

## Policy Core Information Model (PCIM) Extensions

### Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

### Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

### Abstract

This document specifies a number of changes to the Policy Core Information Model (PCIM, RFC 3060). Two types of changes are included. First, several completely new elements are introduced, for example, classes for header filtering, that extend PCIM into areas that it did not previously cover. Second, there are cases where elements of PCIM (for example, policy rule priorities) are deprecated, and replacement elements are defined (in this case, priorities tied to associations that refer to policy rules). Both types of changes are done in such a way that, to the extent possible, interoperability with implementations of the original PCIM model is preserved. This document updates RFC 3060.

### Table of Contents

1. Introduction.....	5
2. Changes since RFC 3060.....	5
3. Overview of the Changes.....	6
3.1. How to Change an Information Model.....	6
3.2. List of Changes to the Model.....	6
3.2.1. Changes to PolicyRepository.....	6
3.2.2. Additional Associations and Additional Reusable Elements.....	7
3.2.3. Priorities and Decision Strategies.....	7
3.2.4. Policy Roles.....	8
3.2.5. CompoundPolicyConditions and CompoundPolicyActions.....	8

3.2.6. Variables and Values.....	9
3.2.7. Domain-Level Packet Filtering.....	9
3.2.8. Device-Level Packet Filtering.....	9
4. The Updated Class and Association Class Hierarchies.....	10
5. Areas of Extension to PCIM.....	13
5.1. Policy Scope.....	13
5.1.1. Levels of Abstraction: Domain- and Device-Level Policies.....	13
5.1.2. Administrative and Functional Scopes.....	14
5.2. Reusable Policy Elements.....	15
5.3. Policy Sets.....	16
5.4. Nested Policy Rules.....	16
5.4.1. Usage Rules for Nested Rules.....	17
5.4.2. Motivation.....	17
5.5. Priorities and Decision Strategies.....	18
5.5.1. Structuring Decision Strategies.....	19
5.5.2. Side Effects.....	21
5.5.3. Multiple PolicySet Trees For a Resource.....	21
5.5.4. Deterministic Decisions.....	22
5.6. Policy Roles.....	23
5.6.1. Comparison of Roles in PCIM with Roles in snmpconf.....	23
5.6.2. Addition of PolicyRoleCollection to PCIME.....	24
5.6.3. Roles for PolicyGroups.....	25
5.7. Compound Policy Conditions and Compound Policy Actions....	27
5.7.1. Compound Policy Conditions.....	27
5.7.2. Compound Policy Actions.....	27
5.8. Variables and Values.....	28
5.8.1. Simple Policy Conditions.....	29
5.8.2. Using Simple Policy Conditions.....	29
5.8.3. The Simple Condition Operator.....	31
5.8.4. SimplePolicyActions.....	33
5.8.5. Policy Variables.....	35
5.8.6. Explicitly Bound Policy Variables.....	36
5.8.7. Implicitly Bound Policy Variables.....	37
5.8.8. Structure and Usage of Pre-Defined Variables.....	38
5.8.9. Rationale for Modeling Implicit Variables as Classes.....	39
5.8.10. Policy Values.....	40
5.9. Packet Filtering.....	41
5.9.1. Domain-Level Packet Filters.....	41
5.9.2. Device-Level Packet Filters.....	42
5.10. Conformance to PCIM and PCIME.....	43
6. Class Definitions.....	44
6.1. The Abstract Class "PolicySet".....	44
6.2. Update PCIM's Class "PolicyGroup".....	45
6.3. Update PCIM's Class "PolicyRule".....	45
6.4. The Class "SimplePolicyCondition".....	46

6.5. The Class "CompoundPolicyCondition".....	47
6.6. The Class "CompoundFilterCondition".....	47
6.7. The Class "SimplePolicyAction".....	48
6.8. The Class "CompoundPolicyAction".....	48
6.9. The Abstract Class "PolicyVariable".....	50
6.10. The Class "PolicyExplicitVariable".....	50
6.10.1. The Single-Valued Property "ModelClass".....	51
6.10.2. The Single-Valued Property ModelProperty.....	51
6.11. The Abstract Class "PolicyImplicitVariable".....	51
6.11.1. The Multi-Valued Property "ValueTypes".....	52
6.12. Subclasses of "PolicyImplicitVariable" Specified in PCIME.....	52
6.12.1. The Class "PolicySourceIPv4Variable".....	52
6.12.2. The Class "PolicySourceIPv6Variable".....	52
6.12.3. The Class "PolicyDestinationIPv4Variable".....	53
6.12.4. The Class "PolicyDestinationIPv6Variable".....	53
6.12.5. The Class "PolicySourcePortVariable".....	54
6.12.6. The Class "PolicyDestinationPortVariable".....	54
6.12.7. The Class "PolicyIPProtocolVariable".....	54
6.12.8. The Class "PolicyIPVersionVariable".....	55
6.12.9. The Class "PolicyIPToSVariable".....	55
6.12.10. The Class "PolicyDSCPVariable".....	55
6.12.11. The Class "PolicyFlowIdVariable".....	56
6.12.12. The Class "PolicySourceMACVariable".....	56
6.12.13. The Class "PolicyDestinationMACVariable".....	56
6.12.14. The Class "PolicyVLANVariable".....	56
6.12.15. The Class "PolicyCoSVariable".....	57
6.12.16. The Class "PolicyEthertypeVariable".....	57
6.12.17. The Class "PolicySourceSAPVariable".....	57
6.12.18. The Class "PolicyDestinationSAPVariable".....	58
6.12.19. The Class "PolicySNAPOUIVariable".....	58
6.12.20. The Class "PolicySNAPTypeVariable".....	59
6.12.21. The Class "PolicyFlowDirectionVariable".....	59
6.13. The Abstract Class "PolicyValue".....	59
6.14. Subclasses of "PolicyValue" Specified in PCIME.....	60
6.14.1. The Class "PolicyIPv4AddrValue".....	60
6.14.2. The Class "PolicyIPv6AddrValue".....	61
6.14.3. The Class "PolicyMACAddrValue".....	62
6.14.4. The Class "PolicyStringValue".....	63
6.14.5. The Class "PolicyBitStringValue".....	63
6.14.6. The Class "PolicyIntegerValue".....	64
6.14.7. The Class "PolicyBooleanValue".....	65
6.15. The Class "PolicyRoleCollection".....	65
6.15.1. The Single-Valued Property "PolicyRole".....	66
6.16. The Class "ReusablePolicyContainer".....	66
6.17. Deprecate PCIM's Class "PolicyRepository".....	66
6.18. The Abstract Class "FilterEntryBase".....	67
6.19. The Class "IpHeadersFilter".....	67

6.19.1.	The Property HdrIpVersion.....	68
6.19.2.	The Property HdrSrcAddress.....	68
6.19.3.	The Property HdrSrcAddressEndOfRange.....	68
6.19.4.	The Property HdrSrcMask.....	69
6.19.5.	The Property HdrDestAddress.....	69
6.19.6.	The Property HdrDestAddressEndOfRange.....	69
6.19.7.	The Property HdrDestMask.....	70
6.19.8.	The Property HdrProtocolID.....	70
6.19.9.	The Property HdrSrcPortStart.....	70
6.19.10.	The Property HdrSrcPortEnd.....	70
6.19.11.	The Property HdrDestPortStart.....	71
6.19.12.	The Property HdrDestPortEnd.....	71
6.19.13.	The Property HdrDSCP.....	72
6.19.14.	The Property HdrFlowLabel.....	72
6.20.	The Class "8021Filter".....	72
6.20.1.	The Property 8021HdrSrcMACAddr.....	73
6.20.2.	The Property 8021HdrSrcMACMask.....	73
6.20.3.	The Property 8021HdrDestMACAddr.....	73
6.20.4.	The Property 8021HdrDestMACMask.....	73
6.20.5.	The Property 8021HdrProtocolID.....	74
6.20.6.	The Property 8021HdrPriorityValue.....	74
6.20.7.	The Property 8021HdrVLANID.....	74
6.21.	The Class FilterList.....	74
6.21.1.	The Property Direction.....	75
7.	Association and Aggregation Definitions.....	75
7.1.	The Aggregation "PolicySetComponent".....	75
7.2.	Deprecate PCIM's Aggregation "PolicyGroupInPolicyGroup"...	76
7.3.	Deprecate PCIM's Aggregation "PolicyRuleInPolicyGroup"...	76
7.4.	The Abstract Association "PolicySetInSystem".....	77
7.5.	Update PCIM's Weak Association "PolicyGroupInSystem".....	77
7.6.	Update PCIM's Weak Association "PolicyRuleInSystem".....	78
7.7.	The Abstract Aggregation "PolicyConditionStructure".....	79
7.8.	Update PCIM's Aggregation "PolicyConditionInPolicyRule"...	79
7.9.	The Aggregation "PolicyConditionInPolicyCondition".....	79
7.10.	The Abstract Aggregation "PolicyActionStructure".....	80
7.11.	Update PCIM's Aggregation "PolicyActionInPolicyRule".....	80
7.12.	The Aggregation "PolicyActionInPolicyAction".....	80
7.13.	The Aggregation "PolicyVariableInSimplePolicyCondition"....	80
7.14.	The Aggregation "PolicyValueInSimplePolicyCondition".....	81
7.15.	The Aggregation "PolicyVariableInSimplePolicyAction".....	82
7.16.	The Aggregation "PolicyValueInSimplePolicyAction".....	83
7.17.	The Association "ReusablePolicy".....	83
7.18.	Deprecate PCIM's "PolicyConditionInPolicyRepository".....	84
7.19.	Deprecate PCIM's "PolicyActionInPolicyRepository".....	84
7.20.	The Association ExpectedPolicyValuesForVariable.....	84
7.21.	The Aggregation "ContainedDomain".....	85
7.22.	Deprecate PCIM's "PolicyRepositoryInPolicyRepository"....	86
7.23.	The Aggregation "EntriesInFilterList".....	86

7.23.1. The Reference GroupComponent.....	86
7.23.2. The Reference PartComponent.....	87
7.23.3. The Property EntrySequence.....	87
7.24. The Aggregation "ElementInPolicyRoleCollection".....	87
7.25. The Weak Association "PolicyRoleCollectionInSystem".....	87
8. Intellectual Property.....	88
9. Acknowledgements.....	89
10. Contributors.....	89
11. Security Considerations.....	91
12. Normative References.....	91
13. Informative References.....	91
Author's Address.....	92
Full Copyright Statement.....	93

## 1. Introduction

This document specifies a number of changes to the Policy Core Information Model (PCIM), RFC 3060 [1]. Two types of changes are included. First, several completely new elements are introduced, for example, classes for header filtering, that extend PCIM into areas that it did not previously cover. Second, there are cases where elements of PCIM (for example, policy rule priorities) are deprecated, and replacement elements are defined (in this case, priorities tied to associations that refer to policy rules). Both types of changes are done in such a way that, to the extent possible, interoperability with implementations of the original PCIM model is preserved.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [8].

## 2. Changes since RFC 3060

Section 3.2 contains a short discussion of the changes that this document makes to the RFC 3060 information model. Here is a very brief list of the changes:

1. Deprecate and replace PolicyRepository and its associations.
2. Clarify and expand the ways that PolicyRules and PolicyGroups are aggregated.
3. Change how prioritization for PolicyRules is represented, and introduce administrator-specified decision strategies for rule evaluation.
4. Expand the role of PolicyRoles, and introduce a means of associating a PolicyRole with a resource.
5. Introduce compound policy conditions and compound policy actions into the model.

6. Introduce variables and values into the model.
7. Introduce variable and value subclasses for packet-header filtering.
8. Introduce classes for device-level packet-header filtering.

### 3. Overview of the Changes

#### 3.1. How to Change an Information Model

The Policy Core Information Model is closely aligned with the DMTF's CIM Core Policy model. Since there is no separately documented set of rules for specifying IETF information models such as PCIM, it is reasonable to look to the CIM specifications for guidance on how to modify and extend the model. Among the CIM rules for changing an information model are the following. Note that everything said here about "classes" applies to association classes (including aggregations) as well as to non- association classes.

- o Properties may be added to existing classes.
- o Classes, and individual properties, may be marked as DEPRECATED. If there is a replacement feature for the deprecated class or property, it is identified explicitly. Otherwise the notation "No value" is used. In this document, the notation "DEPRECATED FOR <feature-name>" is used to indicate that a feature has been deprecated, and to identify its replacement feature.
- o Classes may be inserted into the inheritance hierarchy above existing classes, and properties from the existing classes may then be "pulled up" into the new classes. The net effect is that the existing classes have exactly the same properties they had before, but the properties are inherited rather than defined explicitly in the classes.
- o New subclasses may be defined below existing classes.

#### 3.2. List of Changes to the Model

The following subsections provide a very brief overview of the changes to PCIM defined in PCIMe. In several cases, the origin of the change is noted, as QPIM [11], ICPM [12], or QDDIM [15].

##### 3.2.1. Changes to PolicyRepository

Because of the potential for confusion with the Policy Framework component Policy Repository (from the four-box picture: Policy Management Tool, Policy Repository, PDP, PEP), "PolicyRepository" is a bad name for the PCIM class representing a container of reusable policy elements. Thus the class PolicyRepository is being replaced with the class ReusablePolicyContainer. To accomplish this change, it is necessary to deprecate the PCIM class PolicyRepository and its

three associations, and replace them with a new class ReusablePolicyContainer and new associations. As a separate change, the associations for ReusablePolicyContainer are being broadened, to allow a ReusablePolicyContainer to contain any reusable policy elements. In PCIM, the only associations defined for a PolicyRepository were for it to contain reusable policy conditions and policy actions.

### 3.2.2. Additional Associations and Additional Reusable Elements

The PolicyRuleInPolicyRule and PolicyGroupInPolicyRule aggregations have, in effect, been imported from QPIM. ("In effect" because these two aggregations, as well as PCIM's two aggregations PolicyGroupInPolicyGroup and PolicyRuleInPolicyGroup, are all being combined into a single aggregation PolicySetComponent.) These aggregations make it possible to define larger "chunks" of reusable policy to place in a ReusablePolicyContainer. These aggregations also introduce new semantics representing the contextual implications of having one PolicyRule executing within the scope of another PolicyRule.

### 3.2.3. Priorities and Decision Strategies

Drawing from both QPIM and ICPM, the Priority property has been deprecated in PolicyRule, and placed instead on the aggregation PolicySetComponent. The QPIM rules for resolving relative priorities across nested PolicyGroups and PolicyRules have been incorporated into PCIME as well. With the removal of the Priority property from PolicyRule, a new modeling dependency is introduced. In order to prioritize a PolicyRule/PolicyGroup relative to other PolicyRules/PolicyGroups, the elements being prioritized must all reside in one of three places: in a common PolicyGroup, in a common PolicyRule, or in a common System.

In the absence of any clear, general criterion for detecting policy conflicts, the PCIM restriction stating that priorities are relevant only in the case of conflicts is being removed. In its place, a PolicyDecisionStrategy property has been added to the PolicyGroup and PolicyRule classes. This property allows policy administrator to select one of two behaviors with respect to rule evaluation: either perform the actions for all PolicyRules whose conditions evaluate to TRUE, or perform the actions only for the highest-priority PolicyRule whose conditions evaluate to TRUE. (This is accomplished by placing the PolicyDecisionStrategy property in an abstract class PolicySet,

from which PolicyGroup and PolicyRule are derived.) The QPIM rules for applying decision strategies to a nested set of PolicyGroups and PolicyRules have also been imported.

#### 3.2.4. Policy Roles

The concept of policy roles is added to PolicyGroups (being present already in the PolicyRule class). This is accomplished via a new superclass for both PolicyRules and PolicyGroups - PolicySet. For nested PolicyRules and PolicyGroups, any roles associated with the outer rule or group are automatically "inherited" by the nested one. Additional roles may be added at the level of a nested rule or group.

It was also observed that there is no mechanism in PCIM for assigning roles to resources. For example, while it is possible in PCIM to associate a PolicyRule with the role "FrameRelay&&WAN", there is no way to indicate which interfaces match this criterion. A new PolicyRoleCollection class has been defined in PCIME, representing the collection of resources associated with a particular role. The linkage between a PolicyRule or PolicyGroup and a set of resources is then represented by an instance of PolicyRoleCollection. Equivalent values should be defined in the PolicyRoles property of PolicyRules and PolicyGroups, and in the PolicyRole property in PolicyRoleCollection.

#### 3.2.5. CompoundPolicyConditions and CompoundPolicyActions

The concept of a CompoundPolicyCondition has also been imported into PCIME from QPIM, and broadened to include a parallel CompoundPolicyAction. In both cases the idea is to create reusable "chunks" of policy that can exist as named elements in a ReusablePolicyContainer. The "Compound" classes and their associations incorporate the condition and action semantics that PCIM defined at the PolicyRule level: DNF/CNF for conditions, and ordering for actions.

Compound conditions and actions are defined to work with any component conditions and actions. In other words, while the components may be instances, respectively, of SimplePolicyCondition and SimplePolicyAction (discussed immediately below), they need not be.

### 3.2.6. Variables and Values

The SimplePolicyCondition / PolicyVariable / PolicyValue structure has been imported into PCIMe from QPIM. A list of PCIMe-level variables is defined, as well as a list of PCIMe-level values. Other variables and values may, if necessary, be defined in submodels of PCIMe. For example, QPIM defines a set of implicit variables corresponding to fields in RSVP flows.

A corresponding SimplePolicyAction / PolicyVariable / PolicyValue structure is also defined. While the semantics of a SimplePolicyCondition are "variable matches value", a SimplePolicyAction has the semantics "set variable to value".

### 3.2.7. Domain-Level Packet Filtering

For packet filtering specified at the domain level, a set of PolicyVariables and PolicyValues are defined, corresponding to the fields in an IP packet header plus the most common Layer 2 frame header fields. It is expected that domain-level policy conditions that filter on these header fields will be expressed in terms of CompoundPolicyConditions built up from SimplePolicyConditions that use these variables and values. An additional PolicyVariable, PacketDirection, is also defined, to indicate whether a packet being filtered is traveling inbound or outbound on an interface.

### 3.2.8. Device-Level Packet Filtering

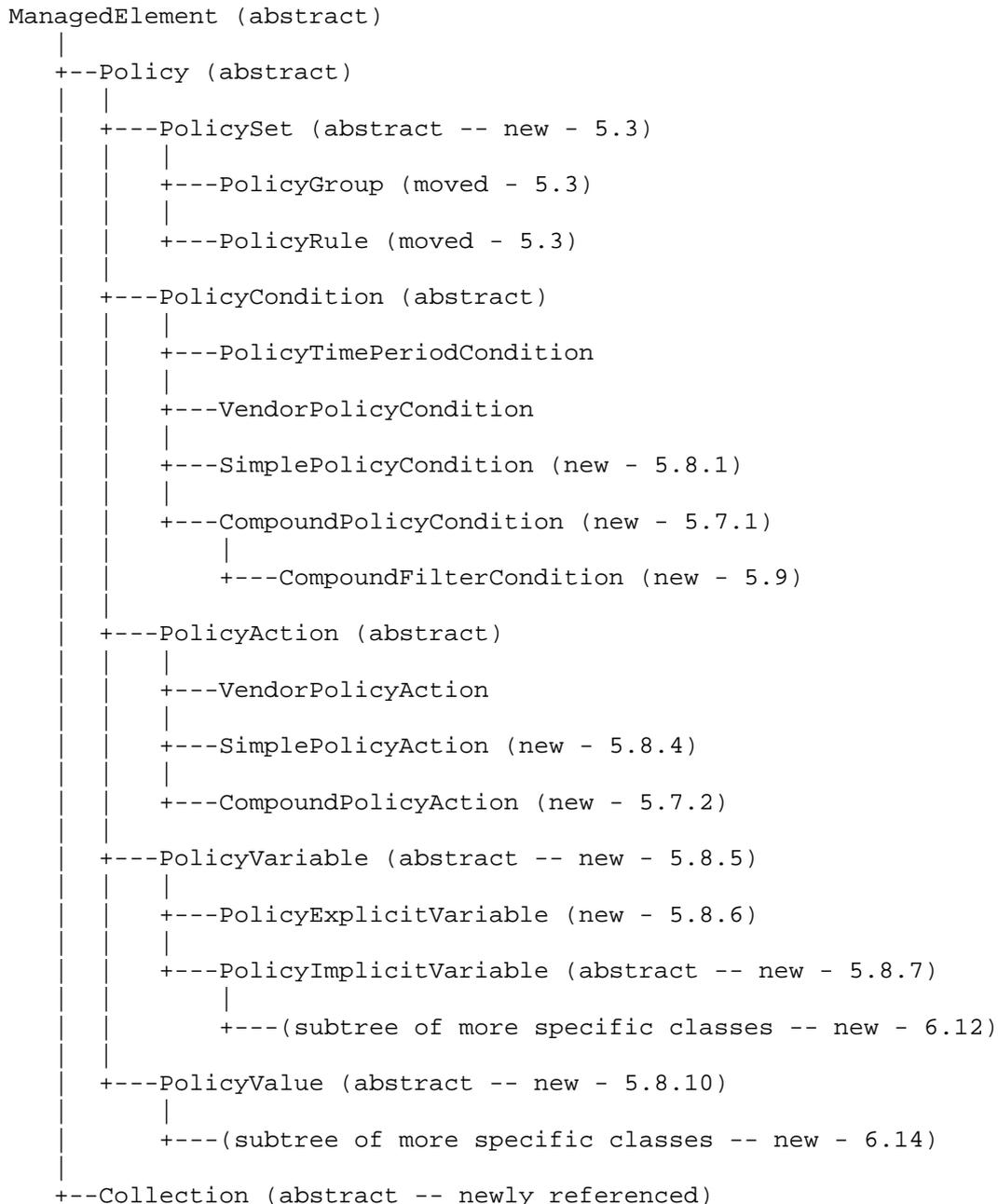
For packet filtering expressed at the device level, including the packet classifier filters modeled in QDDIM, the variables and values discussed in Section 3.2.7 need not be used. Filter classes derived from the CIM FilterEntryBase class hierarchy are available for use in these contexts. These latter classes have two important differences from the domain-level classes:

- o They support specification of filters for all of the fields in a particular protocol header in a single object instance. With the domain-level classes, separate instances are needed for each header field.
- o They provide native representations for the filter values, as opposed to the string representation used by the domain-level classes.

Device-level filter classes for the IP-related headers (IP, UDP, and TCP) and the 802 MAC headers are defined, respectively, in Sections 6.19 and 6.20.

#### 4. The Updated Class and Association Class Hierarchies

The following figure shows the class inheritance hierarchy for PCIMe. Changes from the PCIM hierarchy are noted parenthetically.



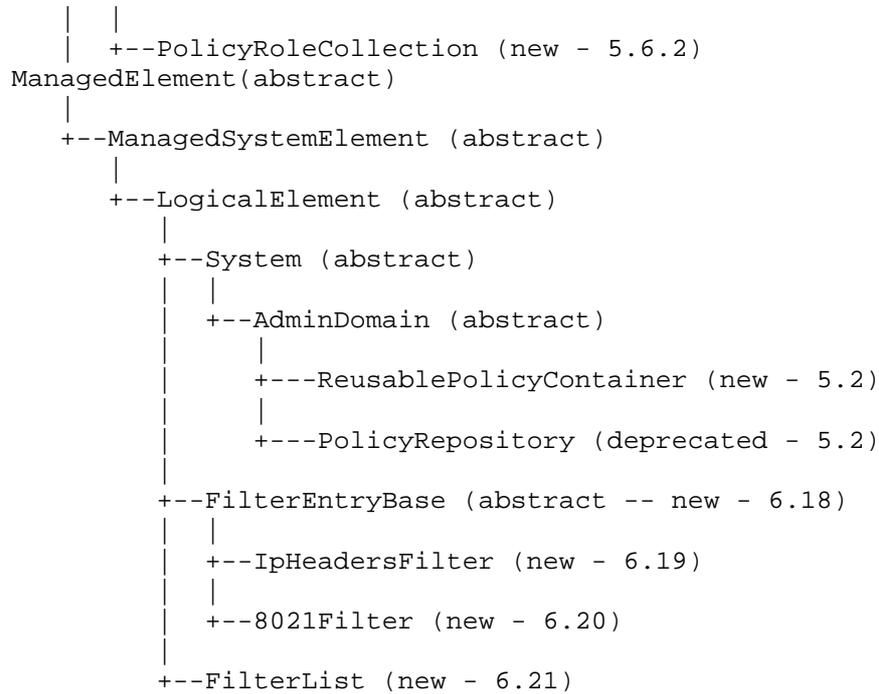


Figure 1. Class Inheritance Hierarchy for PCIME

The following figure shows the association class hierarchy for PCIMe. As before, changes from PCIM are noted parenthetically.



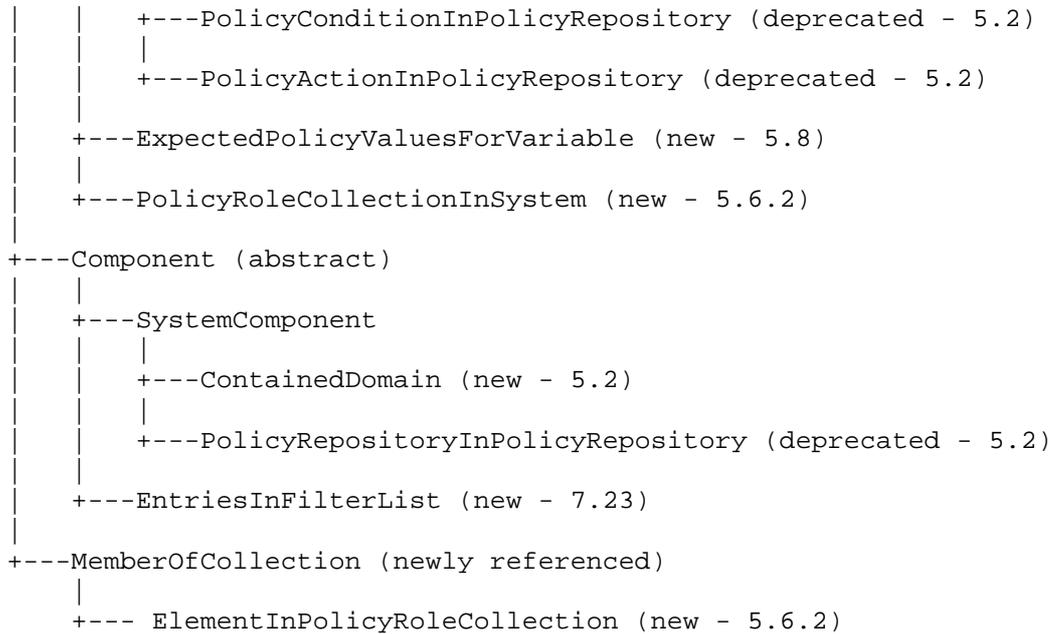


Figure 2. Association Class Inheritance Hierarchy for PCIME

In addition to these changes that show up at the class and association class level, there are other changes from PCIM involving individual class properties. In some cases new properties are introduced into existing classes, and in other cases existing properties are deprecated (without deprecating the classes that contain them).

## 5. Areas of Extension to PCIM

The following subsections describe each of the areas for which PCIM extensions are being defined.

### 5.1. Policy Scope

Policy scopes may be thought of in two dimensions: 1) the level of abstraction of the policy specification and 2) the applicability of policies to a set of managed resources.

#### 5.1.1. Levels of Abstraction: Domain- and Device-Level Policies

Policies vary in level of abstraction, from the business-level expression of service level agreements (SLAs) to the specification of a set of rules that apply to devices in a network. Those latter policies can, themselves, be classified into at least two groups:

those policies consumed by a Policy Decision Point (PDP) that specify the rules for an administrative and functional domain, and those policies consumed by a Policy Enforcement Point (PEP) that specify the device-specific rules for a functional domain. The higher-level rules consumed by a PDP, called domain-level policies, may have late binding variables unspecified, or specified by a classification, whereas the device-level rules are likely to have fewer unresolved bindings.

There is a relationship between these levels of policy specification that is out of scope for this standards effort, but that is necessary in the development and deployment of a usable policy-based configuration system. An SLA-level policy transformation to the domain-level policy may be thought of as analogous to a visual builder that takes human input and develops a programmatic rule specification. The relationship between the domain-level policy and the device-level policy may be thought of as analogous to that of a compiler and linkage editor that translates the rules into specific instructions that can be executed on a specific type of platform.

PCIM and PCIME may be used to specify rules at any and all of these levels of abstraction. However, at different levels of abstraction, different mechanisms may be more or less appropriate.

#### 5.1.2. Administrative and Functional Scopes

Administrative scopes for policy are represented in PCIM and in these extensions to PCIM as System subclass instances. Typically, a domain-level policy would be scoped by an AdminDomain instance (or by a hierarchy of AdminDomain instances) whereas a device-level policy might be scoped by a System instance that represents the PEP (e.g., an instance of ComputerSystem, see CIM [2]). In addition to collecting policies into an administrative domain, these System classes may also aggregate the resources to which the policies apply.

Functional scopes (sometimes referred to as functional domains) are generally defined by the submodels derived from PCIM and PCIME, and correspond to the service or services to which the policies apply. So, for example, Quality of Service may be thought of as a functional scope, or Diffserv and Intserv may each be thought of as functional scopes. These scoping decisions are represented by the structure of the submodels derived from PCIM and PCIME, and may be reflected in the number and types of PEP policy client(s), services, and the interaction between policies. Policies in different functional scopes are organized into disjoint sets of policy rules. Different functional domains may share some roles, some conditions, and even some actions. The rules from different functional domains may even be enforced at the same managed resource, but for the purposes of

policy evaluation they are separate. See section 5.5.3 for more information.

The functional scopes MAY be reflected in administrative scopes. That is, deployments of policy may have different administrative scopes for different functional scopes, but there is no requirement to do so.

## 5.2. Reusable Policy Elements

In PCIM, a distinction was drawn between reusable PolicyConditions and PolicyActions and rule-specific ones. The PolicyRepository class was also defined, to serve as a container for these reusable elements. The name "PolicyRepository" has proven to be an unfortunate choice for the class that serves as a container for reusable policy elements. This term is already used in documents like the Policy Framework, to denote the location from which the PDP retrieves all policy specifications, and into which the Policy Management Tool places all policy specifications. Consequently, the PolicyRepository class is being deprecated, in favor of a new class ReusablePolicyContainer.

When a class is deprecated, any associations that refer to it must also be deprecated. So replacements are needed for the two associations PolicyConditionInPolicyRepository and PolicyActionInPolicyRepository, as well as for the aggregation PolicyRepositoryInPolicyRepository. In addition to renaming the PolicyRepository class to ReusablePolicyContainer, however, PCIME is also broadening the types of policy elements that can be reusable. Consequently, rather than providing one-for-one replacements for the two associations, a single higher-level association ReusablePolicy is defined. This new association allows any policy element (that is, an instance of any subclass of the abstract class Policy) to be placed in a ReusablePolicyContainer.

Summarizing, the following changes in Sections 6 and 7 are the result of this item:

- o The class ReusablePolicyContainer is defined.
- o PCIM's PolicyRepository class is deprecated.
- o The association ReusablePolicy is defined.
- o PCIM's PolicyConditionInPolicyRepository association is deprecated.
- o PCIM's PolicyActionInPolicyRepository association is deprecated.
- o The aggregation ContainedDomain is defined.
- o PCIM's PolicyRepositoryInPolicyRepository aggregation is deprecated.

### 5.3. Policy Sets

A "policy" can be thought of as a coherent set of rules to administer, manage, and control access to network resources ("Policy Terminology", reference [10]). The structuring of these coherent sets of rules into subsets is enhanced in this document. In Section 5.4, we discuss the new options for the nesting of policy rules.

A new abstract class, `PolicySet`, is introduced to provide an abstraction for a set of rules. It is derived from `Policy`, and it is inserted into the inheritance hierarchy above both `PolicyGroup` and `PolicyRule`. This reflects the additional structural flexibility and semantic capability of both subclasses.

Two properties are defined in `PolicySet`: `PolicyDecisionStrategy` and `PolicyRoles`. The `PolicyDecisionStrategy` property is included in `PolicySet` to define the evaluation relationship among the rules in the policy set. See Section 5.5 for more information. The `PolicyRoles` property is included in `PolicySet` to characterize the resources to which the `PolicySet` applies. See Section 5.6 for more information.

Along with the definition of the `PolicySet` class, a new concrete aggregation class is defined that will also be discussed in the following sections. `PolicySetComponent` is defined as a subclass of `PolicyComponent`; it provides the containment relationship for a `PolicySet` in a `PolicySet`. `PolicySetComponent` replaces the two PCIM aggregations `PolicyGroupInPolicyGroup` and `PolicyRuleInPolicyGroup`, so these two aggregations are deprecated.

A `PolicySet`'s relationship to an `AdminDomain` or other administrative scoping system (for example, a `ComputerSystem`) is represented by the `PolicySetInSystem` abstract association. This new association is derived from `PolicyInSystem`, and the `PolicyGroupInSystem` and `PolicyRuleInSystem` associations are now derived from `PolicySetInSystem` instead of directly from `PolicyInSystem`. The `PolicySetInSystem.Priority` property is discussed in Section 5.5.3.

### 5.4. Nested Policy Rules

As previously discussed, policy is described by a set of policy rules that may be grouped into subsets. In this section we introduce the notion of nested rules, or the ability to define rules within rules. Nested rules are also called sub-rules, and we use both terms in this document interchangeably. The aggregation `PolicySetComponent` is used to represent the nesting of a policy rule in another policy rule.

#### 5.4.1. Usage Rules for Nested Rules

The relationship between rules and sub-rules is defined as follows:

- o The parent rule's condition clause is a condition for evaluation of all nested rules; that is, the conditions of the parent are logically ANDed to the conditions of the sub-rules. If the parent rule's condition clause evaluates to FALSE, sub-rules MAY be skipped since they also evaluate to FALSE.
- o If the parent rule's condition evaluates to TRUE, the set of sub-rules SHALL BE evaluated according to the decision strategy and priorities as discussed in Section 5.5.
- o If the parent rule's condition evaluates to TRUE, the parent rule's set of actions is executed BEFORE execution of the sub-rules actions. The parent rule's actions are not to be confused with default actions. A default action is one that is to be executed only if none of the more specific sub-rules are executed. If a default action needs to be specified, it needs to be defined as an action that is part of a catchall sub-rule associated with the parent rule. The association linking the default action(s) in this special sub-rule should have the lowest priority relative to all other sub-rule associations:

```

if parent-condition then parent rule's action
    if condA then actA
    if condB then ActB
    if True then default action
  
```

Such a default action functions as a default when FirstMatching decision strategies are in effect (see section 5.5). If AllMatching applies, the "default" action is always performed.

- o Policy rules have a context in which they are executed. The rule engine evaluates and applies the policy rules in the context of the managed resource(s) that are identified by the policy roles (or by an explicit association). Submodels MAY add additional context to policy rules based on rule structure; any such additional context is defined by the semantics of the action classes of the submodel.

#### 5.4.2. Motivation

Rule nesting enhances Policy readability, expressiveness and reusability. The ability to nest policy rules and form sub-rules is important for manageability and scalability, as it enables complex policy rules to be constructed from multiple simpler policy rules.

These enhancements ease the policy management tools' task, allowing policy rules to be expressed in a way closer to how humans think.

Although rule nesting can be used to suggest optimizations in the way policy rules are evaluated, as discussed in section 5.5.2 "Side Effects," nesting does not specify nor does it require any particular order of evaluation of conditions. Optimization of rule evaluation can be done in the PDP or in the PEP by dedicated code. This is similar to the relation between a high level programming language like C and machine code. An optimizer can create a more efficient machine code than any optimization done by the programmer within the source code. Nevertheless, if the PEP or PDP does not do optimization, the administrator writing the policy may be able to influence the evaluation of the policy rules for execution using rule nesting.

Nested rules are not designed for policy repository retrieval optimization. It is assumed that all rules and groups that are assigned to a role are retrieved by the PDP or PEP from the policy repository and enforced. Optimizing the number of rules retrieved should be done by clever selection of roles.

#### 5.5. Priorities and Decision Strategies

A "decision strategy" is used to specify the evaluation method for the policies in a PolicySet. Two decision strategies are defined: "FirstMatching" and "AllMatching." The FirstMatching strategy is used to cause the evaluation of the rules in a set such that the only actions enforced on a given examination of the PolicySet are those for the first rule (that is, the rule with the highest priority) that has its conditions evaluate to TRUE. The AllMatching strategy is used to cause the evaluation of all rules in a set; for all of the rules whose conditions evaluate to TRUE, the actions are enforced. Implementations MUST support the FirstMatching decision strategy; implementations MAY support the AllMatching decision strategy.

As previously discussed, the PolicySet subclasses are PolicyGroup and PolicyRule: either subclass may contain PolicySets of either subclass. Loops, including the degenerate case of a PolicySet that contains itself, are not allowed when PolicySets contain other PolicySets. The containment relationship is specified using the PolicySetComponent aggregation.

The relative priority within a PolicySet is established by the Priority property of the PolicySetComponent aggregation of the contained PolicyGroup and PolicyRule instances. The use of PCIM's PolicyRule.Priority property is deprecated in favor of this new property. The separation of the priority property from the rule has

two advantages. First, it generalizes the concept of priority, so that it can be used for both groups and rules. Second, it places the priority on the relationship between the parent policy set and the subordinate policy group or rule. The assignment of a priority value then becomes much easier, in that the value is used only in relationship to other priorities in the same set.

Together, the `PolicySet.PolicyDecisionStrategy` and `PolicySetComponent.Priority` determine the processing for the rules contained in a `PolicySet`. As before, the larger priority value represents the higher priority. Unlike the earlier definition, `PolicySetComponent.Priority` MUST have a unique value when compared with others defined for the same aggregating `PolicySet`. Thus, the evaluation of rules within a set is deterministically specified.

For a `FirstMatching` decision strategy, the first rule (that is, the one with the highest priority) in the set that evaluates to `True`, is the only rule whose actions are enforced for a particular evaluation pass through the `PolicySet`.

For an `AllMatching` decision strategy, all of the matching rules are enforced. The relative priority of the rules is used to determine the order in which the actions are to be executed by the enforcement point: the actions of the higher priority rules are executed first. Since the actions of higher priority rules are executed first, lower priority rules that also match may get the "last word," and thus produce a counter-intuitive result. So, for example, if two rules both evaluate to `True`, and the higher priority rule sets the DSCP to 3 and the lower priority rule sets the DSCP to 4, the action of the lower priority rule will be executed later and, therefore, will "win," in this example, setting the DSCP to 4. Thus, conflicts between rules are resolved by this execution order.

An implementation of the rule engine need not provide the action sequencing but the actions MUST be sequenced by the PEP or PDP on its behalf. So, for example, the rule engine may provide an ordered list of actions to be executed by the PEP and any required serialization is then provided by the service configured by the rule engine. See Section 5.5.2 for a discussion of side effects.

#### 5.5.1. Structuring Decision Strategies

As discussed in Sections 5.3 and 5.4, `PolicySet` instances may be nested arbitrarily. For a `FirstMatching` decision strategy on a `PolicySet`, any contained `PolicySet` that matches satisfies the termination criteria for the `FirstMatching` strategy. A `PolicySet` is considered to match if it is a `PolicyRule` and its conditions evaluate to `True`, or if the `PolicySet` is a `PolicyGroup` and at least one of its

contained PolicyGroups or PolicyRules match. The priority associated with contained PolicySets, then, determines when to terminate rule evaluation in the structured set of rules.

In the example shown in Figure 3, the relative priorities for the nested rules, high to low, are 1A, 1B1, 1X2, 1B3, 1C, 1C1, 1X2 and 1C3. (Note that PolicyRule 1X2 is included in both PolicyGroup 1B and PolicyRule 1C, but with different priorities.) Of course, which rules are enforced is also dependent on which rules, if any, match.

```

PolicyGroup 1: FirstMatching
|
+-- Pri=6 -- PolicyRule 1A
|
+-- Pri=5 -- PolicyGroup 1B: AllMatching
|
|           +-- Pri=5 -- PolicyGroup 1B1: AllMatching
|           |
|           |           +---- etc.
|           |
|           +-- Pri=4 -- PolicyRule 1X2
|           |
|           +-- Pri=3 -- PolicyRule 1B3: FirstMatching
|           |
|           |           +---- etc.
|
+-- Pri=4 -- PolicyRule 1C: FirstMatching
|
|           +-- Pri=4 -- PolicyRule 1C1
|           |
|           +-- Pri=3 -- PolicyRule 1X2
|           |
|           +-- Pri=2 -- PolicyRule 1C3

```

Figure 3. Nested PolicySets with Different Decision Strategies

- o Because PolicyGroup 1 has a FirstMatching decision strategy, if the conditions of PolicyRule 1A match, its actions are enforced and the evaluation stops.
- o If it does not match, PolicyGroup 1B is evaluated using an AllMatching strategy. Since PolicyGroup 1B1 also has an AllMatching strategy all of the rules and groups of rules contained in PolicyGroup 1B1 are evaluated and enforced as appropriate. PolicyRule 1X2 and PolicyRule 1B3 are also evaluated and enforced as appropriate. If any of the sub-rules in the

subtrees of PolicyGroup 1B evaluate to True, then PolicyRule 1C is not evaluated because the FirstMatching strategy of PolicyGroup 1 has been satisfied.

- o If neither PolicyRule 1A nor PolicyGroup 1B yield a match, then PolicyRule 1C is evaluated. Since it is first matching, rules 1C1, 1X2, and 1C3 are evaluated until the first match, if any.

#### 5.5.2. Side Effects

Although evaluation of conditions is sometimes discussed as an ordered set of operations, the rule engine need not be implemented as a procedural language interpreter. Any side effects of condition evaluation or the execution of actions MUST NOT affect the result of the evaluation of other conditions evaluated by the rule engine in the same evaluation pass. That is, an implementation of a rule engine MAY evaluate all conditions in any order before applying the priority and determining which actions are to be executed.

So, regardless of how a rule engine is implemented, it MUST NOT include any side effects of condition evaluation in the evaluation of conditions for either of the decision strategies. For both the AllMatching decision strategy and for the nesting of rules within rules (either directly or indirectly) where the actions of more than one rule may be enforced, any side effects of the enforcement of actions MUST NOT be included in condition evaluation on the same evaluation pass.

#### 5.5.3. Multiple PolicySet Trees For a Resource

As shown in the example in Figure 3., PolicySet trees are defined by the PolicySet subclass instances and the PolicySetComponent aggregation instances between them. Each PolicySet tree has a defined set of decision strategies and evaluation priorities. In section 5.6 we discuss some improvements in the use of PolicyRoles that cause the parent PolicySet.PolicyRoles to be applied to all contained PolicySet instances. However, a given resource may still have multiple, disjoint PolicySet trees regardless of how they are collected. These top-level PolicySet instances are called "unrooted" relative to the given resource.

So, a PolicySet instance is defined to be rooted or unrooted in the context of a particular managed element; the relationship to the managed element is usually established by the policy roles of the PolicySet instance and of the managed element (see 5.6 "Policy Roles"). A PolicySet instance is unrooted in that context if and only if there is no PolicySetComponent association to a parent PolicySet that is also related to the same managed element. These

PolicySetComponent aggregations are traversed up the tree without regard to how a PolicySet instance came to be related with the ManagedElement. Figure 4. shows an example where instance A has role A, instance B has role B and so on. In this example, in the context of interface X, instances B, and C are unrooted and instances D, E, and F are all rooted. In the context of interface Y, instance A is unrooted and instances B, C, D, E and F are all rooted.

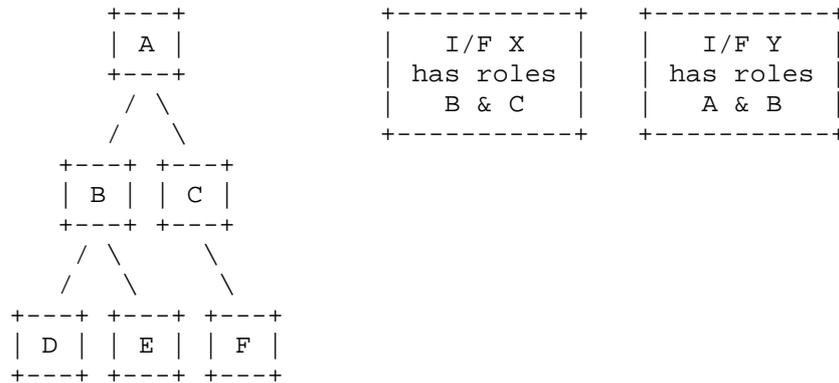


Figure 4. Unrooted PolicySet Instances

For those cases where there are multiple unrooted PolicySet instances that apply to the same managed resource (i.e., not in a common PolicySetComponent tree), the decision strategy among these disjoint PolicySet instances is the FirstMatching strategy. The priority used with this FirstMatching strategy is defined in the PolicySetInSystem association. The PolicySetInSystem subclass instances are present for all PolicySet instances (it is a required association) but the priority is only used as a default for unrooted PolicySet instances in a given ManagedElement context.

The FirstMatching strategy is used among all unrooted PolicySet instances that apply to a given resource for a given functional domain. So, for example, the PolicySet instances that are used for QoS policy and the instances that are used for IKE policy, although they are disjoint, are not joined in a FirstMatching decision strategy. Instead, they are evaluated independently of one another.

#### 5.5.4. Deterministic Decisions

As previously discussed, PolicySetComponent.Priority values MUST be unique within a containing PolicySet and PolicySetInSystem.Priority values MUST be unique for an associated System. Each PolicySet, then, has a deterministic behavior based upon the decision strategy and uniquely defined priority.

There are certainly cases where rules need not have a unique priority value (i.e., where evaluation and execution priority is not important). However, it is believed that the flexibility gained by this capability is not sufficiently beneficial to justify the possible variations in implementation behavior and the resulting confusion that might occur.

## 5.6. Policy Roles

A policy role is defined in [10] as "an administratively specified characteristic of a managed element (for example, an interface). It is a selector for policy rules and Provisioning Classes (PRCs), to determine the applicability of the rule/PRC to a particular managed element."

In PCIME, PolicyRoles is defined as a property of PolicySet, which is inherited by both PolicyRules and PolicyGroups. In this document, we also add PolicyRole as the identifying name of a collection of resources (PolicyRoleCollection), where each element in the collection has the specified role characteristic.

### 5.6.1. Comparison of Roles in PCIM with Roles in snmpconf

In the Configuration Management with SNMP (snmpconf) working group's Policy Based Management MIB [14], policy rules are of the form

```
if <policyFilter> then <policyAction>
```

where <policyFilter> is a set of conditions that are used to determine whether or not the policy applies to an object instance. The policy filter can perform comparison operations on SNMP variables already defined in MIBS (e.g., "ifType == ethernet").

The policy management MIB defined in [14] defines a Role table that enables one to associate Roles with elements, where roles have the same semantics as in PCIM. Then, since the policyFilter in a policy allows one to define conditions based on the comparison of the values of SNMP variables, one can filter elements based on their roles as defined in the Role group.

This approach differs from that adopted in PCIM in the following ways. First, in PCIM, a set of role(s) is associated with a policy rule as the values of the PolicyRoles property of a policy rule. The semantics of role(s) are then expected to be implemented by the PDP (i.e., policies are applied to the elements with the appropriate roles). In [14], however, no special processing is required for

realizing the semantics of roles; roles are treated just as any other SNMP variables and comparisons of role values can be included in the policy filter of a policy rule.

Secondly, in PCIM, there is no formally defined way of associating a role with an object instance, whereas in [14] this is done via the use of the Role tables (pmRoleESTable and pmRoleSETable). The Role tables associate Role values with elements.

#### 5.6.2. Addition of PolicyRoleCollection to PCIME

In order to remedy the latter shortcoming in PCIM (the lack of a way of associating a role with an object instance), PCIME has a new class PolicyRoleCollection derived from the CIM Collection class. Resources that share a common role are aggregated by a PolicyRoleCollection instance, via the ElementInPolicyRoleCollection aggregation. The role is specified in the PolicyRole property of the aggregating PolicyRoleCollection instance.

A PolicyRoleCollection always exists in the context of a system. As was done in PCIM for PolicyRules and PolicyGroups, an association, PolicyRoleCollectionInSystem, captures this relationship. Remember that in CIM, System is a base class for describing network devices and administrative domains.

The association between a PolicyRoleCollection and a system should be consistent with the associations that scope the policy rules/groups that are applied to the resources in that collection. Specifically, a PolicyRoleCollection should be associated with the same System as the applicable PolicyRules and/or PolicyGroups, or to a System higher in the tree formed by the SystemComponent association. When a PEP belongs to multiple Systems (i.e., AdminDomains), and scoping by a single domain is impractical, two alternatives exist. One is to arbitrarily limit domain membership to one System/AdminDomain. The other option is to define a more global AdminDomain that simply includes the others, and/or that spans the business or enterprise.

As an example, suppose that there are 20 traffic trunks in a network, and that an administrator would like to assign three of them to provide "gold" service. Also, the administrator has defined several policy rules which specify how the "gold" service is delivered. For these rules, the PolicyRoles property (inherited from PolicySet) is set to "Gold Service".

In order to associate three traffic trunks with "gold" service, an instance of the PolicyRoleCollection class is created and its PolicyRole property is also set to "Gold Service". Following this, the administrator associates three traffic trunks with the new

instance of PolicyRoleCollection via the ElementInPolicyRoleCollection aggregation. This enables a PDP to determine that the "Gold Service" policy rules apply to the three aggregated traffic trunks.

Note that roles are used to optimize policy retrieval. It is not mandatory to implement roles or, if they have been implemented, to group elements in a PolicyRoleCollection. However, if roles are used, then either the collection approach should be implemented, or elements should be capable of reporting their "pre-programmed" roles (as is done in COPS).

#### 5.6.3. Roles for PolicyGroups

In PCIM, role(s) are only associated with policy rules. However, it may be desirable to associate role(s) with groups of policy rules. For example, a network administrator may want to define a group of rules that apply only to Ethernet interfaces. A policy group can be defined with a role-combination="Ethernet", and all the relevant policy rules can be placed in this policy group. (Note that in PCIME, role(s) are made available to PolicyGroups as well as to PolicyRules by moving PCIM's PolicyRoles property up from PolicyRule to the new abstract class PolicySet. The property is then inherited by both PolicyGroup and PolicyRule.) Then every policy rule in this policy group implicitly inherits this role-combination from the containing policy group. A similar implicit inheritance applies to nested policy groups.

There is no explicit copying of role(s) from container to contained entity. Obviously, this implicit inheritance of role(s) leads to the possibility of defining inconsistent role(s) (as explained in the example below); the handling of such inconsistencies is beyond the scope of PCIME.

As an example, suppose that there is a PolicyGroup PG1 that contains three PolicyRules, PR1, PR2, and PR3. Assume that PG1 has the roles "Ethernet" and "Fast". Also, assume that the contained policy rules have the role(s) shown below:

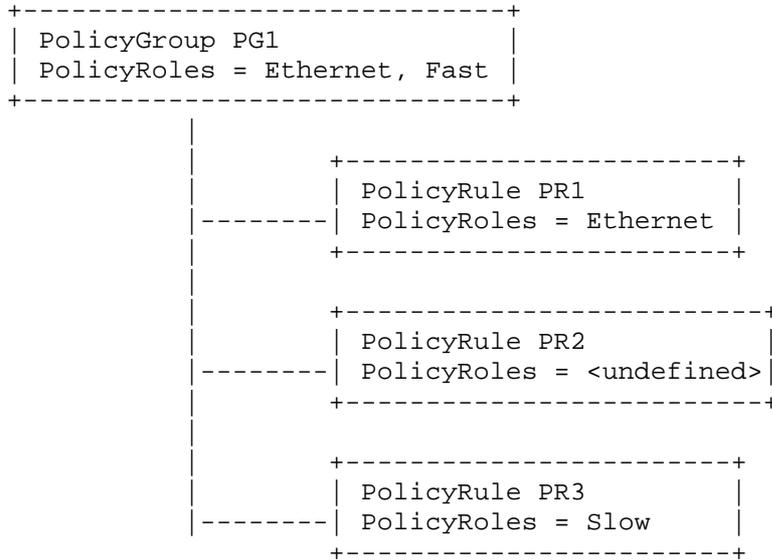


Figure 5. Inheritance of Roles

In this example, the PolicyRoles property value for PR1 is consistent with the value in PG1, and in fact, did not need to be redefined. The value of PolicyRoles for PR2 is undefined. Its roles are implicitly inherited from PG1. Lastly, the value of PolicyRoles for PR3 is "Slow". This appears to be in conflict with the role, "Fast," defined in PG1. However, whether these roles are actually in conflict is not clear. In one scenario, the policy administrator may have wanted only "Fast"- "Ethernet" rules in the policy group. In another scenario, the administrator may be indicating that PR3 applies to all "Ethernet" interfaces regardless of whether they are "Fast" or "Slow." Only in the former scenario (only "Fast"- "Ethernet" rules in the policy group) is there a role conflict.

Note that it is possible to override implicitly inherited roles via appropriate conditions on a PolicyRule. For example, suppose that PR3 above had defined the following conditions:

```
(interface is not "Fast") and (interface is "Slow")
```

This results in unambiguous semantics for PR3.

## 5.7. Compound Policy Conditions and Compound Policy Actions

Compound policy conditions and compound policy actions are introduced to provide additional reusable "chunks" of policy.

### 5.7.1. Compound Policy Conditions

A `CompoundPolicyCondition` is a `PolicyCondition` representing a Boolean combination of simpler conditions. The conditions being combined may be `SimplePolicyConditions` (discussed below in Section 6.4), but the utility of reusable combinations of policy conditions is not necessarily limited to the case where the component conditions are simple ones.

The PCIM extensions to introduce compound policy conditions are relatively straightforward. Since the purpose of the extension is to apply the DNF / CNF logic from PCIM's `PolicyConditionInPolicyRule` aggregation to a compound condition that aggregates simpler conditions, the following changes are required:

- o Create a new aggregation `PolicyConditionInPolicyCondition`, with the same `GroupNumber` and `ConditionNegated` properties as `PolicyConditionInPolicyRule`. The cleanest way to do this is to move the properties up to a new abstract aggregation superclass `PolicyConditionStructure`, from which the existing aggregation `PolicyConditionInPolicyRule` and a new aggregation `PolicyConditionInPolicyCondition` are derived. For now there is no need to re-document the properties themselves, since they are already documented in PCIM as part of the definition of the `PolicyConditionInPolicyRule` aggregation.
- o It is also necessary to define a concrete subclass `CompoundPolicyCondition` of `PolicyCondition`, to introduce the `ConditionListType` property. This property has the same function, and works in exactly the same way, as the corresponding property currently defined in PCIM for the `PolicyRule` class.

The class and property definitions for representing compound policy conditions are below, in Section 6.

### 5.7.2. Compound Policy Actions

A compound action is a convenient construct to represent a sequence of actions to be applied as a single atomic action within a policy rule. In many cases, actions are related to each other and should be looked upon as sub-actions of one "logical" action. An example of such a logical action is "shape & mark" (i.e., shape a certain stream to a set of predefined bandwidth characteristics and then mark these

packets with a certain DSCP value). This logical action is actually composed of two different QoS actions, which should be performed in a well-defined order and as a complete set.

The `CompoundPolicyAction` construct allows one to create a logical relationship between a number of actions, and to define the activation logic associated with this logical action.

The `CompoundPolicyAction` construct allows the reusability of these complex actions, by storing them in a `ReusablePolicyContainer` and reusing them in different policy rules. Note that a compound action may also be aggregated by another compound action.

As was the case with `CompoundPolicyCondition`, the PCIM extensions to introduce compound policy actions are relatively straightforward. This time the goal is to apply the property `ActionOrder` from PCIM's `PolicyActionInPolicyRule` aggregation to a compound action that aggregates simpler actions. The following changes are required:

- o Create a new aggregation `PolicyActionInPolicyAction`, with the same `ActionOrder` property as `PolicyActionInPolicyRule`. The cleanest way to do this is to move the property up to a new abstract aggregation superclass `PolicyActionStructure`, from which the existing aggregation `PolicyActionInPolicyRule` and a new aggregation `PolicyActionInPolicyAction` are derived.
- o It is also necessary to define a concrete subclass `CompoundPolicyAction` of `PolicyAction`, to introduce the `SequencedActions` property. This property has the same function, and works in exactly the same way, as the corresponding property currently defined in PCIM for the `PolicyRule` class.
- o Finally, a new property `ExecutionStrategy` is needed for both the PCIM class `PolicyRule` and the new class `CompoundPolicyAction`. This property allows the policy administrator to specify how the PEP should behave in the case where there are multiple actions aggregated by a `PolicyRule` or by a `CompoundPolicyAction`.

The class and property definitions for representing compound policy actions are below, in Section 6.

## 5.8. Variables and Values

The following subsections introduce several related concepts, including `PolicyVariables` and `PolicyValues` (and their numerous subclasses), `SimplePolicyConditions`, and `SimplePolicyActions`.

### 5.8.1. Simple Policy Conditions

The SimplePolicyCondition class models elementary Boolean expressions of the form: "(`<variable>` MATCH `<value>`)". The relationship 'MATCH', which is implicit in the model, is interpreted based on the variable and the value. Section 5.8.3 explains the semantics of the 'MATCH' operator. Arbitrarily complex Boolean expressions can be formed by chaining together any number of simple conditions using relational operators. Individual simple conditions can be negated as well. Arbitrarily complex Boolean expressions are modeled by the class CompoundPolicyCondition (described in Section 5.7.1).

For example, the expression "SourcePort == 80" can be modeled by a simple condition. In this example, 'SourcePort' is a variable, '==' is the relational operator denoting the equality relationship (which is generalized by PCIMe to a "MATCH" relationship), and '80' is an integer value. The complete interpretation of a simple condition depends on the binding of the variable. Section 5.8.5 describes variables and their binding rules.

The SimplePolicyCondition class refines the basic structure of the PolicyCondition class defined in PCIM by using the pair (`<variable>`, `<value>`) to form the condition. Note that the operator between the variable and the value is always implied in PCIMe: it is not a part of the formal notation.

The variable specifies the attribute of an object that should be matched when evaluating the condition. For example, for a QoS model, this object could represent the flow that is being conditioned. A set of predefined variables that cover network attributes commonly used for filtering is introduced in PCIMe, to encourage interoperability. This list covers layer 3 IP attributes such as IP network addresses, protocols and ports, as well as a set of layer 2 attributes (e.g., MAC addresses).

The bound variable is matched against a value to produce the Boolean result. For example, in the condition "The source IP address of the flow belongs to the 10.1.x.x subnet", a source IP address variable is matched against a 10.1.x.x subnet value.

### 5.8.2. Using Simple Policy Conditions

Simple conditions can be used in policy rules directly, or as building blocks for creating compound policy conditions.

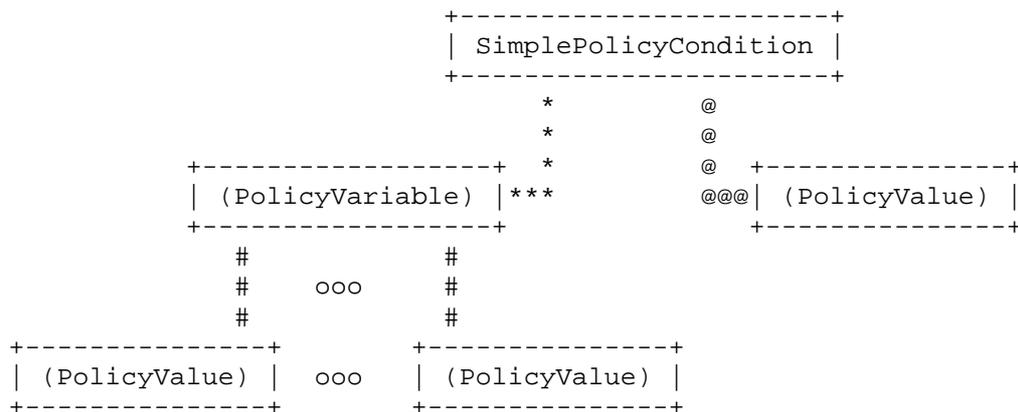
Simple condition composition MUST enforce the following data-type conformance rule: The ValueTypes property of the variable must be compatible with the type of the value class used. The simplest (and

friendliest, from a user point-of-view) way to do this is to equate the type of the value class with the name of the class. By ensuring that the ValueTypes property of the variable matches the name of the value class used, we know that the variable and value instance values are compatible with each other.

Composing a simple condition requires that an instance of the class SimplePolicyCondition be created, and that instances of the variable and value classes that it uses also exist. Note that the variable and/or value instances may already exist as reusable objects in an appropriate ReusablePolicyContainer.

Two aggregations are used in order to create the pair (<variable>, <value>). The aggregation PolicyVariableInSimplePolicyCondition relates a SimplePolicyCondition to a single variable instance. Similarly, the aggregation PolicyValueInSimplePolicyCondition relates a SimplePolicyCondition to a single value instance. Both aggregations are defined in this document.

Figure 6. depicts a SimplePolicyCondition with its associated variable and value. Also shown are two PolicyValue instances that identify the values that the variable can assume.



Aggregation Legend:  
 \*\*\*\* PolicyVariableInSimplePolicyCondition  
 @@@@ PolicyValueInSimplePolicyCondition  
 #### ExpectedPolicyValuesForVariable

Figure 6. SimplePolicyCondition

Note: The class names in parenthesis denote subclasses. The classes named in the figure are abstract, and thus cannot themselves be instantiated.

### 5.8.3. The Simple Condition Operator

A simple condition models an elementary Boolean expression of the form "variable MATCHes value". However, the formal notation of the SimplePolicyCondition, together with its associations, models only a pair, (<variable>, <value>). The 'MATCH' operator is not directly modeled -- it is implied. Furthermore, this implied 'MATCH' operator carries overloaded semantics.

For example, in the simple condition "DestinationPort MATCH '80'", the interpretation of the 'MATCH' operator is equality (the 'equal' operator). Clearly, a different interpretation is needed in the following cases:

- o "DestinationPort MATCH {'80', '8080'}" -- operator is 'IS SET MEMBER'
- o "DestinationPort MATCH {'1 to 255'}" -- operator is 'IN INTEGER RANGE'
- o "SourceIPAddress MATCH 'MyCompany.com'" -- operator is 'IP ADDRESS AS RESOLVED BY DNS'

The examples above illustrate the implicit, context-dependent nature of the 'MATCH' operator. The interpretation depends on the actual variable and value instances in the simple condition. The interpretation is always derived from the bound variable and the value instance associated with the simple condition. Text accompanying the value class and implicit variable definition is used for interpreting the semantics of the 'MATCH' relationship. In the following list, we define generic (type-independent) matching.

PolicyValues may be multi-fielded, where each field may contain a range of values. The same equally holds for PolicyVariables. Basically, we have to deal with single values (singleton), ranges ([lower bound .. upper bound]), and sets (a,b,c). So independent of the variable and value type, the following set of generic matching rules for the 'MATCH' operator are defined.

- o singleton matches singleton -> the matching rule is defined in the type
- o singleton matches range [lower bound .. upper bound] -> the matching evaluates to true, if the singleton matches the lower bound or the upper bound or a value in between

- o singleton matches set -> the matching evaluates to true, if the value of the singleton matches one of the components in the set, where a component may be a singleton or range again
- o ranges [A..B] matches singleton -> is true if A matches B matches singleton
- o range [A..B] matches range [X..Y] -> the matching evaluates to true, if all values of the range [A..B] are also in the range [X..Y]. For instance, [3..5] match [1..6] evaluates to true, whereas [3..5] match [4..6] evaluates to false.
- o range [A..B] matches set (a,b,c, ...) -> the matching evaluates to true, if all values in the range [A..B] are part of the set. For instance, range [2..3] match set ([1..2],3) evaluates to true, as well as range [2..3] match set (2,3), and range [2..3] match set ([1..2],[3..5]).
- o set (a,b,c, ...) match singleton -> is true if a match b match c match ... match singleton
- o set match range -> the matching evaluates to true, if all values in the set are part of the range. For example, set (2,3) match range [1..4] evaluates to true.
- o set (a,b,c,...) match set (x,y,z,...) -> the matching evaluates to true, if all values in the set (a,b,c,...) are part of the set (x,y,z,...). For example, set (1,2,3) match set (1,2,3,4) evaluates to true. Set (1,2,3) match set (1,2) evaluates to false.

Variables may contain various types (Section 6.11.1). When not stated otherwise, the type of the value bound to the variable at condition evaluation time and the value type of the PolicyValue instance need to be of the same type. If they differ, then the condition evaluates to FALSE.

The ExpectedPolicyValuesForVariable association specifies an expected set of values that can be matched with a variable within a simple condition. Using this association, a source or destination port can be limited to the range 0-200, a source or destination IP address can be limited to a specified list of IPv4 address values, etc.

```

+-----+
| SimplePolicyCondition |
+-----+
          *                @
          *                @
          *                @
+-----+
| Name=SmallSourcePorts | | Name=Port300 |
| Class=PolicySourcePortVariable | | Class=PolicyIntegerValue |
| ValueTypes=[PolicyIntegerValue] | | IntegerList = [300] |
+-----+
          #
          #
          #
+-----+
| Name=SmallPortsValues |
| Class=PolicyIntegerValue |
| IntegerList=[1..200] |
+-----+

```

Aggregation Legend:

```

**** PolicyVariableInSimplePolicyCondition
@@@ PolicyValueInSimplePolicyCondition
#### ExpectedPolicyValuesForVariable

```

Figure 7. An Invalid SimplePolicyCondition

The ability to express these limitations appears in the model to support validation of a SimplePolicyCondition prior to its deployment to an enforcement point. A Policy Management Tool, for example SHOULD NOT accept the SimplePolicyCondition shown in Figure 7. If, however, a policy rule containing this condition does appear at an enforcement point, the expected values play no role in the determination of whether the condition evaluates to True or False. Thus in this example, the SimplePolicyCondition evaluates to True if the source port for the packet under consideration is 300, and it evaluates to False otherwise.

#### 5.8.4. SimplePolicyActions

The SimplePolicyAction class models the elementary set operation. "SET <variable> TO <value>". The set operator MUST overwrite an old value of the variable. In the case where the variable to be updated is multi-valued, the only update operation defined is a complete replacement of all previous values with a new set. In other words, there are no Add or Remove [to/from the set of values] operations defined for SimplePolicyActions.

For example, the action "set DSCP to EF" can be modeled by a simple action. In this example, 'DSCP' is an implicit variable referring to the IP packet header DSCP field. 'EF' is an integer or bit string value (6 bits). The complete interpretation of a simple action depends on the binding of the variable.

The SimplePolicyAction class refines the basic structure of the PolicyAction class defined in PCIM, by specifying the contents of the action using the (<variable>, <value>) pair to form the action. The variable specifies the attribute of an object. The value of this attribute is set to the value specified in <value>. Selection of the object is a function of the type of variable involved. See Sections 5.8.6 and 5.8.7, respectively, for details on object selection for explicitly bound and implicitly bound policy variables.

SimplePolicyActions can be used in policy rules directly, or as building blocks for creating CompoundPolicyActions.

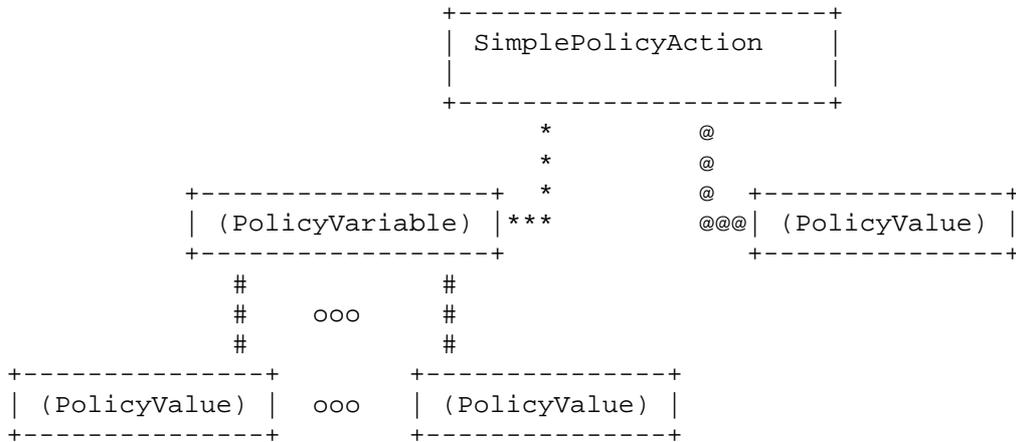
The set operation is only valid if the list of types of the variable (ValueTypes property of PolicyImplicitVariable) includes the specified type of the value. Conversion of values from one representation into another is not defined. For example, a variable of IPv4Address type may not be set to a string containing a DNS name. Conversions are part of an implementation-specific mapping of the model.

As was the case with SimplePolicyConditions, the role of expected values for the variables that appear in SimplePolicyActions is for validation, prior to the time when an action is executed. Expected values play no role in action execution.

Composing a simple action requires that an instance of the class SimplePolicyAction be created, and that instances of the variable and value classes that it uses also exist. Note that the variable and/or value instances may already exist as reusable objects in an appropriate ReusablePolicyContainer.

Two aggregations are used in order to create the pair (<variable>, <value>). The aggregation PolicyVariableInSimplePolicyAction relates a SimplePolicyAction to a single variable instance. Similarly, the aggregation PolicyValueInSimplePolicyAction relates a SimplePolicyAction to a single value instance. Both aggregations are defined in this document.

Figure 8. depicts a SimplePolicyAction with its associated variable and value.



## Aggregation Legend:

```

**** PolicyVariableInSimplePolicyAction
@@@ PolicyValueInSimplePolicyAction
### ExpectedPolicyValuesForVariable

```

Figure 8. SimplePolicyAction

## 5.8.5. Policy Variables

A variable generically represents information that changes (or "varies"), and that is set or evaluated by software. In policy, conditions and actions can abstract information as "policy variables" to be evaluated in logical expressions, or set by actions.

PCIME defines two types of PolicyVariables, PolicyImplicitVariables and PolicyExplicitVariables. The semantic difference between these classes is based on modeling context. Explicit variables are bound to exact model constructs, while implicit variables are defined and evaluated outside of a model. For example, one can imagine a PolicyCondition testing whether a CIM ManagedSystemElement's Status property has the value "Error." The Status property is an explicitly defined PolicyVariable (i.e., it is defined in the context of the CIM Schema, and evaluated in the context of a specific instance). On the other hand, network packets are not explicitly modeled or instantiated, since there is no perceived value (at this time) in managing at the packet level. Therefore, a PolicyCondition can make no explicit reference to a model construct that represents a network packet's source address. In this case, an implicit PolicyVariable is defined, to allow evaluation or modification of a packet's source address.

### 5.8.6. Explicitly Bound Policy Variables

Explicitly bound policy variables indicate the class and property names of the model construct to be evaluated or set. The CIM Schema defines and constrains "appropriate" values for the variable (i.e., model property) using data types and other information such as class/property qualifiers.

A PolicyExplicitVariable is "explicit" because its model semantics are exactly defined. It is NOT explicit due to an exact binding to a particular object instance. If PolicyExplicitVariables were tied to instances (either via associations or by an object identification property in the class itself), then we would be forcing element-specific rules. On the other hand, if we only specify the object's model context (class and property name), but leave the binding to the policy framework (for example, using policy roles), then greater flexibility results for either general or element-specific rules.

For example, an element-specific rule is obtained by a condition ((<variable>, <value>) pair) that defines CIM LogicalDevice DeviceID="12345". Alternately, if a PolicyRule's PolicyRoles is "edge device" and the condition ((<variable>, <value>) pair) is Status="Error", then a general rule results for all edge devices in error.

Currently, the only binding for a PolicyExplicitVariable defined in PCIME is to the instances selected by policy roles. For each such instance, a SimplePolicyCondition that aggregates the PolicyExplicitVariable evaluates to True if and only if ALL of the following are true:

- o The instance selected is of the class identified by the variable's ModelClass property, or of a subclass of this class.
- o The instance selected has the property identified by the variable's ModelProperty property.
- o The value of this property in the instance matches the value specified in the PolicyValue aggregated by the condition.

In all other cases, the SimplePolicyCondition evaluates to False.

For the case where a SimplePolicyAction aggregates a PolicyExplicitVariable, the indicated property in the selected instance is set to the value represented by the PolicyValue that the SimplePolicyAction also aggregates. However, if the selected instance is not of the class identified by the variable's ModelClass property, or of a subclass of this class, then the action is not performed. In this case the SimplePolicyAction is not treated either as a successfully executed action (for the execution strategy Do

Until Success) or as a failed action (for the execution strategy Do Until Failure). Instead, the remaining actions for the policy rule, if any, are executed as if this SimplePolicyAction were not present at all in the list of actions aggregated by the rule.

Explicit variables would be more powerful if they could reach beyond the instances selected by policy roles, to related instances. However, to represent a policy rule involving such variables in any kind of general way requires something that starts to resemble very much a complete policy language. Clearly such a language is outside the scope of PCIME, although it might be the subject of a future document.

By restricting much of the generality, it would be possible for explicit variables in PCIME to reach slightly beyond a selected instance. For example, if a selected instance were related to exactly one instance of another class via a particular association class, and if the goal of the policy rule were both to test a property of this related instance and to set a property of that same instance, then it would be possible to represent the condition and action of the rule using PolicyExplicitVariables. Rather than handling this one specific case with explicit variables, though, it was decided to lump them with the more general case, and deal with them if and when a policy language is defined.

Refer to Section 6.10 for the formal definition of the class PolicyExplicitVariable.

#### 5.8.7. Implicitly Bound Policy Variables

Implicitly bound policy variables define the data type and semantics of a variable. This determines how the variable is bound to a value in a condition or an action. Further instructions are provided for specifying data type and/or value constraints for implicitly bound variables.

PCIME introduces an abstract class, PolicyImplicitVariable, to model implicitly bound variables. This class is derived from the abstract class PolicyVariable also defined in PCIME. Each of the implicitly bound variables introduced by PCIME (and those that are introduced by domain-specific sub-models) MUST be derived from the PolicyImplicitVariable class. The rationale for using this mechanism for modeling is explained below in Section 5.8.9.

A domain-specific policy information model that extends PCIME may define additional implicitly bound variables either by deriving them directly from the class PolicyImplicitVariable, or by further

refining an existing variable class such as SourcePort. When refining a class such as SourcePort, existing binding rules, type or value constraints may be narrowed.

#### 5.8.8. Structure and Usage of Pre-Defined Variables

A class derived from PolicyImplicitVariable to model a particular implicitly bound variable SHOULD be constructed so that its name depicts the meaning of the variable. For example, a class defined to model the source port of a TCP/UDP flow SHOULD have 'SourcePort' in its name.

PCIME defines one association and one general-purpose mechanism that together characterize each of the implicitly bound variables that it introduces:

1. The ExpectedPolicyValuesForVariable association defines the set of value classes that could be matched to this variable.
2. The list of constraints on the values that the PolicyVariable can hold (i.e., values that the variable must match) are defined by the appropriate properties of an associated PolicyValue class.

In the example presented above, a PolicyImplicitVariable represents the SourcePort of incoming traffic. The ValueTypes property of an instance of this class will hold the class name PolicyIntegerValue. This by itself constrains the data type of the SourcePort instance to be an integer. However, we can further constrain the particular values that the SourcePort variable can hold by entering valid ranges in the IntegerList property of the PolicyIntegerValue instance (0 - 65535 in this document).

The combination of the VariableName and the ExpectedPolicyValuesForVariable association provide a consistent and extensible set of metadata that define the semantics of variables that are used to form policy conditions. Since the ExpectedPolicyValuesForVariable association points to a PolicyValue instance, any of the values expressible in the PolicyValue class can be used to constrain values that the PolicyImplicitVariable can hold. For example:

- o The ValueTypes property can be used to ensure that only proper classes are used in the expression. For example, the SourcePort variable will not be allowed to ever be of type PolicyIPv4AddrValue, since source ports have different semantics than IP addresses and may not be matched. However, integer value types are allowed as the property ValueTypes holds the string "PolicyIntegerValue", which is the class name for integer values.

- o The ExpectedPolicyValuesForVariable association also ensures that variable-specific semantics are enforced (e.g., the SourcePort variable may include a constraint association to a value object defining a specific integer range that should be matched).

#### 5.8.9. Rationale for Modeling Implicit Variables as Classes

An implicitly bound variable can be modeled in one of several ways, including a single class with an enumerator for each individual implicitly bound variable and an abstract class extended for each individual variable. The reasons for using a class inheritance mechanism for specifying individual implicitly bound variables are these:

1. It is easy to extend. A domain-specific information model can easily extend the PolicyImplicitVariable class or its subclasses to define domain-specific and context-specific variables. For example, a domain-specific QoS policy information model may introduce an implicitly bound variable class to model applications by deriving a qosApplicationVariable class from the PolicyImplicitVariable abstract class.
2. Introduction of a single structural class for implicitly bound variables would have to include an enumerator property that contains all possible individual implicitly bound variables. This means that a domain-specific information model wishing to introduce an implicitly bound variable must extend the enumerator itself. This results in multiple definitions of the same class, differing in the values available in the enumerator class. One definition, in this document, would include the common implicitly bound variables' names, while a second definition, in the domain-specific information model document, may include additional values ('qosApplicationVariable' in the example above). It wouldn't even be obvious to the application developer that multiple class definitions existed. It would be harder still for the application developer to actually find the correct class to use.
3. In addition, an enumerator-based definition would require each additional value to be registered with IANA to ascertain adherence to standards. This would make the process cumbersome.
4. A possible argument against the inheritance mechanism would cite the fact that this approach results in an explosion of class definitions compared to an enumerator class, which only introduces a single class. While, by itself, this is not a strike against the approach, it may be argued that data models derived from this information model may be more difficult to optimize for applications. This argument is rejected on the grounds that

application optimization is of lesser value for an information model than clarity and ease of extension. In addition, it is hard to claim that the inheritance model places an absolute burden on the optimization. For example, a data model may still use enumeration to denote instances of pre-defined variables and claim PCIMe compliance, as long as the data model can be mapped correctly to the definitions specified in this document.

#### 5.8.10. Policy Values

The abstract class `PolicyValue` is used for modeling values and constants used in policy conditions. Different value types are derived from this class, to represent the various attributes required. Extensions of the abstract class `PolicyValue`, defined in this document, provide a list of values for basic network attributes. Values can be used to represent constants as named values. Named values can be kept in a reusable policy container to be reused by multiple conditions. Examples of constants include well-known ports, well-known protocols, server addresses, and other similar concepts.

The `PolicyValue` subclasses define three basic types of values: scalars, ranges and sets. For example, a well-known port number could be defined using the `PolicyIntegerValue` class, defining a single value (80 for HTTP), a range (80-88), or a set (80, 82, 8080) of ports, respectively. For details, please see the class definition for each value type in Section 6.14 of this document.

PCIMe defines the following subclasses of the abstract class `PolicyValue`:

Classes for general use:

- `PolicyStringValue`,
- `PolicyIntegerValue`,
- `PolicyBitStringValue`
- `PolicyBooleanValue`.

Classes for layer 3 Network values:

- `PolicyIPv4AddrValue`,
- `PolicyIPv6AddrValue`.

Classes for layer 2 Network values:

- `PolicyMACAddrValue`.

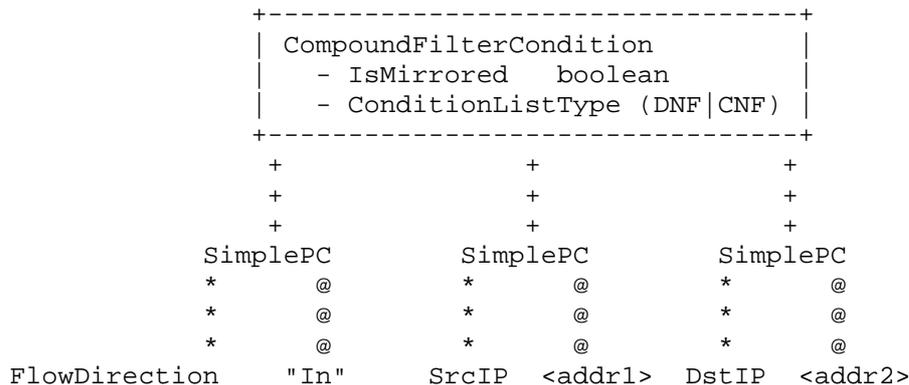
For details, please see the class definition section of each class in Section 6.14 of this document.

## 5.9. Packet Filtering

PCIME contains two mechanisms for representing packet filters. The more general of these, termed here the domain-level model, expresses packet filters in terms of policy variables and policy values. The other mechanism, termed here the device-level model, expresses packet filters in a way that maps more directly to the packet fields to which the filters are being applied. While it is possible to map between these two representations of packet filters, no mapping is provided in PCIME itself.

### 5.9.1. Domain-Level Packet Filters

In addition to filling in the holes in the overall Policy infrastructure, PCIME proposes a single mechanism for expressing domain-level packet filters in policy conditions. This is being done in response to concerns that even though the initial "wave" of submodels derived from PCIM were all filtering on IP packets, each was doing it in a slightly different way. PCIME proposes a common way to express IP packet filters. The following figure illustrates how packet-filtering conditions are expressed in PCIME.



Aggregation Legend:

```

++++ PolicyConditionInPolicyCondition
**** PolicyVariableInSimplePolicyCondition
@@@ PolicyValueInSimplePolicyCondition

```

Figure 9. Packet Filtering in Policy Conditions

In Figure 9., each SimplePolicyCondition represents a single field to be filtered on: Source IP address, Destination IP address, Source port, etc. An additional SimplePolicyCondition indicates the direction that a packet is traveling on an interface: inbound or outbound. Because of the FlowDirection condition, care must be taken

in aggregating a set of SimplePolicyConditions into a CompoundFilterCondition. Otherwise, the resulting CompoundPolicyCondition may match all inbound packets, or all outbound packets, when this is probably not what was intended.

Individual SimplePolicyConditions may be negated when they are aggregated by a CompoundFilterCondition.

CompoundFilterCondition is a subclass of CompoundPolicyCondition. It introduces one additional property, the Boolean property IsMirrored. The purpose of this property is to allow a single CompoundFilterCondition to match packets traveling in both directions on a higher-level connection such as a TCP session. When this property is TRUE, additional packets match a filter, beyond those that would ordinarily match it. An example will illustrate how this property works.

Suppose we have a CompoundFilterCondition that aggregates the following three filters, which are ANDed together:

- o FlowDirection = "In"
- o Source IP = 9.1.1.1
- o Source Port = 80

Regardless of whether IsMirrored is TRUE or FALSE, inbound packets will match this CompoundFilterCondition if their Source IP address = 9.1.1.1 and their Source port = 80. If IsMirrored is TRUE, however, an outbound packet will also match the CompoundFilterCondition if its Destination IP address = 9.1.1.1 and its Destination port = 80.

IsMirrored "flips" the following Source/Destination packet header fields:

- o FlowDirection "In" / FlowDirection "Out"
- o Source IP address / Destination IP address
- o Source port / Destination port
- o Source MAC address / Destination MAC address
- o Source [layer-2] SAP / Destination [layer-2] SAP.

#### 5.9.2. Device-Level Packet Filters

At the device level, packet header filters are represented by two subclasses of the abstract class FilterEntryBase: IpHeadersFilter and 8021Filter. Submodels of PCIME may define other subclasses of FilterEntryBase in addition to these two; ICPM [12], for example, defines subclasses for IPsec-specific filters.

Instances of the subclasses of `FilterEntryBase` are not used directly as filters. They are always aggregated into a `FilterList`, by the aggregation `EntriesInFilterList`. For `PCIMe` and its submodels, the `EntrySequence` property in this aggregation always takes its default value '0', indicating that the aggregated filter entries are ANDed together.

The `FilterList` class includes an enumeration property `Direction`, representing the direction of the traffic flow to which the `FilterList` is to be applied. The value `Mirrored(4)` for `Direction` represents exactly the same thing as the `IsMirrored` boolean does in `CompoundFilterCondition`. See Section 5.9.1 for details.

#### 5.10. Conformance to PCIM and PCIME

Because `PCIM` and `PCIME` provide the core classes for modeling policies, they are not in general sufficient by themselves for representing actual policy rules. Submodels, such as `QPIM` and `ICPM`, provide the means for expressing policy rules, by defining subclasses of the classes defined in `PCIM` and `PCIME`, and/or by indicating how the `PolicyVariables` and `PolicyValues` defined in `PCIME` can be used to express conditions and actions applicable to the submodel.

A particular submodel will not, in general, need to use every element defined in `PCIM` and `PCIME`. For the elements it does not use, a submodel SHOULD remain silent on whether its implementations must support the element, must not support the element, should support the element, etc. For the elements it does use, a submodel SHOULD indicate which elements its implementations must support, which elements they should support, and which elements they may support.

`PCIM` and `PCIME` themselves simply define elements that may be of use to submodels. These documents remain silent on whether implementations are required to support an element, should support it, etc.

This model (and derived submodels) defines conditions and actions that are used by policy rules. While the conditions and actions defined herein are straightforward and may be presumed to be widely supported, as submodels are developed it is likely that situations will arise in which specific conditions or actions are not supported by some part of the policy execution system. Similarly, situations may also occur where rules contain syntactic or semantic errors.

It should be understood that the behavior and effect of undefined or incorrectly defined conditions or actions is not prescribed by this information model. While it would be helpful if it were prescribed, the variations in implementation restrict the ability for this

information model to control the effect. For example, if an implementation only detected that a PEP could not enforce a given action on that PEP, it would be very difficult to declare that such a failure should affect other PEPs, or the PDP process. On the other hand, if the PDP determines that it cannot properly evaluate a condition, that failure may well affect all applications of the containing rules.

## 6. Class Definitions

The following definitions supplement those in PCIM itself. PCIM definitions that are not DEPRECATED here are still current parts of the overall Policy Core Information Model.

### 6.1. The Abstract Class "PolicySet"

PolicySet is an abstract class that may group policies into a structured set of policies.

NAME	PolicySet
DESCRIPTION	An abstract class that represents a set of policies that form a coherent set. The set of contained policies has a common decision strategy and a common set of policy roles. Subclasses include PolicyGroup and PolicyRule.
DERIVED FROM	Policy
ABSTRACT	TRUE
PROPERTIES	PolicyDecisionStrategy PolicyRoles

The PolicyDecisionStrategy property specifies the evaluation method for policy groups and rules contained within the policy set.

NAME	PolicyDecisionStrategy
DESCRIPTION	The evaluation method used for policies contained in the PolicySet. FirstMatching enforces the actions of the first rule that evaluates to TRUE; All Matching enforces the actions of all rules that evaluate to TRUE.
SYNTAX	uint16
VALUES	1 [FirstMatching], 2 [AllMatching]
DEFAULT VALUE	1 [FirstMatching]

The definition of PolicyRoles is unchanged from PCIM. It is, however, moved from the class Policy up to the superclass PolicySet.

## 6.2. Update PCIM's Class "PolicyGroup"

The PolicyGroup class is moved, so that it is now derived from PolicySet.

NAME	PolicyGroup
DESCRIPTION	A container for a set of related PolicyRules and PolicyGroups.
DERIVED FROM	PolicySet
ABSTRACT	FALSE
PROPERTIES	(none)

## 6.3. Update PCIM's Class "PolicyRule"

The PolicyRule class is moved, so that it is now derived from PolicySet. The Priority property is also deprecated in PolicyRule, and PolicyRoles is now inherited from the parent class PolicySet. Finally, a new property ExecutionStrategy is introduced, paralleling the property of the same name in the class CompoundPolicyAction.

NAME	PolicyRule
DESCRIPTION	The central class for representing the "If Condition then Action" semantics associated with a policy rule.
DERIVED FROM	PolicySet
ABSTRACT	FALSE
PROPERTIES	Enabled ConditionListType RuleUsage Priority DEPRECATED FOR PolicySetComponent.Priority AND FOR PolicySetInSystem.Priority Mandatory SequencedActions ExecutionStrategy

The property ExecutionStrategy defines the execution strategy to be used upon the sequenced actions aggregated by this PolicyRule. (An equivalent ExecutionStrategy property is also defined for the CompoundPolicyAction class, to provide the same indication for the sequenced actions aggregated by a CompoundPolicyAction.) This document defines three execution strategies:

Do Until Success - execute actions according to predefined order, until successful execution of a single action.

Do All - execute ALL actions which are part of the modeled set, according to their predefined order. Continue doing this, even if one or more of the actions fails.

Do Until Failure - execute actions according to predefined order, until the first failure in execution of a single sub-action.

The property definition is as follows:

NAME	ExecutionStrategy
DESCRIPTION	An enumeration indicating how to interpret the action ordering for the actions aggregated by this PolicyRule.
SYNTAX	uint16 (ENUM, {1=Do Until Success, 2=Do All, 3=Do Until Failure} )
DEFAULT VALUE	Do All (2)

#### 6.4. The Class "SimplePolicyCondition"

A simple policy condition is composed of an ordered triplet:

<Variable> MATCH <Value>

No formal modeling of the MATCH operator is provided. The 'match' relationship is implied. Such simple conditions are evaluated by answering the question:

Does <variable> match <value>?

The 'match' relationship is to be interpreted by analyzing the variable and value instances associated with the simple condition.

Simple conditions are building blocks for more complex Boolean Conditions, modeled by the CompoundPolicyCondition class.

The SimplePolicyCondition class is derived from the PolicyCondition class defined in PCIM.

A variable and a value must be associated with a simple condition to make it a meaningful condition, using, respectively, the aggregations PolicyVariableInSimplePolicyCondition and PolicyValueInSimplePolicyCondition.

The class definition is as follows:

NAME	SimplePolicyCondition
DERIVED FROM	PolicyCondition
ABSTRACT	False
PROPERTIES	(none)

### 6.5. The Class "CompoundPolicyCondition"

This class represents a compound policy condition, formed by aggregation of simpler policy conditions.

NAME	CompoundPolicyCondition
DESCRIPTION	A subclass of PolicyCondition that introduces the ConditionListType property, used for assigning DNF / CNF semantics to subordinate policy conditions.
DERIVED FROM	PolicyCondition
ABSTRACT	FALSE
PROPERTIES	ConditionListType

The ConditionListType property is used to specify whether the list of policy conditions associated with this compound policy condition is in disjunctive normal form (DNF) or conjunctive normal form (CNF). If this property is not present, the list type defaults to DNF. The property definition is as follows:

NAME	ConditionListType
DESCRIPTION	Indicates whether the list of policy conditions associated with this policy rule is in disjunctive normal form (DNF) or conjunctive normal form (CNF).
SYNTAX	uint16
VALUES	DNF(1), CNF(2)
DEFAULT VALUE	DNF(1)

### 6.6. The Class "CompoundFilterCondition"

This subclass of CompoundPolicyCondition introduces one additional property, the boolean IsMirrored. This property turns on or off the "flipping" of corresponding source and destination fields in a filter specification.

NAME	CompoundFilterCondition
DESCRIPTION	A subclass of CompoundPolicyCondition that introduces the IsMirrored property.
DERIVED FROM	CompoundPolicyCondition
ABSTRACT	FALSE
PROPERTIES	IsMirrored

The IsMirrored property indicates whether packets that "mirror" a compound filter condition should be treated as matching the filter. The property definition is as follows:

NAME	IsMirrored
DESCRIPTION	Indicates whether packets that mirror the specified filter are to be treated as matching the filter.
SYNTAX	boolean
DEFAULT VALUE	FALSE

#### 6.7. The Class "SimplePolicyAction"

The SimplePolicyAction class models the elementary set operation. "SET <variable> TO <value>". The set operator MUST overwrite an old value of the variable.

Two aggregations are used in order to create the pair <variable> <value>. The aggregation PolicyVariableInSimplePolicyAction relates a SimplePolicyAction to a single variable instance. Similarly, the aggregation PolicyValueInSimplePolicyAction relates a SimplePolicyAction to a single value instance. Both aggregations are defined in this document.

NAME	SimplePolicyAction
DESCRIPTION	A subclass of PolicyAction that introduces the notion of "SET variable TO value".
DERIVED FROM	PolicyAction
ABSTRACT	FALSE
PROPERTIES	(none)

#### 6.8. The Class "CompoundPolicyAction"

The CompoundPolicyAction class is used to represent an expression consisting of an ordered sequence of action terms. Each action term is represented as a subclass of the PolicyAction class, defined in [PCIM]. Compound actions are constructed by associating dependent action terms together using the PolicyActionInPolicyAction aggregation.

The class definition is as follows:

NAME	CompoundPolicyAction
DESCRIPTION	A class for representing sequenced action terms. Each action term is defined to be a subclass of the PolicyAction class.
DERIVED FROM	PolicyAction
ABSTRACT	FALSE
PROPERTIES	SequencedActions ExecutionStrategy

This is a concrete class, and is therefore directly instantiable.

The Property SequencedActions is identical to the SequencedActions property defined in PCIM for the class PolicyRule.

The property ExecutionStrategy defines the execution strategy to be used upon the sequenced actions associated with this compound action. (An equivalent ExecutionStrategy property is also defined for the PolicyRule class, to provide the same indication for the sequenced actions associated with a PolicyRule.) This document defines three execution strategies:

- Do Until Success - execute actions according to predefined order, until successful execution of a single sub-action.
- Do All - execute ALL actions which are part of the modeled set, according to their predefined order. Continue doing this, even if one or more of the sub-actions fails.
- Do Until Failure - execute actions according to predefined order, until the first failure in execution of a single sub-action.

Since a CompoundPolicyAction may itself be aggregated either by a PolicyRule or by another CompoundPolicyAction, its success or failure will be an input to the aggregating entity's execution strategy. Consequently, the following rules are specified, for determining whether a CompoundPolicyAction succeeds or fails:

If the CompoundPolicyAction's ExecutionStrategy is Do Until Success, then:

- o If one component action succeeds, then the CompoundPolicyAction succeeds.
- o If all component actions fail, then the CompoundPolicyAction fails.

If the CompoundPolicyAction's ExecutionStrategy is Do All, then:

- o If all component actions succeed, then the CompoundPolicyAction succeeds.
- o If at least one component action fails, then the CompoundPolicyAction fails.

If the CompoundPolicyAction's ExecutionStrategy is Do Until Failure, then:

- o If all component actions succeed, then the CompoundPolicyAction succeeds.
- o If at least one component action fails, then the CompoundPolicyAction fails.

The definition of the ExecutionStrategy property is as follows:

NAME	ExecutionStrategy
DESCRIPTION	An enumeration indicating how to interpret the action ordering for the actions aggregated by this CompoundPolicyAction.
SYNTAX	uint16 (ENUM, {1=Do Until Success, 2=Do All, 3=Do Until Failure} )
DEFAULT VALUE	Do All (2)

#### 6.9. The Abstract Class "PolicyVariable"

Variables are used for building individual conditions. The variable specifies the property of a flow or an event that should be matched when evaluating the condition. However, not every combination of a variable and a value creates a meaningful condition. For example, a source IP address variable can not be matched against a value that specifies a port number. A given variable selects the set of matchable value types.

A variable can have constraints that limit the set of values within a particular value type that can be matched against it in a condition. For example, a source-port variable limits the set of values to represent integers to the range of 0-65535. Integers outside this range cannot be matched to the source-port variable, even though they are of the correct data type. Constraints for a given variable are indicated through the ExpectedPolicyValuesForVariable association.

The PolicyVariable is an abstract class. Implicit and explicit context variable classes are defined as sub classes of the PolicyVariable class. A set of implicit variables is defined in this document as well.

The class definition is as follows:

NAME	PolicyVariable
DERIVED FROM	Policy
ABSTRACT	TRUE
PROPERTIES	(none)

#### 6.10. The Class "PolicyExplicitVariable"

Explicitly defined policy variables are evaluated within the context of the CIM Schema and its modeling constructs. The PolicyExplicitVariable class indicates the exact model property to be evaluated or manipulated. See Section 5.8.6 for a complete discussion of what happens when the values of the ModelClass and

ModelProperty properties in an instance of this class do not correspond to the characteristics of the model construct being evaluated or updated.

The class definition is as follows:

NAME	PolicyExplicitVariable
DERIVED FROM	PolicyVariable
ABSTRACT	False
PROPERTIES	ModelClass, ModelProperty

#### 6.10.1. The Single-Valued Property "ModelClass"

This property is a string specifying the class name whose property is evaluated or set as a PolicyVariable.

The property is defined as follows:

NAME	ModelClass
SYNTAX	String

#### 6.10.2. The Single-Valued Property ModelProperty

This property is a string specifying the property name, within the ModelClass, which is evaluated or set as a PolicyVariable. The property is defined as follows:

NAME	ModelProperty
SYNTAX	String

#### 6.11. The Abstract Class "PolicyImplicitVariable"

Implicitly defined policy variables are evaluated outside of the context of the CIM Schema and its modeling constructs. Subclasses specify the data type and semantics of the PolicyVariables.

Interpretation and evaluation of a PolicyImplicitVariable can vary, depending on the particular context in which it is used. For example, a "SourceIP" address may denote the source address field of an IP packet header, or the sender address delivered by an RSVP PATH message.

The class definition is as follows:

NAME	PolicyImplicitVariable
DERIVED FROM	PolicyVariable
ABSTRACT	True
PROPERTIES	ValueTypes[ ]

### 6.11.1. The Multi-Valued Property "ValueTypes"

This property is a set of strings specifying an unordered list of possible value/data types that can be used in simple conditions and actions, with this variable. The value types are specified by their class names (subclasses of PolicyValue such as PolicyStringValue). The list of class names enables an application to search on a specific name, as well as to ensure that the data type of the variable is of the correct type.

The list of default ValueTypes for each subclass of PolicyImplicitVariable is specified within that variable's definition.

The property is defined as follows:

NAME	ValueTypes
SYNTAX	String

### 6.12. Subclasses of "PolicyImplicitVariable" Specified in PCIME

The following subclasses of PolicyImplicitVariable are defined in PCIME.

#### 6.12.1. The Class "PolicySourceIPv4Variable"

NAME	PolicySourceIPv4Variable
DESCRIPTION	The source IPv4 address. of the outermost IP packet header. "Outermost" here refers to the IP packet as it flows on the wire, before any headers have been stripped from it.
	ALLOWED VALUE TYPES:
	- PolicyIPv4AddrValue
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

#### 6.12.2. The Class "PolicySourceIPv6Variable"

NAME	PolicySourceIPv6Variable
DESCRIPTION	The source IPv6 address of the outermost IP packet header. "Outermost" here refers to the IP packet as it flows on the wire, before any headers have been stripped from it.

ALLOWED VALUE TYPES:  
 - PolicyIPv6AddrValue

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.3. The Class "PolicyDestinationIPv4Variable"

NAME PolicyDestinationIPv4Variable  
 DESCRIPTION The destination IPv4 address of the outermost IP packet header. "Outermost" here refers to the IP packet as it flows on the wire, before any headers have been stripped from it.

ALLOWED VALUE TYPES:  
 - PolicyIPv4AddrValue

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.4. The Class "PolicyDestinationIPv6Variable"

NAME PolicyDestinationIPv6Variable  
 DESCRIPTION The destination IPv6 address of the outermost IP packet header. "Outermost" here refers to the IP packet as it flows on the wire, before any headers have been stripped from it.

ALLOWED VALUE TYPES:  
 - PolicyIPv6AddrValue

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

## 6.12.5. The Class "PolicySourcePortVariable"

NAME	PolicySourcePortVariable
DESCRIPTION	Ports are defined as the abstraction that transport protocols use to distinguish among multiple destinations within a given host computer. For TCP and UDP flows, the PolicySourcePortVariable is logically bound to the source port field of the outermost UDP or TCP packet header. "Outermost" here refers to the IP packet as it flows on the wire, before any headers have been stripped from it. ALLOWED VALUE TYPES: - PolicyIntegerValue (0..65535)
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

## 6.12.6. The Class "PolicyDestinationPortVariable"

NAME	PolicyDestinationPortVariable
DESCRIPTION	Ports are defined as the abstraction that transport protocols use to distinguish among multiple destinations within a given host computer. For TCP and UDP flows, the PolicyDestinationPortVariable is logically bound to the destination port field of the outermost UDP or TCP packet header. "Outermost" here refers to the IP packet as it flows on the wire, before any headers have been stripped from it. ALLOWED VALUE TYPES: - PolicyIntegerValue (0..65535)
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

## 6.12.7. The Class "PolicyIPProtocolVariable"

NAME	PolicyIPProtocolVariable
DESCRIPTION	The IP protocol number. ALLOWED VALUE TYPES: - PolicyIntegerValue (0..255)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.8. The Class "PolicyIPVersionVariable"

NAME PolicyIPVersionVariable  
 DESCRIPTION The IP version number. The well-known values are 4 and 6.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..15)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.9. The Class "PolicyIPToSVariable"

NAME PolicyIPToSVariable  
 DESCRIPTION The IP TOS octet.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..255)  
 - PolicyBitStringValue (8 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.10. The Class "PolicyDSCPVariable"

NAME PolicyDSCPVariable  
 DESCRIPTION The 6 bit Differentiated Service Code Point.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..63)  
 - PolicyBitStringValue (6 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

## 6.12.11. The Class "PolicyFlowIdVariable"

NAME	PolicyFlowIdVariable
DESCRIPTION	The flow identifier of the outermost IPv6 packet header. "Outermost" here refers to the IP packet as it flows on the wire, before any headers have been stripped from it.
	ALLOWED VALUE TYPES:
	- PolicyIntegerValue (0..1048575)
	- PolicyBitStringValue (20 bits)
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

## 6.12.12. The Class "PolicySourceMACVariable"

NAME	PolicySourceMACVariable
DESCRIPTION	The source MAC address.
	ALLOWED VALUE TYPES:
	- PolicyMACAddrValue
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

## 6.12.13. The Class "PolicyDestinationMACVariable"

NAME	PolicyDestinationMACVariable
DESCRIPTION	The destination MAC address.
	ALLOWED VALUE TYPES:
	- PolicyMACAddrValue
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

## 6.12.14. The Class "PolicyVLANVariable"

NAME	PolicyVLANVariable
DESCRIPTION	The virtual Bridged Local Area Network Identifier, a 12-bit field as defined in the IEEE 802.1q standard.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..4095)  
 - PolicyBitStringValue (12 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.15. The Class "PolicyCoSVariable"

NAME PolicyCoSVariable  
 DESCRIPTION Class of Service, a 3-bit field, used in the layer 2 header to select the forwarding treatment. Bound to the IEEE 802.1q user-priority field.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..7)  
 - PolicyBitStringValue (3 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.16. The Class "PolicyEthertypeVariable"

NAME PolicyEthertypeVariable  
 DESCRIPTION The Ethertype protocol number of Ethernet frames.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..65535)  
 - PolicyBitStringValue (16 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.17. The Class "PolicySourceSAPVariable"

NAME PolicySourceSAPVariable  
 DESCRIPTION The Source Service Access Point (SAP) number of the IEEE 802.2 LLC header.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..255)  
 - PolicyBitStringValue (8 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.18. The Class "PolicyDestinationSAPVariable"

NAME PolicyDestinationSAPVariable  
 DESCRIPTION The Destination Service Access Point (SAP) number of the IEEE 802.2 LLC header.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..255)  
 - PolicyBitStringValue (8 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

#### 6.12.19. The Class "PolicySNAPOUIVariable"

NAME PolicySNAPOUIVariable  
 DESCRIPTION The value of the first three octets of the Sub-Network Access Protocol (SNAP) Protocol Identifier field for 802.2 SNAP encapsulation, containing an Organizationally Unique Identifier (OUI). The value 00-00-00 indicates the encapsulation of Ethernet frames (RFC 1042). OUI value 00-00-F8 indicates the special encapsulation of Ethernet frames by certain types of bridges (IEEE 802.1H). Other values are supported, but are not further defined here. These OUI values are to be interpreted according to the endian-notation conventions of IEEE 802. For either of the two Ethernet encapsulations, the remainder of the Protocol Identifier field is represented by the PolicySNAPTypeVariable.

ALLOWED VALUE TYPES:  
 - PolicyIntegerValue (0..16777215)  
 - PolicyBitStringValue (24 bits)

DERIVED FROM PolicyImplicitVariable  
 ABSTRACT FALSE  
 PROPERTIES (none)

## 6.12.20. The Class "PolicySNAPTypeVariable"

NAME	PolicySNAPTypeVariable
DESCRIPTION	The value of the 4th and 5th octets of the Sub-Network Access Protocol (SNAP) Protocol Identifier field for IEEE 802 SNAP encapsulation when the PolicySNAPPOUIVariable indicates one of the two Encapsulated Ethernet frame formats. This value is undefined for other values of PolicySNAPPOUIVariable.
	ALLOWED VALUE TYPES:
	- PolicyIntegerValue (0..65535)
	- PolicyBitStringValue (16 bits)
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

## 6.12.21. The Class "PolicyFlowDirectionVariable"

NAME	PolicyFlowDirectionVariable
DESCRIPTION	The direction of a flow relative to a network element. Direction may be "IN" and/or "OUT".
	ALLOWED VALUE TYPES:
	- PolicyStringValue ("IN", "OUT")
DERIVED FROM	PolicyImplicitVariable
ABSTRACT	FALSE
PROPERTIES	(none)

To match on both inbound and outbound flows, the associated PolicyStringValue object has two entries in its StringList property: "IN" and "OUT".

## 6.13. The Abstract Class "PolicyValue"

This is an abstract class that serves as the base class for all subclasses that are used to define value objects in the PCIMe. It is used for defining values and constants used in policy conditions. The class definition is as follows:

NAME	PolicyValue
DERIVED FROM	Policy
ABSTRACT	True
PROPERTIES	(none)

#### 6.14. Subclasses of "PolicyValue" Specified in PCIME

The following subsections contain the PolicyValue subclasses defined in PCIME. Additional subclasses may be defined in models derived from PCIME.

##### 6.14.1. The Class "PolicyIPv4AddrValue"

This class is used to provide a list of IPv4Addresses, hostnames and address range values to be matched against in a policy condition. The class definition is as follows:

NAME	PolicyIPv4AddrValue
DERIVED FROM	PolicyValue
ABSTRACT	False
PROPERTIES	IPv4AddrList[ ]

The IPv4AddrList property provides an unordered list of strings, each specifying a single IPv4 address, a hostname, or a range of IPv4 addresses, according to the ABNF definition [6] of an IPv4 address, as specified below:

```
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv4prefix  = IPv4address "/" 1*2DIGIT
IPv4range   = IPv4address "-" IPv4address
IPv4maskedaddress = IPv4address "," IPv4address
Hostname (as defined in [4])
```

In the above definition, each string entry is either:

1. A single IPv4address in dot notation, as defined above. Example: 121.1.1.2
2. An IPv4prefix address range, as defined above, specified by an address and a prefix length, separated by "/". Example: 2.3.128.0/15
3. An IPv4range address range defined above, specified by a starting address in dot notation and an ending address in dot notation, separated by "-". The range includes all addresses between the range's starting and ending addresses, including these two addresses. Example: 1.1.22.1-1.1.22.5
4. An IPv4maskedaddress address range, as defined above, specified by an address and mask. The address and mask are represented in dot notation, separated by a comma ",". The masked address appears before the comma, and the mask appears after the comma. Example: 2.3.128.0,255.255.248.0.

5. A single Hostname. The Hostname format follows the guidelines and restrictions specified in [4]. Example: www.bigcompany.com.

Conditions matching IPv4AddrValues evaluate to true according to the generic matching rules. Additionally, a hostname is matched against another valid IPv4address representation by resolving the hostname into an IPv4 address first, and then comparing the addresses afterwards. Matching hostnames against each other is done using a string comparison of the two names.

The property definition is as follows:

NAME	IPv4AddrList
SYNTAX	String
FORMAT	IPv4address   IPv4prefix   IPv4range   IPv4maskedaddress   hostname

#### 6.14.2. The Class "PolicyIPv6AddrValue"

This class is used to define a list of IPv6 addresses, hostnames, and address range values. The class definition is as follows:

NAME	PolicyIPv6AddrValue
DERIVED FROM	PolicyValue
ABSTRACT	False
PROPERTIES	IPv6AddrList[ ]

The property IPv6AddrList provides an unordered list of strings, each specifying an IPv6 address, a hostname, or a range of IPv6 addresses. IPv6 address format definition uses the standard address format defined in [7]. The ABNF definition [6] as specified in [7] is:

```

IPv6address = hexpart [ ":" IPv4address ]
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv6prefix  = hexpart "/" 1*2DIGIT
hexpart    = hexseq | hexseq "::" [ hexseq ] | "::" [ hexseq ]
hexseq     = hex4 *( ":" hex4)
hex4      = 1*4HEXDIG
IPv6range  = IPv6address "-" IPv6address
IPv6maskedaddress = IPv6address ", " IPv6address
Hostname (as defines in [NAMES])

```

Each string entry is either:

1. A single IPv6address as defined above.
2. A single Hostname. Hostname format follows guidelines and restrictions specified in [4].

3. An IPv6range address range, specified by a starting address in dot notation and an ending address in dot notation, separated by "-". The range includes all addresses between the range's starting and ending addresses, including these two addresses.
4. An IPv4maskedaddress address range defined above specified by an address and mask. The address and mask are represented in dot notation separated by a comma ",".
5. A single IPv6prefix as defined above.

Conditions matching IPv6AddrValues evaluate to true according to the generic matching rules. Additionally, a hostname is matched against another valid IPv6address representation by resolving the hostname into an IPv6 address first, and then comparing the addresses afterwards. Matching hostnames against each other is done using a string comparison of the two names.

#### 6.14.3. The Class "PolicyMACAddrValue"

This class is used to define a list of MAC addresses and MAC address range values. The class definition is as follows:

```
NAME                PolicyMACAddrValue
DERIVED FROM        PolicyValue
ABSTRACT            False
PROPERTIES          MACAddrList[ ]
```

The property MACAddrList provides an unordered list of strings, each specifying a MAC address or a range of MAC addresses. The 802 MAC address canonical format is used. The ABNF definition [6] is:

```
MACaddress = 1*4HEXDIG ":" 1*4HEXDIG ":" 1*4HEXDIG
MACmaskedaddress = MACaddress", "MACaddress
```

Each string entry is either:

1. A single MAC address. Example: 0000:00A5:0000
2. A MACmaskedaddress address range defined specified by an address and mask. The mask specifies the relevant bits in the address. Example: 0000:00A5:0000,FFFF:FFFF:0000 defines a range of MAC addresses in which the first four octets are equal to 0000:00A5.

The property definition is as follows:

NAME	MACAddrList
SYNTAX	String
FORMAT	MACaddress   MACmaskedaddress

#### 6.14.4. The Class "PolicyStringValue"

This class is used to represent a single string value, or a set of string values. Each value can have wildcards. The class definition is as follows:

NAME	PolicyStringValue
DERIVED FROM	PolicyValue
ABSTRACT	False
PROPERTIES	StringList[ ]

The property StringList provides an unordered list of strings, each representing a single string with wildcards. The asterisk character "\*" is used as a wildcard, and represents an arbitrary substring replacement. For example, the value "abc\*def" matches the string "abcxyzdef", and the value "abc\*def\*" matches the string "abcxxxdefyyyzzz". The syntax definition is identical to the substring assertion syntax defined in [5]. If the asterisk character is required as part of the string value itself, it MUST be quoted as described in Section 4.3 of [5].

The property definition is as follows:

NAME	StringList
SYNTAX	String

#### 6.14.5. The Class "PolicyBitStringValue"

This class is used to represent a single bit string value, or a set of bit string values. The class definition is as follows:

NAME	PolicyBitStringValue
DERIVED FROM	PolicyValue
ABSTRACT	False
PROPERTIES	BitStringList[ ]

The property BitStringList provides an unordered list of strings, each representing a single bit string or a set of bit strings. The number of bits specified SHOULD equal the number of bits of the expected variable. For example, for a one-octet variable, 8 bits

should be specified. If the variable does not have a fixed length, the bit string should be matched against the variable's most significant bit string. The formal definition of a bit string is:

```
binary-digit = "0" / "1"
bitString = 1*binary-digit
maskedBitString = bitString", "bitString
```

Each string entry is either:

1. A single bit string. Example: 00111010
2. A range of bit strings specified using a bit string and a bit mask. The bit string and mask fields have the same number of bits specified. The mask bit string specifies the significant bits in the bit string value. For example, 110110, 100110 and 110111 would match the maskedBitString 100110,101110 but 100100 would not.

The property definition is as follows:

NAME	BitStringList
SYNTAX	String
FORMAT	bitString   maskedBitString

#### 6.14.6. The Class "PolicyIntegerValue"

This class provides a list of integer and integer range values. Integers of arbitrary sizes can be represented. The class definition is as follows:

NAME	PolicyIntegerValue
DERIVED FROM	PolicyValue
ABSTRACT	False
PROPERTIES	IntegerList[ ]

The property IntegerList provides an unordered list of integers and integer range values, represented as strings. The format of this property takes one of the following forms:

1. An integer value.
2. A range of integers. The range is specified by a starting integer and an ending integer, separated by '..'. The starting integer MUST be less than or equal to the ending integer. The range includes all integers between the starting and ending integers, including these two integers.

To represent a range of integers that is not bounded, the reserved words `-INFINITY` and/or `INFINITY` can be used in place of the starting and ending integers. In addition to ordinary integer matches, `INFINITY` matches `INFINITY` and `-INFINITY` matches `-INFINITY`.

The ABNF definition [6] is:

```
integer = [-]1*DIGIT | "INFINITY" | "-INFINITY"
integerrange = integer"..integer
```

Using ranges, the operators `greater-than`, `greater-than-or-equal-to`, `less-than`, and `less-than-or-equal-to` can be expressed. For example, "X is- greater-than 5" (where X is an integer) can be translated to "X matches 6-INFINITY". This enables the match condition semantics of the operator for the `SimplePolicyCondition` class to be kept simple (i.e., just the value "match").

The property definition is as follows:

NAME	IntegerList
SYNTAX	String
FORMAT	integer   integerrange

#### 6.14.7. The Class "PolicyBooleanValue"

This class is used to represent a Boolean (TRUE/FALSE) value. The class definition is as follows:

NAME	PolicyBooleanValue
DERIVED FROM	PolicyValue
ABSTRACT	False
PROPERTIES	BooleanValue

The property definition is as follows:

NAME	BooleanValue
SYNTAX	boolean

#### 6.15. The Class "PolicyRoleCollection"

This class represents a collection of managed elements that share a common role. The `PolicyRoleCollection` always exists in the context of a system, specified using the `PolicyRoleCollectionInSystem` association. The value of the `PolicyRole` property in this class specifies the role, and can be matched with the value(s) in the `PolicyRoles` array in `PolicyRules` and `PolicyGroups`. `ManagedElements` that share the role defined in this collection are aggregated into the collection via the association `ElementInPolicyRoleCollection`.

NAME	PolicyRoleCollection
DESCRIPTION	A subclass of the CIM Collection class used to group together managed elements that share a role.
DERIVED FROM	Collection
ABSTRACT	FALSE
PROPERTIES	PolicyRole

#### 6.15.1. The Single-Valued Property "PolicyRole"

This property represents the role associated with a PolicyRoleCollection. The property definition is as follows:

NAME	PolicyRole
DESCRIPTION	A string representing the role associated with a PolicyRoleCollection.
SYNTAX	string

#### 6.16. The Class "ReusablePolicyContainer"

The new class ReusablePolicyContainer is defined as follows:

NAME	ReusablePolicyContainer
DESCRIPTION	A class representing an administratively defined container for reusable policy-related information. This class does not introduce any additional properties beyond those in its superclass AdminDomain. It does, however, participate in a number of unique associations.
DERIVED FROM	AdminDomain
ABSTRACT	FALSE
PROPERTIES	(none)

#### 6.17. Deprecate PCIM's Class "PolicyRepository"

The class definition of PolicyRepository (from PCIM) is updated as follows, with an indication that the class has been deprecated. Note that when an element of the model is deprecated, its replacement element is identified explicitly.

NAME	PolicyRepository
DEPRECATED FOR	ReusablePolicyContainer
DESCRIPTION	A class representing an administratively defined container for reusable policy-related information. This class does not introduce any additional properties beyond those in its superclass AdminDomain. It does, however, participate in a number of unique associations.

DERIVED FROM	AdminDomain
ABSTRACT	FALSE
PROPERTIES	(none)

#### 6.18. The Abstract Class "FilterEntryBase"

FilterEntryBase is the abstract base class from which all filter entry classes are derived. It serves as the endpoint for the EntriesInFilterList aggregation, which groups filter entries into filter lists. Its properties include CIM naming attributes and an IsNegated boolean property (to easily "NOT" the match information specified in an instance of one of its subclasses).

The class definition is as follows:

NAME	FilterEntryBase
DESCRIPTION	An abstract class representing a single filter that is aggregated into a FilterList via the aggregation EntriesInFilterList.
DERIVED FROM	Logicalelement
TYPE	Abstract
PROPERTIES	IsNegated

#### 6.19. The Class "IpHeadersFilter"

This concrete class contains the most commonly required properties for performing filtering on IP, TCP or UDP headers. Properties not present in an instance of IPHeadersFilter are treated as 'all values'. A property HdrIpVersion identifies whether the IP addresses in an instance are IPv4 or IPv6 addresses. Since the source and destination IP addresses come from the same packet header, they will always be of the same type.

The class definition is as follows:

NAME	IpHeadersFilter
DESCRIPTION	A class representing an entire IP header filter, or any subset of one.
DERIVED FROM	FilterEntryBase
TYPE	Concrete
PROPERTIES	HdrIpVersion, HdrSrcAddress, HdrSrcAddressEndOfRange, HdrSrcMask, HdrDestAddress, HdrDestAddressEndOfRange, HdrDestMask, HdrProtocolID, HdrSrcPortStart, HdrSrcPortEnd, HdrDestPortStart, HdrDestPortEnd, HdrDSCP[ ], HdrFlowLabel

### 6.19.1. The Property HdrIpVersion

This property is an 8-bit unsigned integer, identifying the version of the IP addresses to be filtered on. IP versions are identified as they are in the Version field of the IP packet header - IPv4 = 4, IPv6 = 6. These two values are the only ones defined for this property.

The value of this property determines the sizes of the OctetStrings in the six properties HdrSrcAddress, HdrSrcAddressEndOfRange, HdrSrcMask, HdrDestAddress, HdrDestAddressEndOfRange, and HdrDestMask, as follows:

- o IPv4: OctetString(SIZE (4))
- o IPv6: OctetString(SIZE (16|20)), depending on whether a scope identifier is present

If a value for this property is not provided, then the filter does not consider IP version in selecting matching packets, i.e., IP version matches for all values. In this case, the HdrSrcAddress, HdrSrcAddressEndOfRange, HdrSrcMask, HdrDestAddress, HdrDestAddressEndOfRange, and HdrDestMask must also not be present.

### 6.19.2. The Property HdrSrcAddress

This property is an OctetString, of a size determined by the value of the HdrIpVersion property, representing a source IP address. When there is no HdrSrcAddressEndOfRange value, this value is compared to the source address in the IP header, subject to the mask represented in the HdrSrcMask property. (Note that the mask is ANDed with the address.) When there is a HdrSrcAddressEndOfRange value, this value is the start of the specified range (i.e., the HdrSrcAddress is lower than the HdrSrcAddressEndOfRange) that is compared to the source address in the IP header and matches on any value in the range.

If a value for this property is not provided, then the filter does not consider HdrSrcAddress in selecting matching packets, i.e., HdrSrcAddress matches for all values.

### 6.19.3. The Property HdrSrcAddressEndOfRange

This property is an OctetString, of a size determined by the value of the HdrIpVersion property, representing the end of a range of source IP addresses (inclusive), where the start of the range is the HdrSrcAddress property value.

If a value for `HdrSrcAddress` is not provided, then this property also MUST NOT be provided. If a value for this property is provided, then `HdrSrcMask` MUST NOT be provided.

#### 6.19.4. The Property `HdrSrcMask`

This property is an `OctetString`, of a size determined by the value of the `HdrIpVersion` property, representing a mask to be used in comparing the source address in the IP header with the value represented in the `HdrSrcAddress` property.

If a value for this property is not provided, then the filter does not consider `HdrSrcMask` in selecting matching packets, i.e., the value of `HdrSrcAddress` or the source address range must match the source address in the packet exactly. If a value for this property is provided, then `HdrSrcAddressEndOfRange` MUST NOT be provided.

#### 6.19.5. The Property `HdrDestAddress`

This property is an `OctetString`, of a size determined by the value of the `HdrIpVersion` property, representing a destination IP address. When there is no `HdrDestAddressEndOfRange` value, this value is compared to the destination address in the IP header, subject to the mask represented in the `HdrDestMask` property. (Note that the mask is ANDed with the address.) When there is a `HdrDestAddressEndOfRange` value, this value is the start of the specified range (i.e., the `HdrDestAddress` is lower than the `HdrDestAddressEndOfRange`) that is compared to the destination address in the IP header and matches on any value in the range.

If a value for this property is not provided, then the filter does not consider `HdrDestAddress` in selecting matching packets, i.e., `HdrDestAddress` matches for all values.

#### 6.19.6. The Property `HdrDestAddressEndOfRange`

This property is an `OctetString`, of a size determined by the value of the `HdrIpVersion` property, representing the end of a range of destination IP addresses (inclusive), where the start of the range is the `HdrDestAddress` property value.

If a value for `HdrDestAddress` is not provided, then this property also MUST NOT be provided. If a value for this property is provided, then `HdrDestMask` MUST NOT be provided.

#### 6.19.7. The Property HdrDestMask

This property is an OctetString, of a size determined by the value of the HdrIpVersion property, representing a mask to be used in comparing the destination address in the IP header with the value represented in the HdrDestAddress property.

If a value for this property is not provided, then the filter does not consider HdrDestMask in selecting matching packets, i.e., the value of HdrDestAddress or the destination address range must match the destination address in the packet exactly. If a value for this property is provided, then HdrDestAddressEndOfRange MUST NOT be provided.

#### 6.19.8. The Property HdrProtocolID

This property is an 8-bit unsigned integer, representing an IP protocol type. This value is compared to the Protocol field in the IP header.

If a value for this property is not provided, then the filter does not consider HdrProtocolID in selecting matching packets, i.e., HdrProtocolID matches for all values.

#### 6.19.9. The Property HdrSrcPortStart

This property is a 16-bit unsigned integer, representing the lower end of a range of UDP or TCP source ports. The upper end of the range is represented by the HdrSrcPortEnd property. The value of HdrSrcPortStart MUST be no greater than the value of HdrSrcPortEnd. A single port is indicated by equal values for HdrSrcPortStart and HdrSrcPortEnd.

A source port filter is evaluated by testing whether the source port identified in the IP header falls within the range of values between HdrSrcPortStart and HdrSrcPortEnd, including these two end points.

If a value for this property is not provided, then the filter does not consider HdrSrcPortStart in selecting matching packets, i.e., there is no lower bound in matching source port values.

#### 6.19.10. The Property HdrSrcPortEnd

This property is a 16-bit unsigned integer, representing the upper end of a range of UDP or TCP source ports. The lower end of the range is represented by the HdrSrcPortStart property. The value of

HdrSrcPortEnd MUST be no less than the value of HdrSrcPortStart. A single port is indicated by equal values for HdrSrcPortStart and HdrSrcPortEnd.

A source port filter is evaluated by testing whether the source port identified in the IP header falls within the range of values between HdrSrcPortStart and HdrSrcPortEnd, including these two end points.

If a value for this property is not provided, then the filter does not consider HdrSrcPortEnd in selecting matching packets, i.e., there is no upper bound in matching source port values.

#### 6.19.11. The Property HdrDestPortStart

This property is a 16-bit unsigned integer, representing the lower end of a range of UDP or TCP destination ports. The upper end of the range is represented by the HdrDestPortEnd property. The value of HdrDestPortStart MUST be no greater than the value of HdrDestPortEnd. A single port is indicated by equal values for HdrDestPortStart and HdrDestPortEnd.

A destination port filter is evaluated by testing whether the destination port identified in the IP header falls within the range of values between HdrDestPortStart and HdrDestPortEnd, including these two end points.

If a value for this property is not provided, then the filter does not consider HdrDestPortStart in selecting matching packets, i.e., there is no lower bound in matching destination port values.

#### 6.19.12. The Property HdrDestPortEnd

This property is a 16-bit unsigned integer, representing the upper end of a range of UDP or TCP destination ports. The lower end of the range is represented by the HdrDestPortStart property. The value of HdrDestPortEnd MUST be no less than the value of HdrDestPortStart. A single port is indicated by equal values for HdrDestPortStart and HdrDestPortEnd.

A destination port filter is evaluated by testing whether the destination port identified in the IP header falls within the range of values between HdrDestPortStart and HdrDestPortEnd, including these two end points.

If a value for this property is not provided, then the filter does not consider HdrDestPortEnd in selecting matching packets, i.e., there is no upper bound in matching destination port values.

## 6.19.13. The Property HdrDSCP

The property HdrDSCP is defined as an array of uint8's, restricted to the range 0..63. Since DSCPs are defined as discrete code points, with no inherent structure, there is no semantically significant relationship between different DSCPs. Consequently, there is no provision for specifying a range of DSCPs in this property. However, a list of individual DSCPs, which are ORed together to form a filter, is supported by the array syntax.

If a value for this property is not provided, then the filter does not consider HdrDSCP in selecting matching packets, i.e., HdrDSCP matches for all values.

## 6.19.14. The Property HdrFlowLabel

The 20-bit Flow Label field in the IPv6 header may be used by a source to label sequences of packets for which it requests special handling by IPv6 devices, such as non-default quality of service or 'real-time' service. This property is an octet string of size 3 (that is, 24 bits), in which the 20-bit Flow Label appears in the rightmost 20 bits, padded on the left with b'0000'.

If a value for this property is not provided, then the filter does not consider HdrFlowLabel in selecting matching packets, i.e., HdrFlowLabel matches for all values.

## 6.20. The Class "8021Filter"

This concrete class allows 802.1.source and destination MAC addresses, as well as the 802.1 protocol ID, priority, and VLAN identifier fields, to be expressed in a single object

The class definition is as follows:

NAME	8021Filter
DESCRIPTION	A class that allows 802.1 source and destination MAC address and protocol ID, priority, and VLAN identifier filters to be expressed in a single object.
DERIVED FROM	FilterEntryBase
TYPE	Concrete
PROPERTIES	8021HdrSrcMACAddr, 8021HdrSrcMACMask, 8021HdrDestMACAddr, 8021HdrDestMACMask, 8021HdrProtocolID, 8021HdrPriorityValue, 8021HDRVLANID

#### 6.20.1. The Property 8021HdrSrcMACAddr

This property is an OctetString of size 6, representing a 48-bit source MAC address in canonical format. This value is compared to the SourceAddress field in the MAC header, subject to the mask represented in the 8021HdrSrcMACMask property.

If a value for this property is not provided, then the filter does not consider 8021HdrSrcMACAddr in selecting matching packets, i.e., 8021HdrSrcMACAddr matches for all values.

#### 6.20.2. The Property 8021HdrSrcMACMask

This property is an OctetString of size 6, representing a 48-bit mask to be used in comparing the SourceAddress field in the MAC header with the value represented in the 8021HdrSrcMACAddr property.

If a value for this property is not provided, then the filter does not consider 8021HdrSrcMACMask in selecting matching packets, i.e., the value of 8021HdrSrcMACAddr must match the source MAC address in the packet exactly.

#### 6.20.3. The Property 8021HdrDestMACAddr

This property is an OctetString of size 6, representing a 48-bit destination MAC address in canonical format. This value is compared to the DestinationAddress field in the MAC header, subject to the mask represented in the 8021HdrDestMACMask property.

If a value for this property is not provided, then the filter does not consider 8021HdrDestMACAddr in selecting matching packets, i.e., 8021HdrDestMACAddr matches for all values.

#### 6.20.4. The Property 8021HdrDestMACMask

This property is an OctetString of size 6, representing a 48-bit mask to be used in comparing the DestinationAddress field in the MAC header with the value represented in the 8021HdrDestMACAddr property.

If a value for this property is not provided, then the filter does not consider 8021HdrDestMACMask in selecting matching packets, i.e., the value of 8021HdrDestMACAddr must match the destination MAC address in the packet exactly.

#### 6.20.5. The Property 8021HdrProtocolID

This property is a 16-bit unsigned integer, representing an Ethernet protocol type. This value is compared to the Ethernet Type field in the 802.3 MAC header.

If a value for this property is not provided, then the filter does not consider 8021HdrProtocolID in selecting matching packets, i.e., 8021HdrProtocolID matches for all values.

#### 6.20.6. The Property 8021HdrPriorityValue

This property is an 8-bit unsigned integer, representing an 802.1Q priority. This value is compared to the Priority field in the 802.1Q header. Since the 802.1Q Priority field consists of 3 bits, the values for this property are limited to the range 0..7.

If a value for this property is not provided, then the filter does not consider 8021HdrPriorityValue in selecting matching packets, i.e., 8021HdrPriorityValue matches for all values.

#### 6.20.7. The Property 8021HdrVLANID

This property is a 32-bit unsigned integer, representing an 802.1Q VLAN Identifier. This value is compared to the VLAN ID field in the 802.1Q header. Since the 802.1Q VLAN ID field consists of 12 bits, the values for this property are limited to the range 0..4095.

If a value for this property is not provided, then the filter does not consider 8021HdrVLANID in selecting matching packets, i.e., 8021HdrVLANID matches for all values.

#### 6.21. The Class FilterList

This is a concrete class that aggregates instances of (subclasses of) FilterEntryBase via the aggregation EntriesInFilterList. It is possible to aggregate different types of filters into a single FilterList - for example, packet header filters (represented by the IpHeadersFilter class) and security filters (represented by subclasses of FilterEntryBase defined by IPsec).

The aggregation property EntriesInFilterList.EntrySequence is always set to 0, to indicate that the aggregated filter entries are ANDed together to form a selector for a class of traffic.

The class definition is as follows:

NAME	FilterList
DESCRIPTION	A concrete class representing the aggregation of multiple filters.
DERIVED FROM	Logicalelement
TYPE	Concrete
PROPERTIES	Direction

#### 6.21.1. The Property Direction

This property is a 16-bit unsigned integer enumeration, representing the direction of the traffic flow to which the FilterList is to be applied. Defined enumeration values are

- o NotApplicable(0)
- o Input(1)
- o Output(2)
- o Both(3) - This value is used to indicate that the direction is immaterial, e.g., to filter on a source subnet regardless of whether the flow is inbound or outbound
- o Mirrored(4) - This value is also applicable to both inbound and outbound flow processing, but it indicates that the filter criteria are applied asymmetrically to traffic in both directions and, thus, specifies the reversal of source and destination criteria (as opposed to the equality of these criteria as indicated by "Both"). The match conditions in the aggregated FilterEntryBase subclass instances are defined from the perspective of outbound flows and applied to inbound flows as well by reversing the source and destination criteria. So, for example, consider a FilterList with 3 filter entries indicating destination port = 80, and source and destination addresses of a and b, respectively. Then, for the outbound direction, the filter entries match as specified and the 'mirror' (for the inbound direction) matches on source port = 80 and source and destination addresses of b and a, respectively.

### 7. Association and Aggregation Definitions

The following definitions supplement those in PCIM itself. PCIM definitions that are not DEPRECATED here are still current parts of the overall Policy Core Information Model.

#### 7.1. The Aggregation "PolicySetComponent"

PolicySetComponent is a new aggregation class that collects instances of PolicySet subclasses (PolicyGroups and PolicyRules) into coherent sets of policies.

NAME	PolicySetComponent
DESCRIPTION	A concrete class representing the components of a policy set that have the same decision strategy, and are prioritized within the set.
DERIVED FROM	PolicyComponent
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref PolicySet[0..n]] PartComponent[ref PolicySet[0..n]] Priority

The definition of the Priority property is unchanged from its previous definition in [PCIM].

NAME	Priority
DESCRIPTION	A non-negative integer for prioritizing this PolicySet component relative to other components of the same PolicySet. A larger value indicates a higher priority.
SYNTAX	uint16
DEFAULT VALUE	0

#### 7.2. Deprecate PCIM's Aggregation "PolicyGroupInPolicyGroup"

The new aggregation PolicySetComponent is used directly to represent aggregation of PolicyGroups by a higher-level PolicyGroup. Thus the aggregation PolicyGroupInPolicyGroup is no longer needed, and can be deprecated.

NAME	PolicyGroupInPolicyGroup
DEPRECATED FOR	PolicySetComponent
DESCRIPTION	A class representing the aggregation of PolicyGroups by a higher-level PolicyGroup.
DERIVED FROM	PolicyComponent
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref PolicyGroup[0..n]] PartComponent[ref PolicyGroup[0..n]]

#### 7.3. Deprecate PCIM's Aggregation "PolicyRuleInPolicyGroup"

The new aggregation PolicySetComponent is used directly to represent aggregation of PolicyRules by a PolicyGroup. Thus the aggregation PolicyRuleInPolicyGroup is no longer needed, and can be deprecated.

NAME	PolicyRuleInPolicyGroup
DEPRECATED FOR	PolicySetComponent
DESCRIPTION	A class representing the aggregation of PolicyRules by a PolicyGroup.
DERIVED FROM	PolicyComponent

ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref PolicyGroup[0..n]] PartComponent[ref PolicyRule[0..n]]

#### 7.4. The Abstract Association "PolicySetInSystem"

PolicySetInSystem is a new association that defines a relationship between a System and a PolicySet used in the administrative scope of that system (e.g., AdminDomain, ComputerSystem). The Priority property is used to assign a relative priority to a PolicySet within the administrative scope in contexts where it is not a component of another PolicySet.

NAME	PolicySetInSystem
DESCRIPTION	An abstract class representing the relationship between a System and a PolicySet that is used in the administrative scope of the System.
DERIVED FROM	PolicyInSystem
ABSTRACT	TRUE
PROPERTIES	Antecedent[ref System[0..1]] Dependent [ref PolicySet[0..n]] Priority

The Priority property is used to specify the relative priority of the referenced PolicySet when there are more than one PolicySet instances applied to a managed resource that are not PolicySetComponents and, therefore, have no other relative priority defined.

NAME	Priority
DESCRIPTION	A non-negative integer for prioritizing the referenced PolicySet among other PolicySet instances that are not components of a common PolicySet. A larger value indicates a higher priority.
SYNTAX	uint16
DEFAULT VALUE	0

#### 7.5. Update PCIM's Weak Association "PolicyGroupInSystem"

Regardless of whether it a component of another PolicySet, a PolicyGroup is itself defined within the scope of a System. This association links a PolicyGroup to the System in whose scope the PolicyGroup is defined. It is a subclass of the abstract PolicySetInSystem association. The class definition for the association is as follows:

NAME	PolicyGroupInSystem
DESCRIPTION	A class representing the fact that a PolicyGroup is defined within the scope of a System.
DERIVED FROM	PolicySetInSystem
ABSTRACT	FALSE
PROPERTIES	Antecedent[ref System[1..1]] Dependent [ref PolicyGroup[weak]]

The Reference "Antecedent" is inherited from PolicySetInSystem, and overridden to restrict its cardinality to [1..1]. It serves as an object reference to a System that provides a scope for one or more PolicyGroups. Since this is a weak association, the cardinality for this object reference is always 1, that is, a PolicyGroup is always defined within the scope of exactly one System.

The Reference "Dependent" is inherited from PolicySetInSystem, and overridden to become an object reference to a PolicyGroup defined within the scope of a System. Note that for any single instance of the association class PolicyGroupInSystem, this property (like all reference properties) is single-valued. The [0..n] cardinality indicates that a given System may have 0, 1, or more than one PolicyGroups defined within its scope.

#### 7.6. Update PCIM's Weak Association "PolicyRuleInSystem"

Regardless of whether it a component of another PolicySet, a PolicyRule is itself defined within the scope of a System. This association links a PolicyRule to the System in whose scope the PolicyRule is defined. It is a subclass of the abstract PolicySetInSystem association. The class definition for the association is as follows:

NAME	PolicyRuleInSystem
DESCRIPTION	A class representing the fact that a PolicyRule is defined within the scope of a System.
DERIVED FROM	PolicySetInSystem
ABSTRACT	FALSE
PROPERTIES	Antecedent[ref System[1..1]] Dependent[ref PolicyRule[weak]]

The Reference "Antecedent" is inherited from PolicySetInSystem, and overridden to restrict its cardinality to [1..1]. It serves as an object reference to a System that provides a scope for one or more PolicyRules. Since this is a weak association, the cardinality for this object reference is always 1, that is, a PolicyRule is always defined within the scope of exactly one System.

The Reference "Dependent" is inherited from PolicySetInSystem, and overridden to become an object reference to a PolicyRule defined within the scope of a System. Note that for any single instance of the association class PolicyRuleInSystem, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given System may have 0, 1, or more than one PolicyRules defined within its scope.

#### 7.7. The Abstract Aggregation "PolicyConditionStructure"

NAME	PolicyConditionStructure
DESCRIPTION	A class representing the aggregation of PolicyConditions by an aggregating instance.
DERIVED FROM	PolicyComponent
ABSTRACT	TRUE
PROPERTIES	PartComponent[ref PolicyCondition[0..n]] GroupNumber ConditionNegated

#### 7.8. Update PCIM's Aggregation "PolicyConditionInPolicyRule"

The PCIM aggregation "PolicyConditionInPolicyRule" is updated, to make it a subclass of the new abstract aggregation PolicyConditionStructure. The properties GroupNumber and ConditionNegated are now inherited, rather than specified explicitly as they were in PCIM.

NAME	PolicyConditionInPolicyRule
DESCRIPTION	A class representing the aggregation of PolicyConditions by a PolicyRule.
DERIVED FROM	PolicyConditionStructure
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref PolicyRule[0..n]]

#### 7.9. The Aggregation "PolicyConditionInPolicyCondition"

A second subclass of PolicyConditionStructure is defined, representing the compounding of policy conditions into a higher-level policy condition.

NAME	PolicyConditionInPolicyCondition
DESCRIPTION	A class representing the aggregation of PolicyConditions by another PolicyCondition.
DERIVED FROM	PolicyConditionStructure
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref CompoundPolicyCondition[0..n]]

## 7.10. The Abstract Aggregation "PolicyActionStructure"

NAME	PolicyActionStructure
DESCRIPTION	A class representing the aggregation of PolicyActions by an aggregating instance.
DERIVED FROM	PolicyComponent
ABSTRACT	TRUE
PROPERTIES	PartComponent[ref PolicyAction[0..n]] ActionOrder

The definition of the ActionOrder property appears in Section 7.8.3 of PCIM [1].

## 7.11. Update PCIM's Aggregation "PolicyActionInPolicyRule"

The PCIM aggregation "PolicyActionInPolicyRule" is updated, to make it a subclass of the new abstract aggregation PolicyActionStructure. The property ActionOrder is now inherited, rather than specified explicitly as it was in PCIM.

NAME	PolicyActionInPolicyRule
DESCRIPTION	A class representing the aggregation of PolicyActions by a PolicyRule.
DERIVED FROM	PolicyActionStructure
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref PolicyRule[0..n]]

## 7.12. The Aggregation "PolicyActionInPolicyAction"

A second subclass of PolicyActionStructure is defined, representing the compounding of policy actions into a higher-level policy action.

NAME	PolicyActionInPolicyAction
DESCRIPTION	A class representing the aggregation of PolicyActions by another PolicyAction.
DERIVED FROM	PolicyActionStructure
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref CompoundPolicyAction[0..n]]

## 7.13. The Aggregation "PolicyVariableInSimplePolicyCondition"

A simple policy condition is represented as an ordered triplet {variable, operator, value}. This aggregation provides the linkage between a SimplePolicyCondition instance and a single PolicyVariable. The aggregation PolicyValueInSimplePolicyCondition links the SimplePolicyCondition to a single PolicyValue. The Operator property of SimplePolicyCondition represents the third element of the triplet, the operator.

The class definition for this aggregation is as follows:

```

NAME           PolicyVariableInSimplePolicyCondition
DERIVED FROM   PolicyComponent
ABSTRACT       False
PROPERTIES     GroupComponent[ref SimplePolicyCondition[0..n]]
                PartComponent[ref PolicyVariable[1..1] ]

```

The reference property "GroupComponent" is inherited from PolicyComponent, and overridden to become an object reference to a SimplePolicyCondition that contains exactly one PolicyVariable. Note that for any single instance of the aggregation class PolicyVariableInSimplePolicyCondition, this property is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more SimplePolicyCondition objects that contain any given policy variable object.

The reference property "PartComponent" is inherited from PolicyComponent, and overridden to become an object reference to a PolicyVariable that is defined within the scope of a SimplePolicyCondition. Note that for any single instance of the association class PolicyVariableInSimplePolicyCondition, this property (like all reference properties) is single-valued. The [1..1] cardinality indicates that a SimplePolicyCondition must have exactly one policy variable defined within its scope in order to be meaningful.

#### 7.14. The Aggregation "PolicyValueInSimplePolicyCondition"

A simple policy condition is represented as an ordered triplet {variable, operator, value}. This aggregation provides the linkage between a SimplePolicyCondition instance and a single PolicyValue. The aggregation PolicyVariableInSimplePolicyCondition links the SimplePolicyCondition to a single PolicyVariable. The Operator property of SimplePolicyCondition represents the third element of the triplet, the operator.

The class definition for this aggregation is as follows:

```

NAME           PolicyValueInSimplePolicyCondition
DERIVED FROM   PolicyComponent
ABSTRACT       False
PROPERTIES     GroupComponent[ref SimplePolicyCondition[0..n]]
                PartComponent[ref PolicyValue[1..1] ]

```

The reference property "GroupComponent" is inherited from PolicyComponent, and overridden to become an object reference to a SimplePolicyCondition that contains exactly one PolicyValue. Note

that for any single instance of the aggregation class `PolicyValueInSimplePolicyCondition`, this property is single-valued. The `[0..n]` cardinality indicates that there may be 0, 1, or more `SimplePolicyCondition` objects that contain any given policy value object.

The reference property "PartComponent" is inherited from `PolicyComponent`, and overridden to become an object reference to a `PolicyValue` that is defined within the scope of a `SimplePolicyCondition`. Note that for any single instance of the association class `PolicyValueInSimplePolicyCondition`, this property (like all reference properties) is single-valued. The `[1..1]` cardinality indicates that a `SimplePolicyCondition` must have exactly one policy value defined within its scope in order to be meaningful.

#### 7.15. The Aggregation "PolicyVariableInSimplePolicyAction"

A simple policy action is represented as a pair `{variable, value}`. This aggregation provides the linkage between a `SimplePolicyAction` instance and a single `PolicyVariable`. The aggregation `PolicyValueInSimplePolicyAction` links the `SimplePolicyAction` to a single `PolicyValue`.

The class definition for this aggregation is as follows:

NAME	<code>PolicyVariableInSimplePolicyAction</code>
DERIVED FROM	<code>PolicyComponent</code>
ABSTRACT	False
PROPERTIES	<code>GroupComponent[ref SimplePolicyAction[0..n]]</code> <code>PartComponent[ref PolicyVariable[1..1] ]</code>

The reference property "GroupComponent" is inherited from `PolicyComponent`, and overridden to become an object reference to a `SimplePolicyAction` that contains exactly one `PolicyVariable`. Note that for any single instance of the aggregation class `PolicyVariableInSimplePolicyAction`, this property is single-valued. The `[0..n]` cardinality indicates that there may be 0, 1, or more `SimplePolicyAction` objects that contain any given policy variable object.

The reference property "PartComponent" is inherited from `PolicyComponent`, and overridden to become an object reference to a `PolicyVariable` that is defined within the scope of a `SimplePolicyAction`. Note that for any single instance of the association class `PolicyVariableInSimplePolicyAction`, this property (like all reference properties) is single-valued. The `[1..1]` cardinality indicates that a `SimplePolicyAction` must have exactly one policy variable defined within its scope in order to be meaningful.

### 7.16. The Aggregation "PolicyValueInSimplePolicyAction"

A simple policy action is represented as a pair {variable, value}. This aggregation provides the linkage between a SimplePolicyAction instance and a single PolicyValue. The aggregation PolicyVariableInSimplePolicyAction links the SimplePolicyAction to a single PolicyVariable.

The class definition for this aggregation is as follows:

```

NAME                PolicyValueInSimplePolicyAction
DERIVED FROM        PolicyComponent
ABSTRACT            False
PROPERTIES           GroupComponent[ref SimplePolicyAction[0..n]]
                   PartComponent[ref PolicyValue[1..1] ]

```

The reference property "GroupComponent" is inherited from PolicyComponent, and overridden to become an object reference to a SimplePolicyAction that contains exactly one PolicyValue. Note that for any single instance of the aggregation class PolicyValueInSimplePolicyAction, this property is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more SimplePolicyAction objects that contain any given policy value object.

The reference property "PartComponent" is inherited from PolicyComponent, and overridden to become an object reference to a PolicyValue that is defined within the scope of a SimplePolicyAction. Note that for any single instance of the association class PolicyValueInSimplePolicyAction, this property (like all reference properties) is single-valued. The [1..1] cardinality indicates that a SimplePolicyAction must have exactly one policy value defined within its scope in order to be meaningful.

### 7.17. The Association "ReusablePolicy"

The association ReusablePolicy makes it possible to include any subclass of the abstract class "Policy" in a ReusablePolicyContainer.

```

NAME                ReusablePolicy
DESCRIPTION          A class representing the inclusion of a reusable
                   policy element in a ReusablePolicyContainer.
                   Reusable elements may be PolicyGroups, PolicyRules,
                   PolicyConditions, PolicyActions, PolicyVariables,
                   PolicyValues, or instances of any other subclasses
                   of the abstract class Policy.

```

```

DERIVED FROM      PolicyInSystem
ABSTRACT          FALSE
PROPERTIES        Antecedent[ref ReusablePolicyContainer[0..1]]

```

#### 7.18. Deprecate PCIM's "PolicyConditionInPolicyRepository"

```

NAME              PolicyConditionInPolicyRepository
DEPRECATED FOR    ReusablePolicy
DESCRIPTION       A class representing the inclusion of a reusable
                  PolicyCondition in a PolicyRepository.
DERIVED FROM      PolicyInSystem
ABSTRACT          FALSE
PROPERTIES        Antecedent[ref PolicyRepository[0..1]]
                  Dependent[ref PolicyCondition[0..n]]

```

#### 7.19. Deprecate PCIM's "PolicyActionInPolicyRepository"

```

NAME              PolicyActionInPolicyRepository
DEPRECATED FOR    ReusablePolicy
DESCRIPTION       A class representing the inclusion of a reusable
                  PolicyAction in a PolicyRepository.
DERIVED FROM      PolicyInSystem
ABSTRACT          FALSE
PROPERTIES        Antecedent[ref PolicyRepository[0..1]]
                  Dependent[ref PolicyAction[0..n]]

```

#### 7.20. The Association ExpectedPolicyValuesForVariable

This association links a PolicyValue object to a PolicyVariable object, modeling the set of expected values for that PolicyVariable. Using this association, a variable (instance) may be constrained to be bound- to/assigned only a set of allowed values. For example, modeling an enumerated source port variable, one creates an instance of the PolicySourcePortVariable class and associates with it the set of values (integers) representing the allowed enumeration, using appropriate number of instances of the ExpectedPolicyValuesForVariable association.

Note that a single variable instance may be constrained by any number of values, and a single value may be used to constrain any number of variables. These relationships are manifested by the n-to-m cardinality of the association.

The purpose of this association is to support validation of simple policy conditions and simple policy actions, prior to their deployment to an enforcement point. This association, and the

PolicyValue object that it refers to, plays no role when a PDP or a PEP is evaluating a simple policy condition, or executing a simple policy action. See Section 5.8.3 for more details on this point.

The class definition for the association is as follows:

NAME	ExpectedPolicyValuesForVariable
DESCRIPTION	A class representing the association of a set of expected values to a variable object.
DERIVED FROM	Dependency
ABSTRACT	FALSE
PROPERTIES	Antecedent [ref PolicyVariable[0..n]] Dependent [ref PolicyValue [0..n]]

The reference property Antecedent is inherited from Dependency. Its type and cardinality are overridden to provide the semantics of a variable optionally having value constraints. The [0..n] cardinality indicates that any number of variables may be constrained by a given value.

The reference property "Dependent" is inherited from Dependency, and overridden to become an object reference to a PolicyValue representing the values that a particular PolicyVariable can have. The [0..n] cardinality indicates that a given policy variable may have 0, 1 or more than one PolicyValues defined to model the set(s) of values that the policy variable can take.

#### 7.21. The Aggregation "ContainedDomain"

The aggregation ContainedDomain provides a means of nesting of one ReusablePolicyContainer inside another one. The aggregation is defined at the level of ReusablePolicyContainer's superclass, AdminDomain, to give it applicability to areas other than Core Policy.

NAME	ContainedDomain
DESCRIPTION	A class representing the aggregation of lower level administrative domains by a higher-level AdminDomain.
DERIVED FROM	SystemComponent
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref AdminDomain [0..n]] PartComponent[ref AdminDomain [0..n]]

## 7.22. Deprecate PCIM's "PolicyRepositoryInPolicyRepository"

NAME	PolicyRepositoryInPolicyRepository
DEPRECATED FOR	ContainedDomain
DESCRIPTION	A class representing the aggregation of PolicyRepositories by a higher-level PolicyRepository.
DERIVED FROM	SystemComponent
ABSTRACT	FALSE
PROPERTIES	GroupComponent[ref PolicyRepository[0..n]] PartComponent[ref PolicyRepository[0..n]]

## 7.23. The Aggregation "EntriesInFilterList"

This aggregation is a specialization of the Component aggregation; it is used to define a set of filter entries (subclasses of FilterEntryBase) that are aggregated by a FilterList.

The cardinalities of the aggregation itself are 0..1 on the FilterList end, and 0..n on the FilterEntryBase end. Thus in the general case, a filter entry can exist without being aggregated into any FilterList. However, the only way a filter entry can figure in the PCIME model is by being aggregated into a FilterList by this aggregation.

The class definition for the aggregation is as follows:

NAME	EntriesInFilterList
DESCRIPTION	An aggregation used to define a set of filter entries (subclasses of FilterEntryBase) that are aggregated by a particular FilterList.
DERIVED FROM	Component
ABSTRACT	False
PROPERTIES	GroupComponent[ref FilterList[0..1]], PartComponent[ref FilterEntryBase[0..n], EntrySequence

## 7.23.1. The Reference GroupComponent

This property is overridden in this aggregation to represent an object reference to a FilterList object (instead of to the more generic ManagedSystemElement object defined in its superclass). It also restricts the cardinality of the aggregate to 0..1 (instead of the more generic 0-or-more), representing the fact that a filter entry always exists within the context of at most one FilterList.

### 7.23.2. The Reference PartComponent

This property is overridden in this aggregation to represent an object reference to a `FilterEntryBase` object (instead of to the more generic `ManagedSystemElement` object defined in its superclass). This object represents a single filter entry, which may be aggregated with other filter entries to form the `FilterList`.

### 7.23.3. The Property EntrySequence

An unsigned 16-bit integer indicating the order of the filter entry relative to all others in the `FilterList`. The default value '0' indicates that order is not significant, because the entries in this `FilterList` are ANDed together.

### 7.24. The Aggregation "ElementInPolicyRoleCollection"

The following aggregation is used to associate `ManagedElements` with a `PolicyRoleCollection` object that represents a role played by these `ManagedElements`.

NAME	<code>ElementInPolicyRoleCollection</code>
DESCRIPTION	A class representing the inclusion of a <code>ManagedElement</code> in a collection, specified as having a given role. All the managed elements in the collection share the same role.
DERIVED FROM	<code>MemberOfCollection</code>
ABSTRACT	FALSE
PROPERTIES	<code>Collection[ref PolicyRoleCollection [0..n]]</code> <code>Member[ref ManagedElement [0..n]]</code>

### 7.25. The Weak Association "PolicyRoleCollectionInSystem"

A `PolicyRoleCollection` is defined within the scope of a `System`. This association links a `PolicyRoleCollection` to the `System` in whose scope it is defined.

When associating a `PolicyRoleCollection` with a `System`, this should be done consistently with the system that scopes the policy rules/groups that are applied to the resources in that collection. A `PolicyRoleCollection` is associated with the same system as the applicable `PolicyRules` and/or `PolicyGroups`, or to a `System` higher in the tree formed by the `SystemComponent` association.

The class definition for the association is as follows:

NAME	PolicyRoleCollectionInSystem
DESCRIPTION	A class representing the fact that a PolicyRoleCollection is defined within the scope of a System.
DERIVED FROM	Dependency
ABSTRACT	FALSE
PROPERTIES	Antecedent[ref System[1..1]] Dependent[ref PolicyRoleCollection[weak]]

The reference property Antecedent is inherited from Dependency, and overridden to become an object reference to a System, and to restrict its cardinality to [1..1]. It serves as an object reference to a System that provides a scope for one or more PolicyRoleCollections. Since this is a weak association, the cardinality for this object reference is always 1, that is, a PolicyRoleCollection is always defined within the scope of exactly one System.

The reference property Dependent is inherited from Dependency, and overridden to become an object reference to a PolicyRoleCollection defined within the scope of a System. Note that for any single instance of the association class PolicyRoleCollectionInSystem, this property (like all Reference properties) is single-valued. The [0..n] cardinality indicates that a given System may have 0, 1, or more than one PolicyRoleCollections defined within its scope.

## 8. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11.

Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

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

## 9. Acknowledgements

The starting point for this document was PCIM itself [1], and the first three submodels derived from it [11], [12], [13]. The authors of these documents created the extensions to PCIM, and asked the questions about PCIM, that are reflected in PCIME.

## 10. Contributors

This document includes text written by a number of authors (including the editor), that was subsequently merged by the editor. The following people contributed text to this document:

Lee Rafalow  
IBM Corporation, BRQA/501  
4205 S. Miami Blvd.  
Research Triangle Park, NC 27709

Phone: +1 919-254-4455  
Fax: +1 919-254-6243  
EMail: rafalow@us.ibm.com

Yoram Ramberg  
Cisco Systems  
4 Maskit Street  
Herzliya Pituach, Israel 46766

Phone: +972-9-970-0081  
Fax: +972-9-970-0219  
EMail: yramberg@cisco.com

Yoram Snir  
Cisco Systems  
4 Maskit Street  
Herzliya Pituach, Israel 46766

Phone: +972-9-970-0085  
Fax: +972-9-970-0366  
EMail: ysnir@cisco.com

Andrea Westerinen  
Cisco Systems  
Building 20  
725 Alder Drive  
Milpitas, CA 95035

Phone: +1-408-853-8294  
Fax: +1-408-527-6351  
EMail: andreaw@cisco.com

Ritu Chadha  
Telcordia Technologies  
MCC 1J-218R  
445 South Street  
Morristown NJ 07960.

Phone: +1-973-829-4869  
Fax: +1-973-829-5889  
EMail: chadha@research.telcordia.com

Marcus Brunner  
NEC Europe Ltd.  
C&C Research Laboratories  
Adenauerplatz 6  
D-69115 Heidelberg, Germany

Phone: +49 (0)6221 9051129  
Fax: +49 (0)6221 9051155  
EMail: brunner@ccrle.nec.de

Ron Cohen  
Ntear LLC

EMail: ronc@ntear.com

John Strassner  
INTELLIDEN, Inc.  
90 South Cascade Avenue  
Colorado Springs, CO 80903

Phone: +1-719-785-0648  
EMail: john.strassner@intelliden.com

## 11. Security Considerations

The Policy Core Information Model (PCIM) [1] describes the general security considerations related to the general core policy model. The extensions defined in this document do not introduce any additional considerations related to security.

## 12. Normative References

- [1] Moore, B., Ellesson, E., Strassner, J. and A. Westerinen, "Policy Core Information Model -- Version 1 Specification", RFC 3060, February 2001.
- [2] Distributed Management Task Force, Inc., "DMTF Technologies: CIM Standards CIM Schema: Version 2.5", available at [http://www.dmtf.org/standards/cim\\_schema\\_v25.php](http://www.dmtf.org/standards/cim_schema_v25.php).
- [3] Distributed Management Task Force, Inc., "Common Information Model (CIM) Specification: Version 2.2", June 14, 1999, available at <http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf>.
- [4] Mockapetris, P., "Domain Names - implementation and specification", STD 13, RFC 1035, November 1987.
- [5] Wahl, M., Coulbeck, A., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.
- [6] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.
- [7] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.
- [8] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

## 13. Informative References

- [9] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.
- [10] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J. and Waldbusser, "Terminology for Policy-Based Management", RFC 3198, November 2001.

- [11] Snir, Y., and Y. Ramberg, J. Strassner, R. Cohen, "Policy QoS Information Model", Work in Progress.
- [12] Jason, J., and L. Rafalow, E. Vyncke, "IPsec Configuration Policy Model", Work in Progress.
- [13] Chadha, R., and M. Brunner, M. Yoshida, J. Quittek, G. Mykoniatis, A. Poylisher, R. Vaidyanathan, A. Kind, F. Reichmeyer, "Policy Framework MPLS Information Model for QoS and TE", Work in Progress.
- [14] S. Waldbusser, and J. Saperia, T. Hongal, "Policy Based Management MIB", Work in Progress.
- [15] B. Moore, and D. Durham, J. Halpern, J. Strassner, A. Westerinen, W. Weiss, "Information Model for Describing Network Device QoS Datapath Mechanisms", Work in Progress.

#### Author's Address

Bob Moore  
IBM Corporation, BRQA/501  
4205 S. Miami Blvd.  
Research Triangle Park, NC 27709

Phone: +1 919-254-4436  
Fax: +1 919-254-6243  
EMail: remoore@us.ibm.com

## Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

