated pmPolicyAdminStatus object must remain
         writable."
    ::= { pmPolicyEntry 19 }

pmPolicyRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "The row status of this pmPolicyEntry.

         The status may not be set to active if any of the related
         entries in the pmPolicyCode table do not have a status of
         active or if any of the objects in this row are not set to
         valid values.  Only the following objects may be modified
         while in the active state:
             pmPolicyParameters
             pmPolicyConditionMaxLatency
             pmPolicyActionMaxLatency
             pmPolicyDebugging
             pmPolicyAdminStatus

         If this row is deleted, any associated entries in the
         pmPolicyCodeTable will be deleted as well."
    ::= { pmPolicyEntry 20 }

-- Policy Code Table

pmPolicyCodeTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF PmPolicyCodeEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The pmPolicyCodeTable stores the code for policy conditions and
        actions.

        An example of the relationships between the code table and the
        policy table follows:

        pmPolicyTable
            AdminGroup  Index   ConditionScriptIndex  ActionScriptIndex
        A   ''          1       1                     2
        B   'oper'      1       1                     2
        C   'oper'      2       3                     4

        pmPolicyCodeTable
        AdminGroup  ScriptIndex  Segment    Note
```

```
        ''          1            1          Filter for policy A
        ''          2            1          Action for policy A
        'oper'      1            1          Filter for policy B
        'oper'      2            1          Action 1/2 for policy B
        'oper'      2            2          Action 2/2 for policy B
        'oper'      3            1          Filter for policy C
        'oper'      4            1          Action for policy C

        In this example, there are 3 policies: 1 in the '' adminGroup,
        and 2 in the 'oper' adminGroup.  Policy A has been assigned
        script indexes 1 and 2 (these script indexes are assigned out of
        a separate pool per adminGroup), with 1 code segment each for
        the filter and the action.  Policy B has been assigned script
        indexes 1 and 2 (out of the pool for the 'oper' adminGroup).
        While the filter has 1 segment, the action is longer and is
        loaded into 2 segments.  Finally, Policy C has been assigned
        script indexes 3 and 4, with 1 code segment each for the filter
        and the action."
    ::= { pmMib 2 }

pmPolicyCodeEntry OBJECT-TYPE
    SYNTAX      PmPolicyCodeEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry in the policy code table representing one code
        segment.  Entries that share a common AdminGroup/ScriptIndex
        pair make up a single script.  Valid values of ScriptIndex are
        retrieved from pmPolicyConditionScriptIndex and
        pmPolicyActionScriptIndex after a pmPolicyEntry is
        created.  Segments of code can then be written to this table
        with the learned ScriptIndex values.

        The StorageType of this entry is determined by the value of
        the associated pmPolicyStorageType.

        The pmPolicyAdminGroup element of the index represents the
        administrative group of the policy of which this code entry is
        a part."
    INDEX { pmPolicyAdminGroup, pmPolicyCodeScriptIndex,
            pmPolicyCodeSegment }
    ::= { pmPolicyCodeTable 1 }

PmPolicyCodeEntry ::= SEQUENCE {
    pmPolicyCodeScriptIndex    Unsigned32,
    pmPolicyCodeSegment        Unsigned32,
    pmPolicyCodeText           PmUTF8String,
    pmPolicyCodeStatus         RowStatus
```

```
}

pmPolicyCodeScriptIndex OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "A unique index for each policy condition or action.  The code
         for each such condition or action may be composed of multiple
         entries in this table if the code cannot fit in one entry.
         Values of pmPolicyCodeScriptIndex may not be used unless
         they have previously been assigned in the
         pmPolicyConditionScriptIndex or pmPolicyActionScriptIndex
         objects."
    ::= { pmPolicyCodeEntry 1 }

pmPolicyCodeSegment OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "A unique index for each segment of a policy condition or
         action.

         When a policy condition or action spans multiple entries in
         this table, the code of that policy starts from the
         lowest-numbered segment and continues with increasing segment
         values until it ends with the highest-numbered segment."
    ::= { pmPolicyCodeEntry 2 }

pmPolicyCodeText OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (1..1024))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "A segment of policy code (condition or action).  Lengthy
         Policy conditions or actions may be stored in multiple
         segments in this table that share the same value of
         pmPolicyCodeScriptIndex.  When multiple segments are used, it
         is recommended that each segment be as large as is practical.

         Entries in this table are associated with policies by values
         of the pmPolicyConditionScriptIndex and
         pmPolicyActionScriptIndex objects.  If the status of the
         related policy is active, then this object may not be
         modified."
    ::= { pmPolicyCodeEntry 3 }
```

```
pmPolicyCodeStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "The status of this code entry.

         Entries in this table are associated with policies by values
         of the pmPolicyConditionScriptIndex and
         pmPolicyActionScriptIndex objects.  If the status of the
         related policy is active, then this object can not be
         modified (i.e., deleted or set to notInService), nor may new
         entries be created.

         If the status of this object is active, no objects in this
         row may be modified."
    ::= { pmPolicyCodeEntry 4 }

-- Element Type Registration Table

pmElementTypeRegTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF PmElementTypeRegEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A registration table for element types managed by this
        system.

        The Element Type Registration table allows 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 index of the element is
        the index part of the discovered OID.  This index will be
        supplied to policy conditions and actions so that this code
        can inspect and configure the element.

        For example, this table might contain the following entries.
        The first three are agent-installed, and the 4th was
        downloaded by a management station:

  OIDPrefix        MaxLatency  Description              StorageType
  ifEntry          100 mS      interfaces - builtin     readOnly
  0.0              100 mS      system element - builtin readOnly
  frCircuitEntry   100 mS      FR Circuits - builtin    readOnly
  hrSWRunEntry     60 sec      Running Processes        volatile
```

```
        Note that agents may automatically configure elements in this
        table for frequently used element types (interfaces, circuits,
        etc.).  In particular, it may configure elements for whom
        discovery is optimized in one or both of the following ways:

        1. The agent may discover elements by scanning internal data
           structures as opposed to issuing local SNMP requests.  It is
           possible to recreate the exact semantics described in this
           table even if local SNMP requests are not issued.

        2. The agent may receive asynchronous notification of new
           elements (for example, ’card inserted’) and use that
           information to instantly create elements rather than
           through polling.  A similar feature might be available for
           the deletion of elements.

        Note that the disposition of agent-installed entries is
        described by the pmPolicyStorageType object."
    ::= { pmMib 3 }

pmElementTypeRegEntry OBJECT-TYPE
    SYNTAX      PmElementTypeRegEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "A registration of an element type.

        Note that some values of this table’s index may result in an
        instance name that exceeds a length of 128 sub-identifiers,
        which exceeds the maximum for the SNMP protocol.
        Implementations should take care to avoid such values."
    INDEX       { pmElementTypeRegOIDPrefix }
    ::= { pmElementTypeRegTable 1 }

PmElementTypeRegEntry ::= SEQUENCE {
    pmElementTypeRegOIDPrefix     OBJECT IDENTIFIER,
    pmElementTypeRegMaxLatency    Unsigned32,
    pmElementTypeRegDescription   PmUTF8String,
    pmElementTypeRegStorageType   StorageType,
    pmElementTypeRegRowStatus     RowStatus
}

pmElementTypeRegOIDPrefix OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This OBJECT IDENTIFIER value identifies a table in which all
```

```
        elements of this type will be found.  Every row in the
        referenced table will be treated as an element for the
        period of time that it remains in the table.  The agent will
        then execute policy conditions and actions as appropriate on
        each of these elements.

        This object identifier value is specified down to the 'entry'
        component (e.g., ifEntry) of the identifier.

        The index of each discovered row will be passed to each
        invocation of the policy condition and policy action.

        The actual mechanism by which instances are discovered is
        implementation dependent.  Periodic walks of the table to
        discover the rows in the table is one such mechanism.  This
        mechanism has the advantage that it can be performed by an
        agent with no knowledge of the names, syntax, or semantics
        of the MIB objects in the table.  This mechanism also serves as
        the reference design.  Other implementation-dependent
        mechanisms may be implemented that are more efficient (perhaps
        because they are hard coded) or that don't require polling.
        These mechanisms must discover the same elements as would the
        table-walking reference design.

        This object can contain a OBJECT IDENTIFIER, '0.0'.
        '0.0' represents the single instance of the system
        itself and provides an execution context for policies to
        operate on the 'system element' and on MIB objects
        modeled as scalars.  For example, '0.0' gives an execution
        context for policy-based selection of the operating system
        code version (likely modeled as a scalar MIB object).  The
        element type '0.0' always exists; as a consequence, no actual
        discovery will take place, and the pmElementTypeRegMaxLatency
        object will have no effect for the '0.0' element
        type.  However, if the '0.0' element type is not registered in
        the table, policies will not be executed on the '0.0' element.

        When a policy is invoked on behalf of a '0.0' entry in this
        table, the element name will be '0.0', and there is no index
        of 'this element' (in other words, it has zero length).

        As this object is used in the index for the
        pmElementTypeRegTable, users of this table should be careful
        not to create entries that would result in instance names with
        more than 128 sub-identifiers."
    ::= { pmElementTypeRegEntry 2 }
```

```
pmElementTypeRegMaxLatency OBJECT-TYPE
    SYNTAX      Unsigned32
    UNITS       "milliseconds"
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The PM agent is responsible for discovering new elements of
        types that are registered.  This object lets the manager
        control the maximum amount of time that may pass between the
        time an element is created and when it is discovered.

        In other words, in any given interval of this duration, all
        new elements must be discovered.  Note that how the policy
        agent schedules the checking of various elements within this
        interval is an implementation-dependent matter."
    ::= { pmElementTypeRegEntry 3 }

pmElementTypeRegDescription OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (0..64))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "A descriptive label for this registered type."
    ::= { pmElementTypeRegEntry 4 }

pmElementTypeRegStorageType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object defines whether this row is kept
         in volatile storage and lost upon reboot or
         backed up by non-volatile or permanent storage.

         If the value of this object is 'permanent', no values in the
         associated row have to be writable."
    ::= { pmElementTypeRegEntry 5 }

pmElementTypeRegRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The status of this registration entry.

        If the value of this object is active, no objects in this row
        may be modified."
    ::= { pmElementTypeRegEntry 6 }
```

```
-- Role Table

pmRoleTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF PmRoleEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The pmRoleTable is a read-create table that organizes role
        strings sorted by element.  This table is used to create and
        modify role strings and their associations, as well as to allow
        a management station to learn about the existence of roles and
        their associations.

        It is the responsibility of the agent to keep track of any
        re-indexing of the underlying SNMP elements and to continue to
        associate role strings with the element with which they were
        initially configured.

        Policy MIB agents that have elements in multiple local SNMP
        contexts have to allow some roles to be assigned to elements
        in particular contexts.  This is particularly true when some
        elements have the same names in different contexts and the
        context is required to disambiguate them.  In those situations,
        a value for the pmRoleContextName may be provided.  When a
        pmRoleContextName value is not provided, the assignment is to
        the element in the default context.

        Policy MIB agents that discover elements on other systems and
        execute policies on their behalf need to have access to role
        information for these remote elements.  In such situations,
        role assignments for other systems can be stored in this table
        by providing values for the pmRoleContextEngineID parameters.

    For example:
    Example:
    element       role    context ctxEngineID   #comment
    ifindex.1     gold                          local, default context
    ifindex.2     gold                          local, default context
    repeaterid.1  foo     rptr1                 local, rptr1 context
    repeaterid.1  bar     rptr2                 local, rptr2 context
    ifindex.1     gold    ’’      A             different system
    ifindex.1     gold    ’’      B             different system

         The agent must store role string associations in non-volatile
         storage."
    ::= { pmMib 4 }
```

```
pmRoleEntry OBJECT-TYPE
    SYNTAX      PmRoleEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "A role string entry associates a role string with an
         individual element.

         Note that some combinations of index values may result in an
         instance name that exceeds a length of 128 sub-identifiers,
         which exceeds the maximum for the SNMP
         protocol.  Implementations should take care to avoid such
         combinations."
    INDEX       { pmRoleElement, pmRoleContextName,
                  pmRoleContextEngineID, pmRoleString }
    ::= { pmRoleTable 1 }

PmRoleEntry ::= SEQUENCE {
    pmRoleElement          RowPointer,
    pmRoleContextName      SnmpAdminString,
    pmRoleContextEngineID  OCTET STRING,
    pmRoleString           PmUTF8String,
    pmRoleStatus           RowStatus
}

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

         For example, if the element is interface 3, then this object
         will contain the OID for 'ifIndex.3'.

         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.

         As this object is used in the index for the pmRoleTable,
         users of this table should be careful not to create entries
         that would result in instance names with more than 128
         sub-identifiers."
    ::= { pmRoleEntry 1 }
```

```
pmRoleContextName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE (0..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "If the associated element is not in the default SNMP context
        for the target system, this object is used to identify the
        context.  If the element is in the default context, this object
        is equal to the empty string."
    ::= { pmRoleEntry 2 }

pmRoleContextEngineID OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0 | 5..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "If the associated element is on a remote system, this object
        is used to identify the remote system.  This object contains
        the contextEngineID of the system for which this role string
        assignment is valid.  If the element is on the local system
        this object will be the empty string."
    ::= { pmRoleEntry 3 }

pmRoleString OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (0..64))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "The role string that is associated with an element through
         this table.  All role strings must have been successfully
         transformed by Stringprep RFC 3454.  Management stations
         must perform this translation and must only set this object
         to string values that have been transformed.

         A role string is an administratively specified characteristic
         of a managed element (for example, an interface).  It is a
         selector for policy rules, that determines the applicability of
         the rule to a particular managed element."
    ::= { pmRoleEntry 4 }

pmRoleStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "The status of this role string.
```

```
         If the value of this object is active, no object in this row
         may be modified."
    ::= { pmRoleEntry 5 }

-- Capabilities table

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 so that
         management stations can learn of an agent's capabilities and
         differentially install policies based on the capabilities.

         Capabilities are expressed at the system level.  There can be
         variation in how capabilities are realized from one vendor or
         model to the next.  Management systems should consider these
         differences before selecting which policy to install in a
         system."
    ::= { pmMib 5 }

pmCapabilitiesEntry OBJECT-TYPE
    SYNTAX      PmCapabilitiesEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "A capabilities entry holds an OID indicating support for a
         particular capability.  Capabilities may include hardware and
         software functions and the implementation of MIB
         Modules.  The semantics of the OID are defined in the
         description of pmCapabilitiesType.

         Entries appear in this table if any element in the system has
         a specific capability.  A capability should appear in this
         table only once, regardless of the number of elements in the
         system with that capability.  An entry is removed from this
         table when the last element in the system that has the
         capability is removed.  In some cases, capabilities are
         dynamic and exist only in software.  This table should have an
         entry for the capability even if there are no current
         instances.  Examples include systems with database or WEB
         services.  While the system has the ability to create new
         databases or WEB services, the entry should exist.  In these
         cases, the ability to create these services could come from
         other processes that are running in the system, even though
         there are no currently open databases or WEB servers running.
```

```
            Capabilities may include the implementation of MIB Modules
            but need not be limited to those that represent MIB Modules
            with one or more configurable objects.  It may also be
            valuable to include entries for capabilities that do not
            include configuration objects, as that information, in
            combination with other entries in this table, might be used
            by the management software to determine whether to
            install a policy.

            Vendor software may also add entries in this table to express
            capabilities from their private branch.

            Note that some values of this table's index may result in an
            instance name that exceeds a length of 128 sub-identifiers,
            which exceeds the maximum for the SNMP
            protocol.  Implementations should take care to avoid such
            values."
       INDEX       { pmCapabilitiesType }
       ::= { pmCapabilitiesTable 1 }

   PmCapabilitiesEntry ::= SEQUENCE {
       pmCapabilitiesType               OBJECT IDENTIFIER
   }

   pmCapabilitiesType OBJECT-TYPE
       SYNTAX      OBJECT IDENTIFIER
       MAX-ACCESS  read-only
       STATUS      current
       DESCRIPTION
            "There are three types of OIDs that may be present in the
            pmCapabilitiesType object:

            1) The OID of a MODULE-COMPLIANCE macro that represents the
            highest level of compliance realized by the agent for that
            MIB Module.  For example, an agent that implements the OSPF
            MIB Module at the highest level of compliance would have the
            value of '1.3.6.1.2.1.14.15.2' in the pmCapabilitiesType
            object.  For software that realizes standard MIB
            Modules that do not have compliance statements, the base OID
            of the MIB Module should be used instead.  If the OSPF MIB
            Module had not been created with a compliance statement, then
            the correct value of the pmCapabilitiesType would be
            '1.3.6.1.2.1.14'.  In the cases where multiple compliance
            statements in a MIB Module are supported by the agent, and
            where one compliance statement does not by definition include
            the other, each of the compliance OIDs would have entries in
            this table.
```

```
         MIB Documents can contain more than one MIB Module.  In the
         case of OSPF, there is a second MIB Module
         that describes notifications for the OSPF Version 2 Protocol.
         If the agent also realizes these functions, an entry will
         also exist for those capabilities in this table.

         2) Vendors should install OIDs in this table that represent
         vendor-specific capabilities.  These capabilities can be
         expressed just as those described above for MIB Modules on
         the standards track.  In addition, vendors may install any
         OID they desire from their registered branch.  The OIDs may be
         at any level of granularity, from the root of their entire
         branch to an instance of a single OID.  There is no
         restriction on the number of registrations they may make,
         though care should be taken to avoid unnecessary entries.

         3) OIDs that represent one capability or a collection of
         capabilities that could be any collection of MIB Objects or
         hardware or software functions may be created in working
         groups and registered in a MIB Module.  Other entities (e.g.,
         vendors) may also make registrations.  Software will register
         these standard capability OIDs, as well as vendor specific
         OIDs.

         If the OID for a known capability is not present in the
         table, then it should be assumed that the capability is not
         implemented.

         As this object is used in the index for the
         pmCapabilitiesTable, users of this table should be careful
         not to create entries that would result in instance names
         with more than 128 sub-identifiers."
    ::= { pmCapabilitiesEntry 1 }

-- Capabilities override table

pmCapabilitiesOverrideTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF PmCapabilitiesOverrideEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "The pmCapabilitiesOverrideTable allows management stations
         to override pmCapabilitiesTable entries that have been
         registered by the agent.  This facility can be used to avoid
         situations in which managers in the network send policies to
         a system that has advertised a capability in the
         pmCapabilitiesTable but that should not be installed on this
         particular system.  One example could be newly deployed
```

```
         equipment that is still in a trial state in a trial state or
         resources reserved for some other administrative reason.
         This table can also be used to override entries in the
         pmCapabilitiesTable through the use of the
         pmCapabilitiesOverrideState object.  Capabilities can also be
         declared available in this table that were not registered in
         the pmCapabilitiesTable.  A management application can make
         an entry in this table for any valid OID and declare the
         capability available by setting the
         pmCapabilitiesOverrideState for that row to valid(1)."
    ::= { pmMib 6 }

pmCapabilitiesOverrideEntry OBJECT-TYPE
    SYNTAX      PmCapabilitiesOverrideEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "An entry in this table indicates whether a particular
         capability is valid or invalid.

         Note that some values of this table's index may result in an
         instance name that exceeds a length of 128 sub-identifiers,
         which exceeds the maximum for the SNMP
         protocol.  Implementations should take care to avoid such
         values."
    INDEX       { pmCapabilitiesOverrideType }
    ::= { pmCapabilitiesOverrideTable 1 }

PmCapabilitiesOverrideEntry ::= SEQUENCE {
    pmCapabilitiesOverrideType              OBJECT IDENTIFIER,
    pmCapabilitiesOverrideState             INTEGER,
    pmCapabilitiesOverrideRowStatus         RowStatus
}

pmCapabilitiesOverrideType OBJECT-TYPE
    SYNTAX      OBJECT IDENTIFIER
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "This is the OID of the capability that is declared valid or
         invalid by the pmCapabilitiesOverrideState value for this
         row.  Any valid OID, as described in the pmCapabilitiesTable,
         is permitted in the pmCapabilitiesOverrideType object.  This
         means that capabilities can be expressed at any level, from a
         specific instance of an object to a table or entire module.
         There are no restrictions on whether these objects are from
         standards track MIB documents or in the private branch of the
         MIB.
```

```
         If an entry exists in this table for which there is a
         corresponding entry in the pmCapabilitiesTable, then this entry
         shall have precedence over the entry in the
         pmCapabilitiesTable.  All entries in this table must be
         preserved across reboots.

         As this object is used in the index for the
         pmCapabilitiesOverrideTable, users of this table should be
         careful not to create entries that would result in instance
         names with more than 128 sub-identifiers."
    ::= { pmCapabilitiesOverrideEntry 1 }

pmCapabilitiesOverrideState OBJECT-TYPE
    SYNTAX      INTEGER {
                    invalid(1),
                    valid(2)
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "A pmCapabilitiesOverrideState of invalid indicates that
         management software should not send policies to this system
         for the capability identified in the
         pmCapabilitiesOverrideType for this row of the table.  This
         behavior is the same whether the capability represented by
         the pmCapabilitiesOverrideType exists only in this table
         (that is, it was installed by an external management
         application) or exists in this table as well as the
         pmCapabilitiesTable.  This would be the case when a manager
         wanted to disable a capability that the native management
         system found and registered in the pmCapabilitiesTable.

         An entry in this table that has a pmCapabilitiesOverrideState
         of valid should be treated as though it appeared in the
         pmCapabilitiesTable.  If the entry also exists in the
         pmCapabilitiesTable in the pmCapabilitiesType object, and if
         the value of this object is valid, then the system shall
         operate as though this entry did not exist and policy
         installations and executions will continue in a normal
         fashion."
    ::= { pmCapabilitiesOverrideEntry 2 }

pmCapabilitiesOverrideRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
         "The row status of this pmCapabilitiesOverrideEntry.
```

```
             If the value of this object is active, no object in this row
             may be modified."
        ::= { pmCapabilitiesOverrideEntry 3 }

-- The Schedule Group

pmSchedLocalTime OBJECT-TYPE
    SYNTAX      DateAndTime (SIZE (11))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
        "The local time used by the scheduler.  Schedules that
         refer to calendar time will use the local time indicated
         by this object.  An implementation MUST return all 11 bytes
         of the DateAndTime textual-convention so that a manager
         may retrieve the offset from GMT time."
    ::= { pmMib 7 }

--
-- The schedule table that controls the scheduler.
--

pmSchedTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF PmSchedEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "This table defines schedules for policies."
    ::= { pmMib 8 }

pmSchedEntry OBJECT-TYPE
    SYNTAX      PmSchedEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "An entry describing a particular schedule.

        Unless noted otherwise, writable objects of this row can be
        modified independently of the current value of pmSchedRowStatus,
        pmSchedAdminStatus and pmSchedOperStatus.  In particular, it
        is legal to modify pmSchedWeekDay, pmSchedMonth, and
        pmSchedDay when pmSchedRowStatus is active."
    INDEX { pmSchedIndex }
    ::= { pmSchedTable 1 }
```

```
PmSchedEntry ::= SEQUENCE {
    pmSchedIndex          Unsigned32,
    pmSchedGroupIndex     Unsigned32,
    pmSchedDescr          PmUTF8String,
    pmSchedTimePeriod     PmUTF8String,
    pmSchedMonth          BITS,
    pmSchedDay            BITS,
    pmSchedWeekDay        BITS,
    pmSchedTimeOfDay      PmUTF8String,
    pmSchedLocalOrUtc     INTEGER,
    pmSchedStorageType    StorageType,
    pmSchedRowStatus      RowStatus
}

pmSchedIndex OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "The locally unique, administratively assigned index for this
        scheduling entry."
    ::= { pmSchedEntry 1 }

pmSchedGroupIndex OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The locally unique, administratively assigned index for the
        schedule group this scheduling entry belongs to.

        To assign multiple schedule entries to the same group, the
        pmSchedGroupIndex of each entry in the group will be set to
        the same value.  This pmSchedGroupIndex value must be equal to
        the pmSchedIndex of one of the entries in the group.  If the
        entry whose pmSchedIndex equals the pmSchedGroupIndex
        for the group is deleted, the agent will assign a new
        pmSchedGroupIndex to all remaining members of the group.

        If an entry is not a member of a group, its pmSchedGroupIndex
        must be assigned to the value of its pmSchedIndex.

        Policies that are controlled by a group of schedule entries
        are active when any schedule in the group is active."
    ::= { pmSchedEntry 2 }
```

```
pmSchedDescr OBJECT-TYPE
    SYNTAX      PmUTF8String
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The human-readable description of the purpose of this
        scheduling entry."
    DEFVAL { ''H }
    ::= { pmSchedEntry 3 }

pmSchedTimePeriod OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (0..31))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The overall range of calendar dates and times over which this
        schedule is active.  It is stored in a slightly extended version
        of the format for a 'period-explicit' defined in RFC 2445.
        This format is expressed as a string representing the
        starting date and time, in which the character 'T' indicates
        the beginning of the time portion, followed by the solidus
        character, '/', followed by a similar string representing an
        end date and time.  The start of the period MUST be before the
        end of the period.  Date-Time values are expressed as
        substrings of the form 'yyyymmddThhmmss'.  For example:

            20000101T080000/20000131T130000

              January 1, 2000, 0800 through January 31, 2000, 1PM

        The 'Date with UTC time' format defined in RFC 2445 in which
        the Date-Time string ends with the character 'Z' is not
        allowed.

        This 'period-explicit' format is also extended to allow two
        special cases in which one of the Date-Time strings is
        replaced with a special string defined in RFC 2445:

        1. If the first Date-Time value is replaced with the string
           'THISANDPRIOR', then the value indicates that the schedule
           is active at any time prior to the Date-Time that appears
           after the '/'.

        2. If the second Date-Time is replaced with the string
           'THISANDFUTURE', then the value indicates that the schedule
           is active at any time after the Date-Time that appears
           before the '/'.
```

```
        Note that although RFC 2445 defines these two strings, they are
        not specified for use in the 'period-explicit' format.  The use
        of these strings represents an extension to the
        'period-explicit' format."
    ::= { pmSchedEntry 4 }

pmSchedMonth OBJECT-TYPE
    SYNTAX      BITS {
                    january(0),
                    february(1),
                    march(2),
                    april(3),
                    may(4),
                    june(5),
                    july(6),
                    august(7),
                    september(8),
                    october(9),
                    november(10),
                    december(11)
                }

    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Within the overall time period specified in the
        pmSchedTimePeriod object, the value of this object specifies
        the specific months within that time period when the schedule
        is active.  Setting all bits will cause the schedule to act
        independently of the month."
    DEFVAL { { january, february, march, april, may, june, july,
               august, september, october, november, december } }
    ::= { pmSchedEntry 5 }

pmSchedDay OBJECT-TYPE
    SYNTAX      BITS {
                    d1(0),   d2(1),   d3(2),   d4(3),   d5(4),
                    d6(5),   d7(6),   d8(7),   d9(8),   d10(9),
                    d11(10), d12(11), d13(12), d14(13), d15(14),
                    d16(15), d17(16), d18(17), d19(18), d20(19),
                    d21(20), d22(21), d23(22), d24(23), d25(24),
                    d26(25), d27(26), d28(27), d29(28), d30(29),
                    d31(30),
                    r1(31),  r2(32),  r3(33),  r4(34),  r5(35),
                    r6(36),  r7(37),  r8(38),  r9(39),  r10(40),
                    r11(41), r12(42), r13(43), r14(44), r15(45),
                    r16(46), r17(47), r18(48), r19(49), r20(50),
                    r21(51), r22(52), r23(53), r24(54), r25(55),
```

```
                    r26(56), r27(57), r28(58), r29(59), r30(60),
                    r31(61)
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Within the overall time period specified in the
        pmSchedTimePeriod object, the value of this object specifies
        the specific days of the month within that time period when
        the schedule is active.

        There are two sets of bits one can use to define the day
        within a month:

        Enumerations starting with the letter 'd' indicate a
        day in a month relative to the first day of a month.
        The first day of the month can therefore be specified
        by setting the bit d1(0), and d31(30) means the last
        day of a month with 31 days.

        Enumerations starting with the letter 'r' indicate a
        day in a month in reverse order, relative to the last
        day of a month.  The last day in the month can therefore
        be specified by setting the bit r1(31), and r31(61) means
        the first day of a month with 31 days.

        Setting multiple bits will include several days in the set
        of possible days for this schedule.  Setting all bits starting
        with the letter 'd' or all bits starting with the letter 'r'
        will cause the schedule to act independently of the day of the
        month."
    DEFVAL { {  d1, d2, d3, d4, d5, d6, d7, d8, d9, d10,
                d11, d12, d13, d14, d15, d16, d17, d18, d19, d20,
                d21, d22, d23, d24, d25, d26, d27, d28, d29, d30,
                d31, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10,
                r11, r12, r13, r14, r15, r16, r17, r18, r19, r20,
                r21, r22, r23, r24, r25, r26, r27, r28, r29, r30,
                r31 } }
    ::= { pmSchedEntry 6 }

pmSchedWeekDay OBJECT-TYPE
    SYNTAX      BITS {
                    sunday(0),
                    monday(1),
                    tuesday(2),
                    wednesday(3),
                    thursday(4),
                    friday(5),
```

```
                    saturday(6)
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "Within the overall time period specified in the
        pmSchedTimePeriod object, the value of this object specifies
        the specific days of the week within that time period when
        the schedule is active.  Setting all bits will cause the
        schedule to act independently of the day of the week."
    DEFVAL { { sunday, monday, tuesday, wednesday, thursday,
               friday, saturday } }
    ::= { pmSchedEntry 7 }

pmSchedTimeOfDay OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (0..15))
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION

        "Within the overall time period specified in the
        pmSchedTimePeriod object, the value of this object specifies
        the range of times in a day when the schedule is active.

        This value is stored in a format based on the RFC 2445 format
        for 'time': The character 'T' followed by a 'time' string,
        followed by the solidus character, '/', followed by the
        character 'T', followed by a second time string.  The first time
        indicates the beginning of the range, and the second time
        indicates the end.  Thus, this value takes the following
        form:

            'Thhmmss/Thhmmss'.

        The second substring always identifies a later time than the
        first substring.  To allow for ranges that span midnight,
        however, the value of the second string may be smaller than
        the value of the first substring.  Thus, 'T080000/T210000'
        identifies the range from 0800 until 2100, whereas
        'T210000/T080000' identifies the range from 2100 until 0800 of
        the following day.

        When a range spans midnight, by definition it includes parts
        of two successive days.  When one of these days is also
        selected by either the MonthOfYearMask, DayOfMonthMask, and/or
        DayOfWeekMask, but the other day is not, then the policy is
        active only during the portion of the range that falls on the
        selected day.  For example, if the range extends from 2100
```

```
        until 0800, and the day of week mask selects Monday and
        Tuesday, then the policy is active during the following three
        intervals:

            From midnight Sunday until 0800 Monday
            From 2100 Monday until 0800 Tuesday
            From 2100 Tuesday until 23:59:59 Tuesday

         Setting this value to 'T000000/T235959' will cause the
         schedule to act independently of the time of day."
    DEFVAL { '543030303030302F54323335393539'H } -- T000000/T235959
    ::= { pmSchedEntry 8 }

pmSchedLocalOrUtc OBJECT-TYPE
    SYNTAX      INTEGER {
                    localTime(1),
                    utcTime(2)
                }
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object indicates whether the times represented in the
        TimePeriod object and in the various Mask objects represent
        local times or UTC times."
    DEFVAL { utcTime }
    ::= { pmSchedEntry 9 }

pmSchedStorageType OBJECT-TYPE
    SYNTAX      StorageType
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "This object defines whether this schedule entry is kept
         in volatile storage and lost upon reboot or
         backed up by non-volatile or permanent storage.

         Conceptual rows having the value 'permanent' must allow write
         access to the columnar objects pmSchedDescr, pmSchedWeekDay,
         pmSchedMonth, and pmSchedDay.

         If the value of this object is 'permanent', no values in the
         associated row have to be writable."
    DEFVAL { volatile }
    ::= { pmSchedEntry 10 }
```

```
pmSchedRowStatus OBJECT-TYPE
    SYNTAX      RowStatus
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
        "The status of this schedule entry.

         If the value of this object is active, no object in this row
         may be modified."
    ::= { pmSchedEntry 11 }

-- Policy Tracking

-- The "policy to element" (PE) table and the "element to policy" (EP)
-- table track the status of execution contexts grouped by policy and
-- element respectively.

pmTrackingPETable OBJECT-TYPE
    SYNTAX      SEQUENCE OF PmTrackingPEEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "The pmTrackingPETable describes what elements
         are active (under control of) a policy.  This table is indexed
         in order to optimize retrieval of the entire status for a
         given policy."
    ::= { pmMib 9 }

pmTrackingPEEntry OBJECT-TYPE
    SYNTAX      PmTrackingPEEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "An entry in the pmTrackingPETable.  The pmPolicyIndex in
         the index specifies the policy tracked by this entry.

         Note that some combinations of index values may result in an
         instance name that exceeds a length of 128 sub-identifiers,
         which exceeds the maximum for the SNMP
         protocol.  Implementations should take care to avoid such
         combinations."
    INDEX       { pmPolicyIndex, pmTrackingPEElement,
                  pmTrackingPEContextName, pmTrackingPEContextEngineID }
    ::= { pmTrackingPETable 1 }
```

```
PmTrackingPEEntry ::= SEQUENCE {
    pmTrackingPEElement          RowPointer,
    pmTrackingPEContextName      SnmpAdminString,
    pmTrackingPEContextEngineID  OCTET STRING,
    pmTrackingPEInfo             BITS
}

pmTrackingPEElement OBJECT-TYPE
    SYNTAX      RowPointer
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "The element that is acted upon by the associated policy.

         As this object is used in the index for the
         pmTrackingPETable, users of this table should be careful not
         to create entries that would result in instance names with
         more than 128 sub-identifiers."
    ::= { pmTrackingPEEntry 1 }

pmTrackingPEContextName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE (0..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "If the associated element is not in the default SNMP context
        for the target system, this object is used to identify the
        context.  If the element is in the default context, this object
        is equal to the empty string."
    ::= { pmTrackingPEEntry 2 }

pmTrackingPEContextEngineID OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0 | 5..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "If the associated element is on a remote system, this object
        is used to identify the remote system.  This object contains
        the contextEngineID of the system on which the associated
        element resides.  If the element is on the local system,
        this object will be the empty string."
    ::= { pmTrackingPEEntry 3 }

pmTrackingPEInfo OBJECT-TYPE
    SYNTAX      BITS {
                    actionSkippedDueToPrecedence(0),
                    conditionRunTimeException(1),
                    conditionUserSignal(2),
```

```
                    actionRunTimeException(3),
                    actionUserSignal(4)
                }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
         "This object returns information about the previous policy
         script executions.

         If the actionSkippedDueToPrecedence(1) bit is set, the last
         execution of the associated policy condition returned non-zero,
         but the action is not active, because it was trumped by a
         matching policy condition in the same precedence group with a
         higher precedence value.

         If the conditionRunTimeException(2) bit is set, the last
         execution of the associated policy condition encountered a
         run-time exception and aborted.

         If the conditionUserSignal(3) bit is set, the last
         execution of the associated policy condition called the
         signalError() function.

         If the actionRunTimeException(4) bit is set, the last
         execution of the associated policy action encountered a
         run-time exception and aborted.

         If the actionUserSignal(5) bit is set, the last
         execution of the associated policy action called the
         signalError() function.

         Entries will only exist in this table of one or more bits are
         set.  In particular, if an entry does not exist for a
         particular policy/element combination, it can be assumed that
         the policy’s condition did not match ’this element’."
    ::= { pmTrackingPEEntry 4 }

-- Element to Policy Table

pmTrackingEPTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF PmTrackingEPEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "The pmTrackingEPTable describes what policies
         are controlling an element.  This table is indexed in
         order to optimize retrieval of the status of all policies
         active for a given element."
```

```
    ::= { pmMib 10 }

pmTrackingEPEntry OBJECT-TYPE
    SYNTAX      PmTrackingEPEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "An entry in the pmTrackingEPTable.  Entries exist for all
         element/policy combinations for which the policy's condition
         matches and only if the schedule for the policy is active.

         The pmPolicyIndex in the index specifies the policy
         tracked by this entry.

         Note that some combinations of index values may result in an
         instance name that exceeds a length of 128 sub-identifiers,
         which exceeds the maximum for the SNMP protocol.
         Implementations should take care to avoid such combinations."
    INDEX       { pmTrackingEPElement, pmTrackingEPContextName,
                  pmTrackingEPContextEngineID, pmPolicyIndex }
    ::= { pmTrackingEPTable 1 }

PmTrackingEPEntry ::= SEQUENCE {
    pmTrackingEPElement          RowPointer,
    pmTrackingEPContextName      SnmpAdminString,
    pmTrackingEPContextEngineID  OCTET STRING,
    pmTrackingEPStatus           INTEGER
}

pmTrackingEPElement OBJECT-TYPE
    SYNTAX      RowPointer
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "The element acted upon by the associated policy.

         As this object is used in the index for the
         pmTrackingEPTable, users of this table should be careful
         not to create entries that would result in instance names
         with more than 128 sub-identifiers."
    ::= { pmTrackingEPEntry 1 }

pmTrackingEPContextName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE (0..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "If the associated element is not in the default SNMP context
```

```
            for the target system, this object is used to identify the
            context.  If the element is in the default context, this object
            is equal to the empty string."
        ::= { pmTrackingEPEntry 2 }

    pmTrackingEPContextEngineID OBJECT-TYPE
        SYNTAX      OCTET STRING (SIZE (0 | 5..32))
        MAX-ACCESS  not-accessible
        STATUS      current
        DESCRIPTION
            "If the associated element is on a remote system, this object
            is used to identify the remote system.  This object contains
            the contextEngineID of the system on which the associated
            element resides.  If the element is on the local system,
            this object will be the empty string."
        ::= { pmTrackingEPEntry 3 }

    pmTrackingEPStatus OBJECT-TYPE
        SYNTAX      INTEGER {
                        on(1),
                        forceOff(2)
                    }
        MAX-ACCESS  read-write
        STATUS      current
        DESCRIPTION
             "This entry will only exist if the calendar for the policy is
             active and if the associated policyCondition returned 1 for
             'this element'.

             A policy can be forcibly disabled on a particular element
             by setting this value to forceOff(2).  The agent should then
             act as though the policyCondition 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.  The
             forceOff(2) state may be set even if the entry does not
             previously exist so that future policy invocations can be
             avoided.

             Unless forcibly disabled, if this entry exists, its value
             will be on(1)."
        ::= { pmTrackingEPEntry 4 }

    -- Policy Debugging Table

    pmDebuggingTable OBJECT-TYPE
        SYNTAX      SEQUENCE OF PmDebuggingEntry
        MAX-ACCESS  not-accessible
        STATUS      current
```

```
    DESCRIPTION
         "Policies that have debugging turned on will generate a log
         entry in the policy debugging table for every runtime
         exception that occurs in either the condition or action
         code.

         The pmDebuggingTable logs debugging messages when
         policies experience run-time exceptions in either the condition
         or action code and the associated pmPolicyDebugging object
         has been turned on.

         The maximum number of debugging entries that will be stored
         and the maximum length of time an entry will be kept are an
         implementation-dependent manner.  If entries must
         be discarded to make room for new entries, the oldest entries
         must be discarded first.

         If the system restarts, all debugging entries may be deleted."
    ::= { pmMib 11 }

pmDebuggingEntry OBJECT-TYPE
    SYNTAX      PmDebuggingEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "An entry in the pmDebuggingTable.  The pmPolicyIndex in the
         index specifies the policy that encountered the exception
         that led to this log entry.

         Note that some combinations of index values may result in an
         instance name that exceeds a length of 128 sub-identifiers,
         which exceeds the maximum for the SNMP protocol.
         Implementations should take care to avoid such combinations."
    INDEX       { pmPolicyIndex, pmDebuggingElement,
                  pmDebuggingContextName, pmDebuggingContextEngineID,
                  pmDebuggingLogIndex }
    ::= { pmDebuggingTable 1 }

PmDebuggingEntry ::= SEQUENCE {
    pmDebuggingElement          RowPointer,
    pmDebuggingContextName      SnmpAdminString,
    pmDebuggingContextEngineID  OCTET STRING,
    pmDebuggingLogIndex         Unsigned32,
    pmDebuggingMessage          PmUTF8String
}
```

```
pmDebuggingElement OBJECT-TYPE
    SYNTAX      RowPointer
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "The element the policy was executing on when it encountered
         the error that led to this log entry.

         For example, if the element is interface 3, then this object
         will contain the OID for 'ifIndex.3'.

         As this object is used in the index for the
         pmDebuggingTable, users of this table should be careful
         not to create entries that would result in instance names
         with more than 128 sub-identifiers."
    ::= { pmDebuggingEntry 1 }

pmDebuggingContextName OBJECT-TYPE
    SYNTAX      SnmpAdminString (SIZE (0..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "If the associated element is not in the default SNMP context
        for the target system, this object is used to identify the
        context.  If the element is in the default context, this object
        is equal to the empty string."
    ::= { pmDebuggingEntry 2 }

pmDebuggingContextEngineID OBJECT-TYPE
    SYNTAX      OCTET STRING (SIZE (0 | 5..32))
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
        "If the associated element is on a remote system, this object
        is used to identify the remote system.  This object contains
        the contextEngineID of the system on which the associated
        element resides.  If the element is on the local system,
        this object will be the empty string."
    ::= { pmDebuggingEntry 3 }

pmDebuggingLogIndex OBJECT-TYPE
    SYNTAX      Unsigned32 (1..4294967295)
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
         "A unique index for this log entry among other log entries
         for this policy/element combination."
    ::= { pmDebuggingEntry 4 }
```

```
pmDebuggingMessage OBJECT-TYPE
    SYNTAX      PmUTF8String (SIZE (0..128))
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
         "An error message generated by the policy execution
         environment.  It is recommended that this message include the
         time of day when the message was generated, if known."
    ::= { pmDebuggingEntry 5 }

-- Notifications

pmNotifications OBJECT IDENTIFIER ::= { pmMib 0 }

pmNewRoleNotification NOTIFICATION-TYPE
    OBJECTS     { pmRoleStatus }
    STATUS      current
    DESCRIPTION
        "The pmNewRoleNotification is sent when an agent is configured
        with its first instance of a previously unused role string
        (not every time a new element is given a particular role).

        An instance of the pmRoleStatus object is sent containing
        the new roleString in its index.  In the event that two or
        more elements are given the same role simultaneously, it is an
        implementation-dependent matter as to which pmRoleTable
        instance will be included in the notification."
    ::= { pmNotifications 1 }

pmNewCapabilityNotification NOTIFICATION-TYPE
    OBJECTS     { pmCapabilitiesType }
    STATUS      current
    DESCRIPTION
        "The pmNewCapabilityNotification is sent when an agent
        gains a new capability that did not previously exist in any
        element on the system (not every time an element gains a
        particular capability).

        An instance of the pmCapabilitiesType object is sent containing
        the identity of the new capability.  In the event that two or
        more elements gain the same capability simultaneously, it is an
        implementation-dependent matter as to which pmCapabilitiesType
        instance will be included in the notification."
    ::= { pmNotifications 2 }

pmAbnormalTermNotification NOTIFICATION-TYPE
    OBJECTS     { pmTrackingPEInfo }
    STATUS      current
```

```
    DESCRIPTION
        "The pmAbnormalTermNotification is sent when a policy's
        pmPolicyAbnormalTerminations gauge value changes from zero to
        any value greater than zero and no such notification has been
        sent for that policy in the last 5 minutes.

        The notification contains an instance of the pmTrackingPEInfo
        object where the pmPolicyIndex component of the index
        identifies the associated policy and the rest of the index
        identifies an element on which the policy failed."
    ::= { pmNotifications 3 }

-- Compliance Statements

    pmConformance   OBJECT IDENTIFIER ::= { pmMib 12 }
    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, pmSchedGroup,
                           pmNotificationGroup }
    ::= { pmCompliances 1 }

pmPolicyManagementGroup OBJECT-GROUP
    OBJECTS { pmPolicyPrecedenceGroup, pmPolicyPrecedence,
              pmPolicySchedule, pmPolicyElementTypeFilter,
              pmPolicyConditionScriptIndex, pmPolicyActionScriptIndex,
              pmPolicyParameters,
              pmPolicyConditionMaxLatency, pmPolicyActionMaxLatency,
              pmPolicyMaxIterations,
              pmPolicyDescription, pmPolicyMatches,
              pmPolicyAbnormalTerminations,
              pmPolicyExecutionErrors, pmPolicyDebugging,
              pmPolicyStorageType, pmPolicyAdminStatus,
              pmPolicyRowStatus, pmPolicyCodeText, pmPolicyCodeStatus,
              pmElementTypeRegMaxLatency, pmElementTypeRegDescription,
              pmElementTypeRegStorageType, pmElementTypeRegRowStatus,
              pmRoleStatus,
              pmCapabilitiesType, pmCapabilitiesOverrideState,
              pmCapabilitiesOverrideRowStatus,
              pmTrackingPEInfo,
              pmTrackingEPStatus,
              pmDebuggingMessage }
```

```
    STATUS  current
    DESCRIPTION
        "Objects that allow for the creation and management of
        configuration policies."
    ::=  { pmGroups 1 }

pmSchedGroup OBJECT-GROUP
    OBJECTS { pmSchedLocalTime, pmSchedGroupIndex,
              pmSchedDescr, pmSchedTimePeriod,
              pmSchedMonth, pmSchedDay, pmSchedWeekDay,
              pmSchedTimeOfDay, pmSchedLocalOrUtc, pmSchedStorageType,
              pmSchedRowStatus
            }
    STATUS current
    DESCRIPTION
        "Objects that allow for the scheduling of policies."
    ::= { pmGroups 2 }

pmNotificationGroup NOTIFICATION-GROUP
    NOTIFICATIONS { pmNewRoleNotification,
                    pmNewCapabilityNotification,
                    pmAbnormalTermNotification }
    STATUS        current
    DESCRIPTION
        "Notifications sent by an Policy MIB agent."
    ::= { pmGroups 3 }

pmBaseFunctionLibrary OBJECT IDENTIFIER ::= { pmGroups 4 }

END
```

12. Relationship to Other MIB Modules

When policy-based management is used specifically for (policy-based) configuration, the "Configuring Networks and Devices With SNMP" RFC 3512 [19] document describes configuration management practices, terminology, and an example of a MIB Module that may be helpful to those developing and using this technology.

The Policy MIB accesses system instrumentation for the purposes of policy evaluation, control, notification, monitoring, and error reporting. This information is available to managers in the form of MIB objects. Information about system configuration is modified by the Policy MIB through MIB objects defined in other MIB Modules.

Details about the operational or configuration details of a system are retrieved by the manager via access to the specific MIB objects available in a network element. As such, the Policy MIB can use any standard or vendor-defined object that exists on a managed system. In particular, the Policy MIB may access standard or vendor specific objects that are instance-specific such as BGP timeout parameters and specific interface counters.

13. Security Considerations

This MIB contains no objects for which read access would disclose sensitive information.

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.

With the exception of pmPolicyDescription, pmPolicyDebugging, pmElementTypeRegDescription, and pmSchedDescr, EVERY read-create and read-write object in this MIB should be considered sensitive because if an unauthorized user could manipulate these objects, s/he could cause the Policy MIB system to use the stored credentials of an authorized user to perform unauthorized and potentially harmful operations.

There are no read-only objects in this MIB that contain sensitive information.

SNMP versions prior to SNMPv3 did not include adequate security. 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 module.

It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework (see [16], section 8), including full support for the SNMPv3 cryptographic mechanisms (for authentication and privacy).

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module 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.

An implementation must ensure that access control rules are applied when SNMP operations are performed in policy scripts. To ensure this, an implementation must record and maintain the security credentials of the last entity to modify each policy's pmPolicyAdminStatus object. The credentials to store are the securityModel, securityName, and securityLevel and will be used as input parameters for isAccessAllowed from the Architecture for Describing SNMP Management Frameworks [1]. This mechanism was first introduced in the DISMAN-SCHEDULE-MIB [12].

SNMP requests made when secModel, secName, and secLevel are specified use credentials stored in the local configuration datastore. Access to these credentials depends on the security credentials of the last entity to modify the policy's pmPolicyAdminStatus object. To determine whether the credentials can be accessed, the isAccessAllowed abstract service interface defined in RFC 3411 [1] is called:

```
statusInformation =          -- success or errorIndication
  isAccessAllowed(

  IN   securityModel         -- Security Model used
  IN   securityName          -- principal who wants to access
  IN   securityLevel         -- Level of Security used
  IN   viewType              -- write
  IN   contextName           -- context containing variableName
  IN   variableName          -- OID for an object in the proper
                             -- LCD entry
       )
```

The securityModel, securityName, and securityLevel parameters are set to the values that were recorded when the policy was modified. The viewType is set to write, and the contextName and variableName are set to select any read-create object in the appropriate LCD entry.

Proper configuration of VACM requires that write access to an LCD entry not be given to entities that aren't authorized to use the credentials therein.

Access control for SNMP requests made to the local system where secModel, secName, and secLevel aren't specified depends on the security credentials of the last entity to modify the policy's pmPolicyAdminStatus object. To determine whether the operation should succeed, the isAccessAllowed abstract service interface defined in RFC 3411 [1] is called:

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

The securityModel, securityName, and securityLevel parameters are set to the values that were recorded when the policy was modified. The viewType, contextName, and variableName parameters are set as appropriate for the requested SNMP operation.

Unless all users who have write access to the pmPolicyTable and pmPolicyCodeTable have equivalent access to the managed system, policy scripts could be used by a user to gain the privileges of another user. Therefore, when policy users have different access, access control should be applied so that a user’s policies cannot be modified by another user. To make this more convenient, a user can place all of his or her policies in the same pmPolicyAdminGroup so that a single access control view can apply to all of them.

Some policies may be designed to ensure the security of a network. If these policies have not been installed pending the appearance of a role or capability, some delay will occur in their activation policies when the role or capability appears because a responsible manager must notice the change and install the policy. This delay may expose the device or the network to unacceptable security vulnerabilities during this delay. If the role or capability appears during a time of network stress or when the management station is unavailable, this delay could be extensive, further increasing the exposure. It is recommended that management stations install any security-related policies that might ever be needed on a particular managed device, even if a nonexistent role or capability suggests that it is not needed at a given time.

This MIB allows the delegation of access rights so that a user ("Joe") can instruct a Policy MIB agent to execute remote operations on his behalf that are authorized by keys stored by "Joe" into the usmUserTable. Care needs to be taken to ensure that unauthorized users are unable to configure their policies to use Joe’s keys. Although there are theoretically many ways to configure SNMP security, users are advised to follow the most straightforward way outlined below to minimize complexity and the resulting opportunity for errors.

Assume that Joe has credentials that give him authority to manage agents A, B, and C, as well as the Policy MIB agent "P". Joe will store credentials for Joe@A, Joe@B, and Joe@C in the usmUserTable of the Policy MIB agent. Then the following VACM configuration will be used:

VACM securityToGroupTable
A single entry mapping user Joe@P to group JoesGroup

VACM accessTable
A single entry mapping group JoesGroup to write view JoesView

VACM viewTreeFamilyTable

ViewName	Subtree	Type
JoesView	points to Joe@A in usmUserTable	included
JoesView	points to Joe@B in usmUserTable	included
JoesView	points to Joe@C in usmUserTable	included

In the preceding examples, the notation Joe@A represents the entry indexed by usmUserEngineID and usmUserName, where the SnmpEngineID is that of system A and the usmUserName is "Joe".

14. IANA Considerations

This is a profile of stringprep. It has been registered by the IANA in the stringprep profile registry located at:

http://www.iana.org/assignments/stringprep-profiles

Name of this profile:
Policy MIB Stringprep.

RFC in which the profile is defined:
This document.

Indicator whether this is the newest version of the profile:

This is the first version of Policy MIB Stringprep.

15. Acknowledgements

The authors gratefully acknowledge the significant contributions to this work made by Jeff Case, Patrik Falstrom, Joel Halpern, Pablo Halpern, Bob Moore, Steve Moulton, David Partain, and Walter Weiss.

This MIB uses a security delegation mechanism that was first introduced in the DISMAN-SCHEDULE-MIB [12]. The Schedule table of this MIB borrows heavily from the PolicyTimePeriodCondition of the Policy Core Information Model (PCIM) [18] and from the DISMAN-SCHEDULE-MIB [12].

16. References

16.1. Normative References

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

[2] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[3] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[4] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[5] Presuhn, R., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

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

[7] Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[8] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", BCP 74, RFC 3584, August 2003.

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

[10] International Standards Organization, "Information Technology - Programming Languages - C++", ISO/IEC 14882-1998

[11] Daniele, M. and J. Schoenwaelder, "Textual Conventions for Transport Addresses", RFC 3419, December 2002.

[12] Levi, D. and J. Schoenwaelder, "Definitions of Managed Objects for Scheduling Management Operations", RFC 3231, January 2002.

[13] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[14] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[15] Dawson, F. and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, November 1998.

16.2. Informative References

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

[17] ECMA, "ECMAScript Language Specification", ECMA-262, December 1999

[18] Moore, B., Ellesson, E., Strassner, J., and A. Westerinen, "Policy Core Information Model -- Version 1 Specification", RFC 3060, February 2001.

[19] MacFaden, M., Partain, D., Saperia, J., and W. Tackabury, "Configuring Networks and Devices with Simple Network Management Protocol (SNMP)", RFC 3512, April 2003.

Author’s Addresses

Steve Waldbusser

Phone: +1-650-948-6500
Fax: +1-650-745-0671
EMail: waldbusser@nextbeacon.com

Jon Saperia (WG Co-chair)
JDS Consulting, Inc.
84 Kettell Plain Road.
Stow MA 01775
USA

Phone: +1-978-461--0249
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

Full Copyright Statement

Copyright (C) The Internet Society (2005).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM 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.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights 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; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat 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 implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

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