# 3GPP TS 29.198-13 V9.0.0 (2009-12)

*Technical Specification*

# 3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 13: Policy management Service Capability Feature (SCF) (Release 9)


Keywords
UMTS, API, OSA

***3GPP***

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

# Contents

# Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

# Introduction

The present document is part 13 of a multi-part TS covering the 3rd Generation Partnership Project: Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API), as identified below. The **API specification** (3GPP TS 29.198) is structured in the following Parts:

| | | |
|---|---|---|
| Part 1: | "Overview"; | |
| Part 2: | "Common Data Definitions"; | |
| Part 3: | "Framework"; | |
| Part 4: | "Call Control"; | |
| | Sub-part 1: "Call Control Common Definitions"; | |
| | Sub-part 2: "Generic Call Control SCF"; | |
| | Sub-part 3: "Multi-Party Call Control SCF"; | |
| | Sub-part 4: "Multi-Media Call Control SCF"; | |
| | Sub-part 5: "Conference Call Control SCF"; | (new in 3GPP Release 7) |
| Part 5: | "User Interaction SCF"; | |
| Part 6: | "Mobility SCF"; | |
| Part 7: | "Terminal Capabilities SCF"; | |
| Part 8: | "Data Session Control SCF"; | |
| Part 9: | "Generic Messaging SCF"; | (not part of 3GPP Release 7) |
| Part 10: | "Connectivity Manager SCF"; | (not part of 3GPP Release 7) |
| Part 11: | "Account Management SCF"; | |
| Part 12: | "Charging SCF". | |
| **Part 13:** | **"Policy Management SCF";** | |
| Part 14: | "Presence and Availability Management SCF"; | |
| Part 15: | "Multi Media Messaging SCF"; | |
| Part 16: | "Service Broker SCF". | (new in Release 7) |

The **Mapping specification of the OSA APIs and network protocols** (3GPP TR 29.998) is also structured as above. A mapping to network protocols is however not applicable for all Parts, but the numbering of Parts is kept. Also in case a Part is not supported in a Release, the numbering of the parts is maintained.

| OSA API specifications 29.198-family | | | | | | OSA API Mapping - 29.998-family | |
|---|---|---|---|---|---|---|---|
| 29.198-01 | Overview | | | | | 29.998-01 | Overview |
| 29.198-02 | Common Data Definitions | | | | | *29.998-02* | *Not Applicable* |
| 29.198-03 | Framework | | | | | *29.998-03* | *Not Applicable* |
| Call Control (CC) SCF | 29.198-04-1 Common CC data definitions | 29.198-04-2 Generic CC SCF | 29.198-04-3 Multi-Party CC SCF | 29.198-04-4 Multi-media CC SCF | 29.198-04-5 Conf CC SCF | 29.998-04-1 | Generic Call Control – CAP mapping |
| | | | | | | *29.998-04-2* | *Generic Call Control – INAP mapping* |
| | | | | | | *29.998-04-3* | *Generic Call Control – Megaco mapping* |
| | | | | | | 29.998-04-4 | Multiparty Call Control – ISC mapping |
| 29.198-05 | User Interaction SCF | | | | | 29.998-05-1 | User Interaction – CAP mapping |
| | | | | | | *29.998-05-2* | *User Interaction – INAP mapping* |
| | | | | | | *29.998-05-3* | *User Interaction – Megaco mapping* |
| | | | | | | 29.998-05-4 | User Interaction – SMS mapping |
| 29.198-06 | Mobility SCF | | | | | 29.998-06-1 | User Status and User Location – MAP mapping |
| | | | | | | 29.998-06-2 | User Status and User Location – SIP mapping |
| 29.198-07 | Terminal Capabilities SCF | | | | | *29.998-07* | *Not Applicable* |
| 29.198-08 | Data Session Control SCF | | | | | 29.998-08 | Data Session Control – CAP mapping |
| *29.198-09* | *Generic Messaging SCF* | | | | | *29.998-09* | *Not Applicable* |
| *29.198-10* | *Connectivity Manager SCF* | | | | | *29.998-10* | *Not Applicable* |
| 29.198-11 | Account Management SCF | | | | | *29.998-11* | *Not Applicable* |
| 29.198-12 | Charging SCF | | | | | *29.998-12* | *Not Applicable* |
| **29.198-13** | **Policy Management SCF** | | | | | *29.998-13* | *Not Applicable* |
| 29.198-14 | Presence & Availability Management SCF | | | | | *29.998-14* | *Not Applicable* |
| 29.198-15 | Multi-media Messaging SCF | | | | | *29.998-15* | *Not Applicable* |
| 29.198-16 | Service Broker SCF | | | | | *29.998-16* | *Not Applicable* |

# 1 Scope

The present document is part 13 of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA).

The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA APIs. The concepts and the functional architecture for the OSA are contained in 3GPP TS 23.198 [3]. The requirements for OSA are contained in 3GPP TS 22.127 [2].

The present document specifies the Policy Management Service Capability Feature (SCF) aspects of the interface. All aspects of the Policy Management SCF are defined here, these being:

- Sequence Diagrams
- Class Diagrams
- Interface specification plus detailed method descriptions
- State Transition diagrams
- Data Definitions
- IDL Description of the interfaces

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

This specification has been defined jointly between 3GPP TSG CT WG5, ETSI TISPAN and the Parlay Consortium, in co-operation with a number of JAIN™ Community member companies.

Maintenance of up to 3GPP Rel-8 and new OSA Stage 1, 2 and 3 work beyond Rel-9 was moved to OMA in June 2008.

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TS 29.198-1: "Open Service Access; Application Programming Interface; Part 1: Overview".

[2] 3GPP TS 22.127: "Service Requirement for the Open Service Access (OSA); Stage 1".

[3] 3GPP TS 23.198: "Open Service Access (OSA); Stage 2".

[4] IETF RFC 3460: PCIM, "Policy Core Information Model (PCIM) Extensions".

[5] IETF RFC 2445: "Internet Calendaring and Scheduling Core Object Specification (iCalendar)".

[6] IETF RFC 3060: PCIM, "Policy Core Information Model -- Version 1 Specification".

[7] IETF RFC 2591: "Definitions of Managed Objects for Scheduling Management Operations".

[8] DMTF CIM: “Common Information Model” http://www.dmtf.org/spec/cims.html

# 3 Definitions and abbreviations

## 3.1 Definitions

For the purposes of the present document, the terms and definitions given in TS 29.198-1 [1] apply.

## 3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TS 29.198-1 [1] apply, in addition to those given below:

| | |
|---|---|
| CIM | DMTF Common Information Model |
| DMTF | Distributed Management Task Force |
| PCIM | Policy Core Information Model, as defined in RFCs 3060 and 3460. |

# 4 Policy Management SCF

It is expected that more and more OSA services will use policies to express operational criteria. It is also expected that network providers will host policy-enabled services that have been written by 3rd party application service providers. In order to manage policy information, control access to it and to request evaluation of policies a policy management service is needed. Consistent with this, a policy management provisioning manager, IpPolicyManager, and a policy evaluation manager, IpPolicyEvalManager have been defined.

APIs have been defined to offer provisioning services. These include APIs to create, update or view policy information for any policy enabled service. Similarly APIs have been defined to facilitate interactions between clients (e.g. a 3rd party application) and the policies of any policy enabled service. These include APIs to subscribe to policy events, to request evaluation of policies and to request the generation of policy events . All APIs conform to an underlying policy information model that is a derived from the policy core information model defined by the IETF in RFC 3460.

Clients that perform administrative tasks of behalf of a policy enabled service, e.g. create, update or delete policy information must obtain access to IpPolicyManager via the Framework. Administrative tasks may then be performed through methods supported by IpPolicyManager. Similarly, clients that need to invoke evaluation of policies of a specific policy enabled service may do so by obtaining access to IpPolicyEvalManager via the Framework.

Consistent with the above the Policy Management Service supports two classes of service interfaces for policy provisioning and policy evaluation. These are the PM Provisioning SCF and the PM Policy Evaluation SCF respectively.

Examples of policy enabled services include: A load balancing service that uses policies to manage application loads on the network, a charging service that determines charging criteria based on policies, a call management service that uses policies to direct end-user calls to appropriate call agents, etc.

Information in the present document is organized as follows:

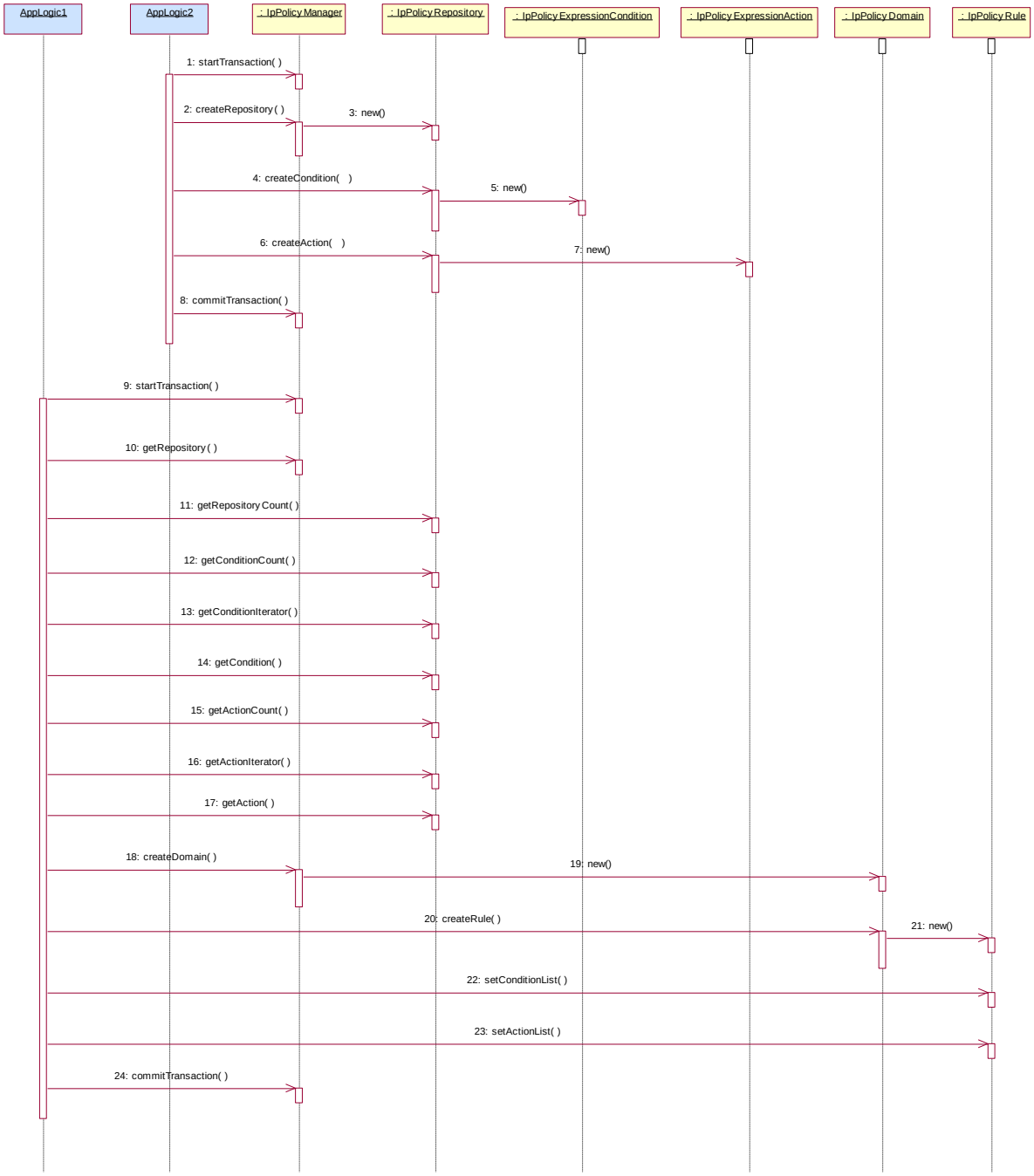
- The Sequence diagrams give the reader a practical idea of how PM provisioning and PM evaluation SCFs are used by clients.
- The Class relationships clause shows relationships between the various interfaces supported by the PM provisioning and PM evaluation SCFs respectively.
- The Interface specification clauses describe in detail each of the interfaces shown within the Class diagram clause.
- The Data Definitions clause shows a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of this specification.

An implementation of this API which supports or implements a method described in the present document, shall support or implement the functionality described for that method, for at least one valid set of values for the parameters of that method. Where a method is not supported by an implementation of a Service interface, the exception P_METHOD_NOT_SUPPORTED shall be returned to any call of that method.

# 5 Sequence Diagrams

## 5.1 Use of Policy Repository

The example shown here shows the use of a Policy Repository. The repository is meant to hold unattached conditions and actions. The Network Operator can populate the repository with the conditions and actions that it can support. These may indeed be based on 'off-line' negotiations with the application developer. The application developer uses the conditions and actions in the Policy Repository to create rules for his own application. In the example application logic represented by AppLogic1 belongs to the Network Operator, whereas the application logic represented by App Logic2 belongs to the ASP. This example uses the same conditions, actions, and rules as the ASP example.




1: The creation of the repository by the Network Operator takes place within one transaction.

2: The method createRepository is invoked on the IpPolicyManager interface to create a new repository.

3: As a result of the createRepository method a new instance of the IpPolicyRepository interface is created. Its interface reference is returned as return parameter of the createRepository method.

4: The Network Operator creates an unattached condition in the new repository by invoking the createCondition method. For simplicity reasons, this is the same condition as in sequence 8 of the ASP example. The same condition attributes apply.

5: A new instance of the IpPolicyExpressionCondition interface is created.

6: The Network Operator creates an unattached action in the repository. Again, this is the same action as in sequence 10 of the ASP example. The same action attributes apply.

7: A new instance of the IpPolicyExpressionAction interface is created.

8: The Network Operator is finished with creating and populating the repository and closes the transaction.

9: Now that a repository exists, the ASP application can open a transaction to start creating a rule based on the conditions and actions stored in the repository.

10: The application invokes the getRepository to obtain a reference to the top-level repository. The returned reference in this case is the reference to the new repository just created by the Network Operator.

11: The application can invoke the getRepositoryCount method on the IpPolicyRepository interface to check whether there are any sub-repositories available. This is not the case for this example.

12: Before trying to obtain all available conditions in this repository the application retrieves the number of conditions by invoking the method getConditionCount.

13: The application can now invoke the getConditioniterator method to obtain the reference to an iterator that contains the names of each of the conditions contained by this repository that the application is authorized to see. As the previous method only return one available condition, this would be only one name, i.e. "SufficientCredit".

14: A reference to the condition can be obtained by invoking getCondition, with the condition name from the iterator as input parameter.

15: Similar to 12.

16: Similar to 13.

17: Similar to 14.

18: At this point in time the application has the names and references to the unattached condition and action from the repository it wants to use to create the rule. First a domain is created by invoking the createDomain method on the IpPolicyManager interface.

19: A new instance of the IpPolicyDomain interface is created.

20: The application invokes createRule to create a rule within the domain that was just created in flow 18 and 19.

21: A new instance of the IpPolicyRule interface is created.

22: By invoking the method setConditionList, the application can now apply the condition from the repository to this rule, by passing the condition reference, obtained by getCondition in flow 14, as an input parameter.

23: Similarly the application can apply the action to the rule by invoking setActionList.

24: Finally, once the rule is created using the condition and action from the policy repository, the transaction can be closed.

## 5.2 Introduce condition and action into rule

This sequence diagram describes how a specific policy rule is managed. A rule consists generally of conditions and of actions, the latter being evaluated if all conditions evaluate to true.

This sequence includes:

- creation of a condition and introduction of it into the rule;

- retrieval of an already defined action object from a repository and introduction into the rule;

- establishing a transaction bracket.

Presumption: the Application got a reference to the group, e.g. by having performed the sequence "create and modify domain" as in clause 5.4.


1: Opens the transaction bracket.

2: creates a rule object in the group by passing the name as parameter. The method returns the reference to the new rule object.

3: Closes the transaction bracket.

4: Opens the transaction bracket.

5: After having created the rule object one can "fill" it with actions and conditions. Here a condition is created on the rule object, thus becoming a part of the rule. Conditions defined in such a way cannot be reused in other rules. For this the repository approach should be used.

Parameters passed are the condition name and the condition type.

Returns a reference to this condition object.

Note that: the type of condition object that is to be created must be one of those specified in TpPolicyConditionType, clause 11.1.4.

The method createCondition() is used to create a new instance of a condition type in the repository or rule. This method passes the name of the condition, the type of the condition and an appropriate set of attribute-value pairs. Note that it is necessary to include, within the conditionAttributes argument of createCondition(), all those attribute-value pairs that are not inherited from IpPolicyCondition - if the inherited attribute-value pairs are included in this argument then their assigned values will override the values assigned prior to this assignment. Thus, for example, if the new condition type to be created is TpPolicyExpressionCondition, then the attribute named "Expression" and its value must be included in conditionAttributes (also see clause 8.1.12). Note that this call may throw an exception if the value of "Expression" is not parsable.

The steps to create an action object instance are similar to those taken to create a condition object instance. We use the method createAction() to create a new action instance. Note that an action object must be one of those specified in TpPolicyActionType, clause 11.1.7. It is necessary to include all the attribute-value pairs that are not inherited from IpPolicyAction, in the actionAttributes argument of createAction() .

6: Closes the transaction bracket.

7: Now we're using the repository approach, i.e. reusable condition or action objects. In this example we reuse an action.

For that purpose we ask at the IpPolicyManager interface for a reference to a named repository.

The repository name is passed.

Returns the reference to the repository.

8: If we know already the name of the action object one retrieves the action directly by passing the name as parameter. Otherwise one has to retrieve the name first by using an action iterator.

Returns a reference to the action object.

9: Opens the transaction bracket.

10: Now, the action(s) must be assigned to the rule. Furthermore and different to the conditions, one has to assign an ordering number to the action.

Passed parameter is the action list, which is a list of action reference/ sequence pairs.

11: After having created or retrieved all needed conditions they must be assigned to the rule. This is done by passing the list of condition to that method.

This is explicitly done by passing TpPolicyConditionList again consisting of TpPolicyConditionListElements which contains the reference the IpPolicyRule object created with message 2.

If the rule is active, this will then cause the expression defined in the condition to be evaluated (as often as necessary). Note that the binding between the variables referenced in the expression and the instances of the variable available is done each time the expression is evaluated. That is, when evaluating a variable reference, each enclosing domain is searched in order (from closest to farthest) for a matching variable. If one is found, it is used. If no matching variable is set, the expression condition fails (evaluates to FALSE).

Activation of actions is done similarly.

12: Closes the transaction bracket.

## 5.3 Create event

This sequence shows how policy events are used.

For clarification we list the different policy related objects used:

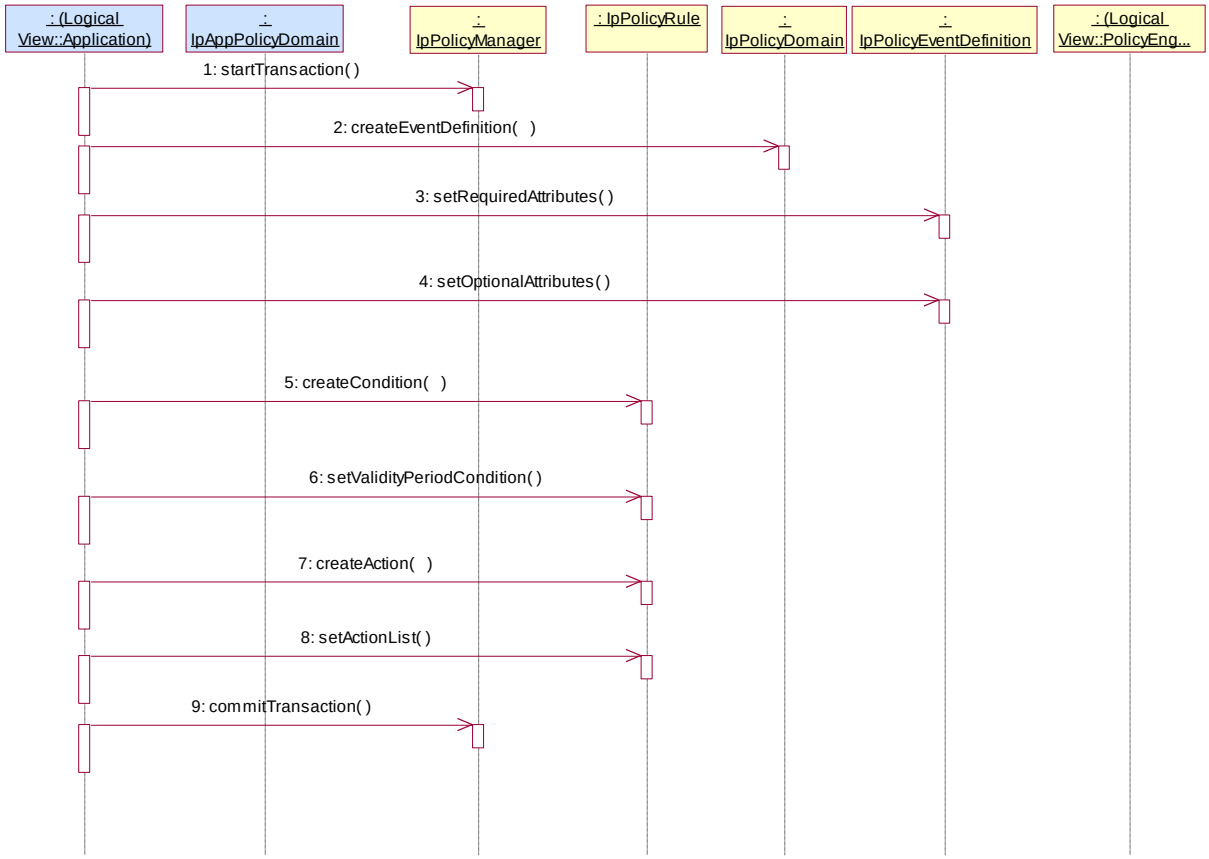
- IpPolicyEventDefinition: The "template" used to define allowable events. The template is used to define formally a distinct type of rule condition and rule action, namely, IpPolicyEventCondition and IpPolicyEventAction.

- IpPolicyEventCondition: A special instance of a policy condition used in a rule. The condition evaluates to "True" on the occurrence of the event instance that is formally associated with it.

- IpPolicyEventAction: A special instance of a policy action used in a rule. The action results in the generation of an instance of the formal event associated with it.

- TpPolicyEvent: This data type is passed as a parameter in the formal notification (to a client) of the occurrence of an instance of an event.

Presumption: the reference to a rule has been somehow retrieved.




1: All changes of policy objects must be performed in a transaction bracket. This method opens the bracket.

2: This method creates a new event type. Event definitions describe the attributes of a specific event class, which can than be instantiated as policy condition or policy event. Returns the reference to the newly created EventDefinition instance which then can be modified according to ones needs.

3: Now, after having created a new instance of a policy event definition, one can set the required attributes by passing the respective attribute set ...

4: ... and the optional attributes. Such attributes may be (...).

5: This createCondition() method creates locally an instance of PolicyTimePeriodCondition defining the validity period of this rule.

Returns a reference to the new instance of IpPolicyTimePeriodCondition object.

Using createCondition() assign the appropriate values to relevant attributes of this new instance of IpPolicyTimePeriodCondition. For example,

TpAttribute.AttributeName = "TimePeriod"

TpAttribute.AttributeValue.SimpleValue.StringValue = "20000101T080000/20000131T120000"

the latter indicating the time period "January 1, 2000, 0800 through January 31, 2000, noon".

6: Using the reference got with createCondition() the validity period is set to rule. Before this created condition will not become valid.

7: The assignment of a policy event is made as for other actions. The difference is the action type passed as parameter: it MUST be of type IpPolicyEventAction.

Passed parameters are the name of the created action, the action type and the attributes of the action; one of these attributes refers by name to the event definition as created before in this sequence.

Returns the reference to the newly created action object.

8: This method activates the action (here the action event) for this rule. After creation this action is not yet active.

The name of the action object is passed.

9: This closes the transaction bracket.

## 5.4 Create and modify domain

This sequence describes how:

- a top-level policy domain is created which is then maintained by the policy manager object;

- a list of domains managed by the policy manager is retrieved and a specific domain is accessed;

- how manipulations on this domain (in this example creation of group and removal of a rule) are performed;

- how the transaction control is initiated.

Presumption: the Application has received a reference to the IpPolicyManager interface.


1: Opens the transaction bracket.

2: Creates a domain by providing the name of the domain object to be created as parameter. The method returns the reference to the domain object.

3: Closes the transaction bracket.

4: The user wants to get all domains handled by the policy manager. This method returns a policy iterator object which can be used to go through the available domains.

5: This method returns the list of domains starting with "index". For efficiency reasons the number of returned entries can be set with the parameter "numberRequested".

6: After having extracted one of the domain name as returned with getList(), the reference to this specific domain get be retrieved by passing the domain name with getDomain(). Returns the domain reference.

7: Opens the transaction bracket.

8: Now, one can act upon the domain, i.e. one can create, modify or delete objects in that domain. Valid objects are domains, groups, and rules.

In this example one creates a group by passing the name of the group to be created with createGroup().

Returns the reference to the new group.

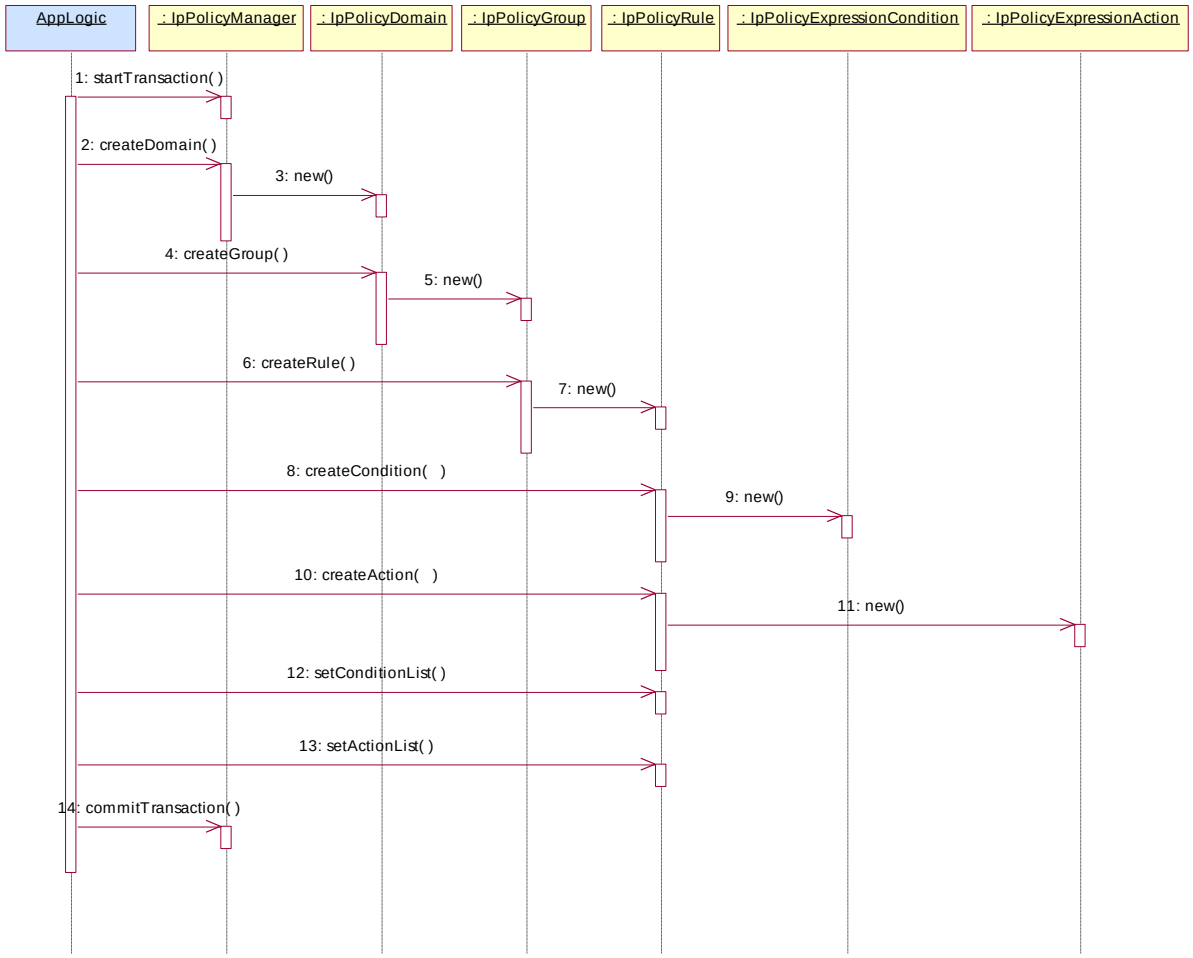
9: Another action is to remove a rule. We assume here that the name of the rule (which is passed as parameter) is already known. Otherwise one has to retrieve the name by using the IpRuleIterator interface (the reference is got with getRuleIterator()).

Returns void.

10: Closes the transaction bracket.

## 5.5 ASP offering services to prepaid subscribers

The example shown here is based on an Application Service Provider (ASP) offering services to the prepaid subscribers of a certain Network Operator. The ASP discovers that, as part of the business logic of the applications it offers, the prepaid credit of the subscriber needs to be verified with regards to the current charge for the service in order to determine whether the purchase should be allowed or not. Rather than including this credit check in the business logic of each and every application that the ASP has in its service portfolio, the ASP may decide to enable a Policy Rule to be hosted in the Policy Engine of the Network Operator.



1: For the sake of this example, all activities to create a Domain, a Group, and the Rule are contained within a single transaction. The method startTransaction is used by the application to open the transaction.

2: The rule in this simplistic example is part of a single group, which in turn is contained within a single domain. The application creates that domain by invoking the method createDomain. The value of the parameter domainName is "eCommerceDomain".

3: As a result of the createDomain method a new instance of the IpPolicyDomain interface is created. Its interface reference is returned as return parameter of the createDomain method.

4: Once the domain is created a group is created within that domain. The application invokes the createGroup method, where the parameter groupName has value "PrePaidGroup".

5: As a result of the createGroup method a new instance of the IpPolicyGroup interface is created. Its interface reference is returned as return parameter of the createGroup method.

6: At this point in time there exists the "PrePaidGroup" group within the "eCommerceDomain" domain. The actual rule can be created, using the method createRule. The parameter ruleName has value "SufficientCreditRule". The new rule SufficientCreditRule has the following attributes:

- Enabled == TRUE; the policy rule is currently enabled.
- RuleUsage == NULL; no free-format usage recommendation is provided.
- Priority == 0; default value, as there is only one rule.
- Mandatory == TRUE; mandatory rule, evaluation of the expression must be attempted.
- PolicyRoles == PrePaidBalanceCheck. Each rule must be assigned a policy role(s).
- ConditionListType == P_PM_DNF; disjunctive normal form (DNF).
- SequencedActions == 3; do not care, as there is only one rule.

7: A new instance of the IpPolicyRule interface is created. createRule returns the reference to this newly created interface.

8: Once an instance of IpPolicyRule exists, the actual policy rule can be constructed by means of conditions and actions. Invoking the method createCondition creates the condition. The parameter conditionName has value "SufficientCredit". The parameter conditionType has value "P_PM_EXPRESSION_CONDITION", to indicate that the condition must satisfy certain expressional syntax. The parameter conditionAttributes is a set of structures. For this example the set contains of only one attribute structure.

- ConditionAttribute.AttributeName = "SufficientCreditExpression".
- ConditionAttribute.AttributeValue.SimpleValue.StringValue = "PrePaidCredit > CurrentCharge".

Note that the variables "PrePaidCredit" and "CurrentCharge" in the expression of AttributeValue are assumed to be defined a priori. The value of the expression is derived from the core grammar expressed in the PM information model.

9: A new instance of the IpPolicyExpressionCondition interface is created.

10: The construction of the rule is completed by creating the action that is to be performed when the condition expression evaluates to TRUE. The parameter actionName has value "PurchaseAllowed". The parameter actionType has value "P_PM_EXPRESSION_ACTION" to indicate that the action must satisfy certain expressional syntax. The actionAttributes are again a set containing of only one structure.

- ActionAttribute.AttributeName = "PurchaseAllowedExpression".
- ActionAttribute.AttributeValue.SimpleValue.StringValue = "AllowedPurchase == TRUE".

11: A new instance of the IpPolicyExpressionAction interface is created.

12: The attributes for the condition are set by invoking the method setConditionList. The conditionList is a list consisting of one structure:

- conditionList.Condition == <reference to the IpPolicyCondition interface returned by 9>.

- conditionList.GroupNumber == 1; indicates how the conditions need to be grouped in DNF or CNF in case more groups of rules exist.

- conditionList.Negated == FALSE.

13: The attributes for the action are set by invoking the method setActionList. The actionList is a list consisting of only one structure:

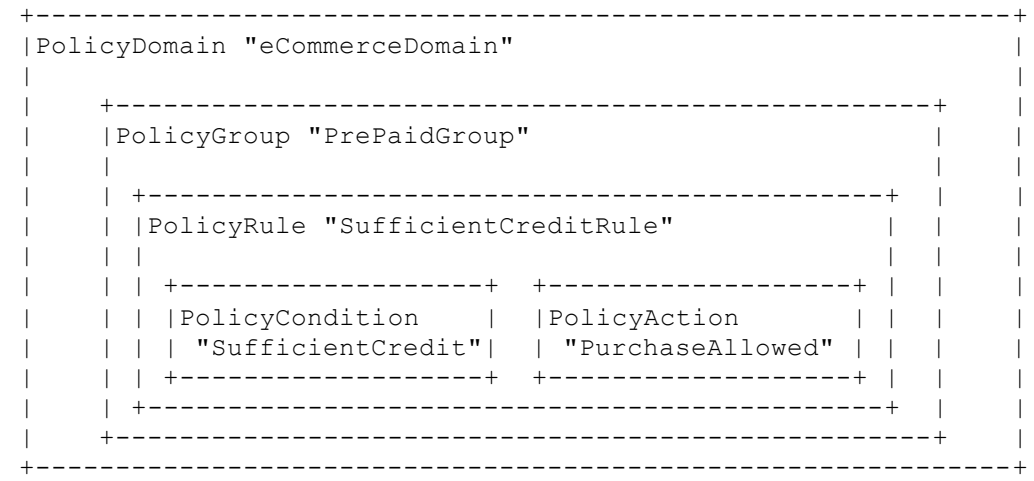
- actionList.Action == <reference to the IpPolicyAction interface returned by step 10>.

- actionList.SequenceNumber == 1.

14: The "SufficientCreditRule" now exists in the "PrePaidGroup" of the "eCommerceDomain" and is assigned the policy role of PrePaidBalanceCheck. The rules is as follows:

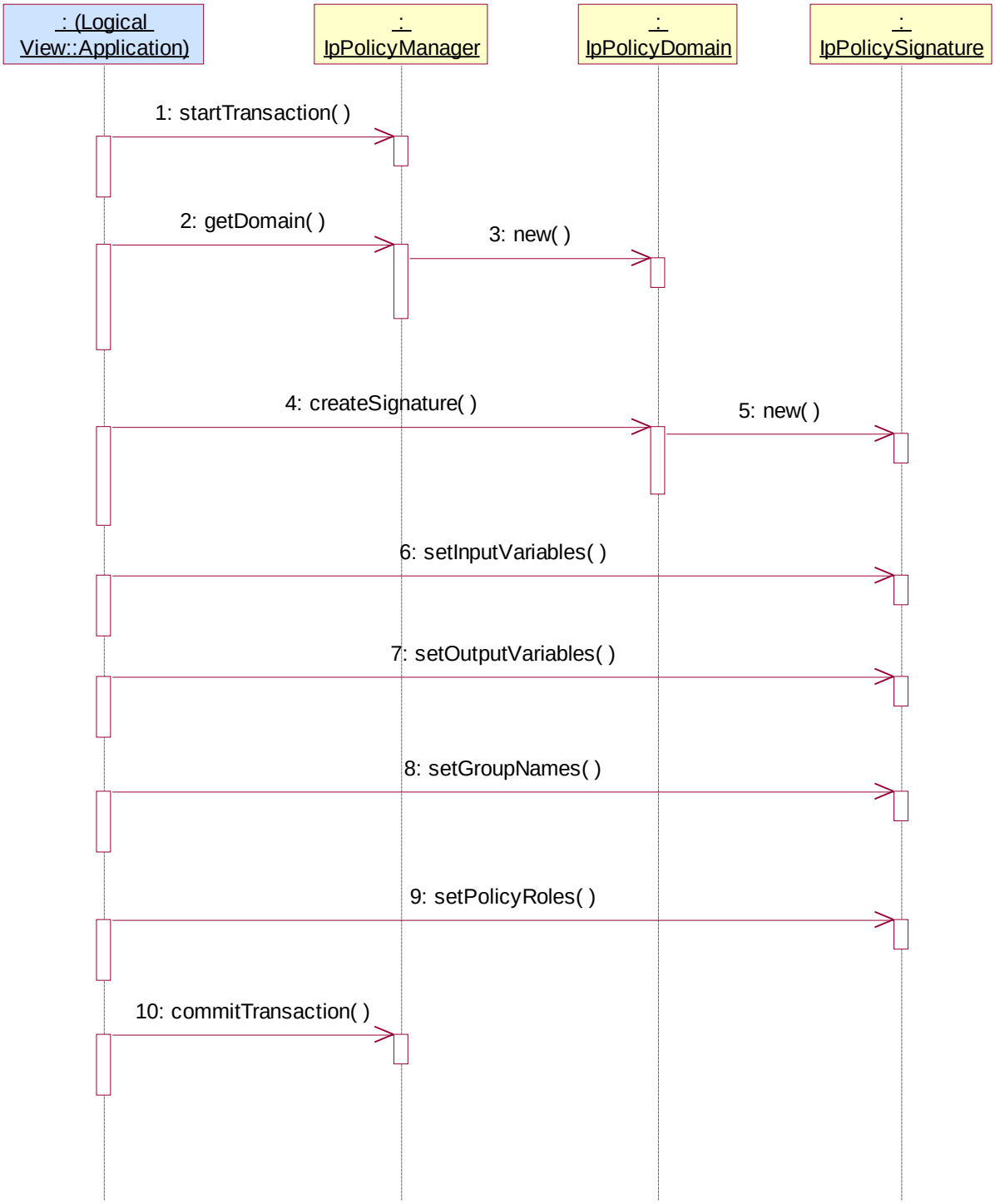
IF " PrePaidCredit > CurrentCharge " THEN "AllowedPurchase == TRUE". This policy rule is enabled upon creation and it is mandatory for the policy engine to load this rule (and any other within the Pre PaidGroup with policy role of PrePaidBalanceCheck) upon an evaluation request and then evaluate it.

The class IpPolicyDomain is defined as a generalized aggregation container, enabling PolicyDomains, PolicyGroups, and PolicyRules to be aggregated in a single container. The following figure shows how this container looks for the example.




## 5.6 Create Signature for an evaluation context

The following sequence diagram shows how a policy signature interface is created within a given (service) domain. A signature is used to establish the context of a policy evaluation request made by a client. A signature definition includes the names of input variables that may be used in the evaluation request. It also includes the name of all output, relevant policy roles and group names. The latter are used to select exactly those groups and rules that are relevant for the evaluation request.




1: All changes are performed in a transaction bracket.

2: This method is used by the client to navigate to the relevant domain.

4: A new Signature is created within the chosen domain.

6: Names of all input variables associated with this signature are specified.

7: Names of all output variables associated with the signature are specified.

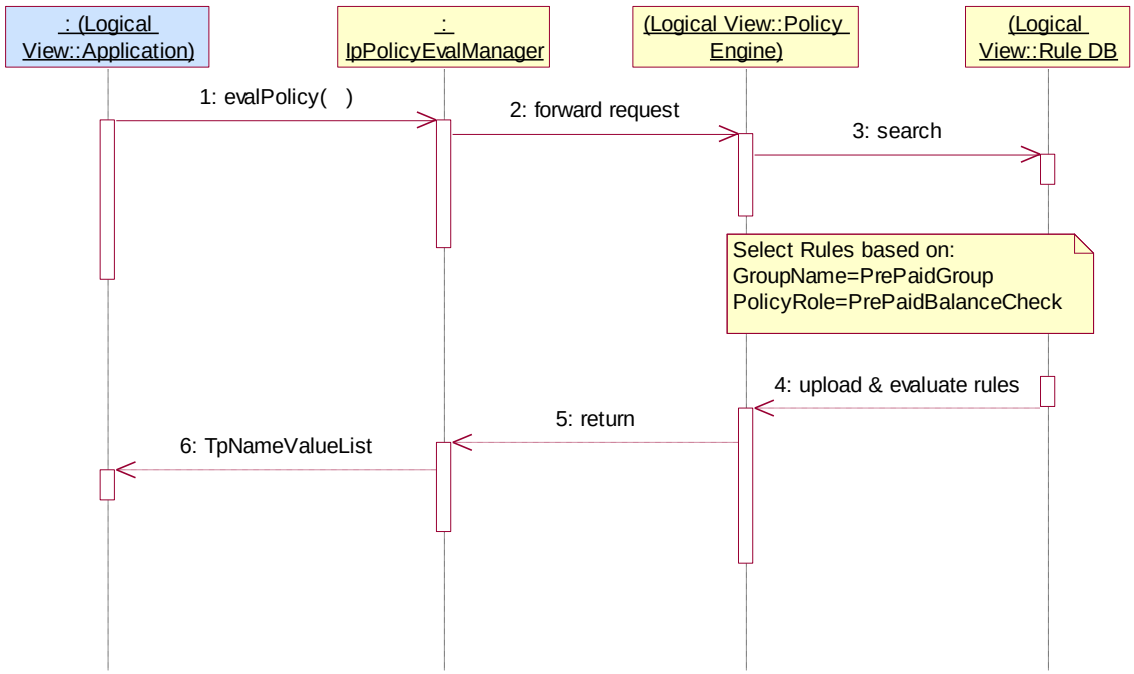
8: Name of the rule Group(s) relevant to the request is specified. This could be a NULL collection but it is advisable to specify a name(s) for performance reasons.

9: Name of policy roles that are used to select rules relevant to the request. This could be a NULL collection but it is advisable to specify policy role name(s).

10: Transaction is committed. At this stage a new signature has been created under the selected domain.

## 5.7 Request Evaluation of Policies

The following sequence diagram shows how a client may request the evaluation of policies associated with a pre-paid service. Assume that rules have been defined as in example 5.5 and a signature has been defined for the "eCommerceDomain" (see example 5.6). Note that a client needs to access IpPolicyEvalManager in-order to request rule evaluation.




1: Make a policy evaluation request via evalPolicy(). Note that parameters for the method include domain name (following 5.5 this is "eCommerceDomain"), signature name and a (sub) set of input variables. Note that, as in example 5.6, the signature determines the context of the request. In this instance, the request is made to evaluate rules in the PrePaid group using the rule(s) whose policy role has been specified as PrePaidBalance Check.

2: The request is forwarded to the Rules Engine.

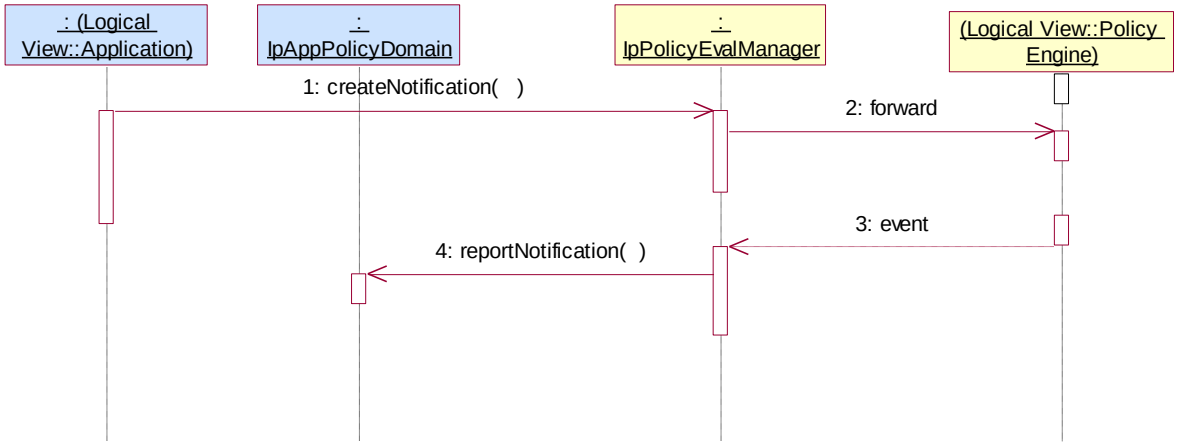
3: The Rules Engine uploads relevant rules from the rules database.

4: Rules Engine evaluates uploaded rules.

6: Results, i.e, output variable name-value pair(s) are returned to the client in TpNameValueList. Following example 5.5 the output variable name is AllowedPurchase and its value is TRUE.

## 5.8 Register for and Receive Notification of a Policy Event

The following sequence diagram show how a client subscribes to a policy event and receives notification when the event is triggered. We assume that the policy event has been defined (for a specific domain) as in example 5.3. Assume, in this case, that the event is triggered when the action part of a rule fires. This may happen when, e.g. a pre-defined threshold (say, a credit limit) is reached causing the conditions of a policy rule to be satisfied thus resulting in the action part to be executed.




1: Subscribe to a policy event in a domain of choice. Note that the parameters of createNotification are: domain name, the call back address of the client application. Immediately after createNotification is invoked a TpAssignmentID is returned to the client (flow not show in the diagram). The client uses this ID to identify its subscription to the policy event of choice.

4: When the policy event is triggered, a notification of the event is sent to every client that subscribed to the event.

# 6 Class Diagrams

Policy Management (PM) comprises of the following SCFs':

- Policy Management Provisioning Service whose Interfaces are used to define policy information, e.g. policy rules, policy events, etc., and to update and view this information.
- Policy Management Policy Evaluation Service whose Interfaces are used to request evaluation of policies and for subscription to policy events & to receive to notification of these.

## 6.1 PM Provisioning SCF Class Diagrams

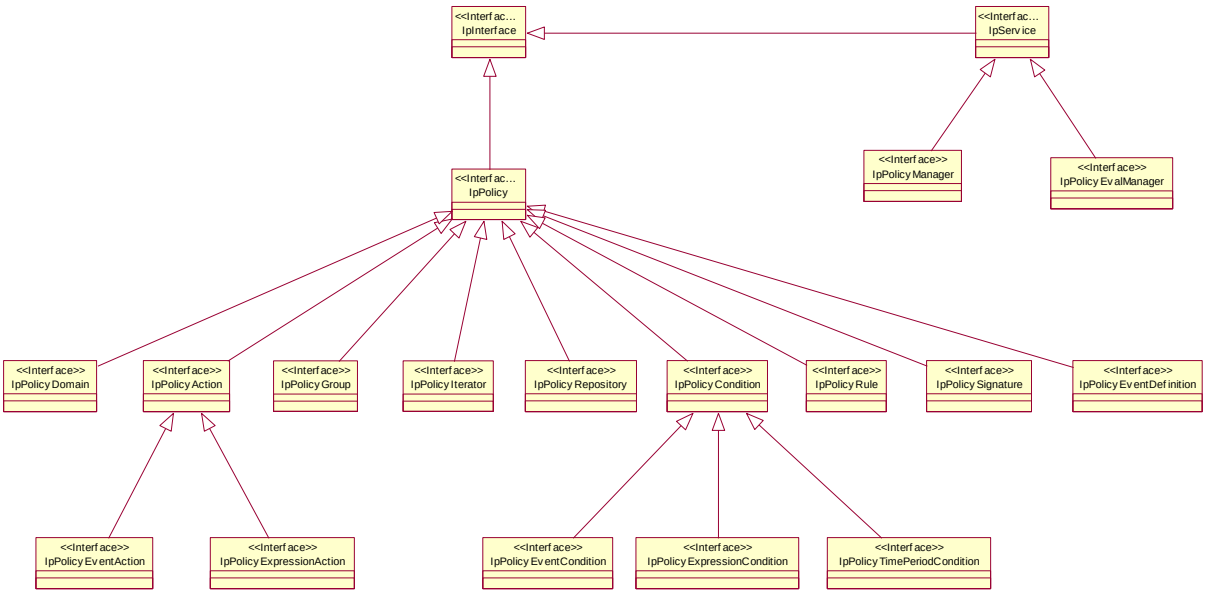



**Figure: Policy Provisioning Classes**

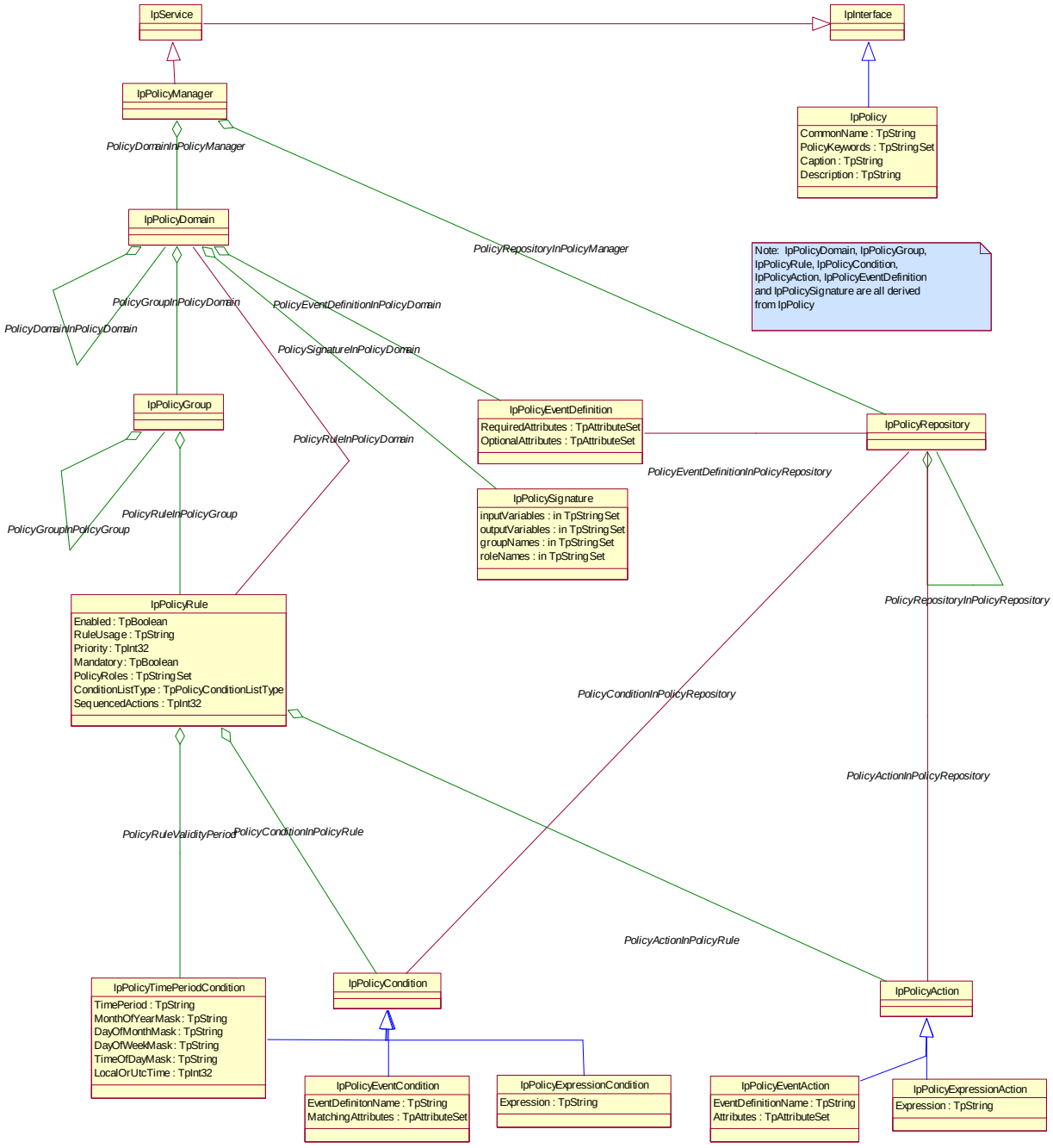



**Figure: Policy Management Information Model**

## 6.2 PM Policy Evaluation SCF Class Diagrams

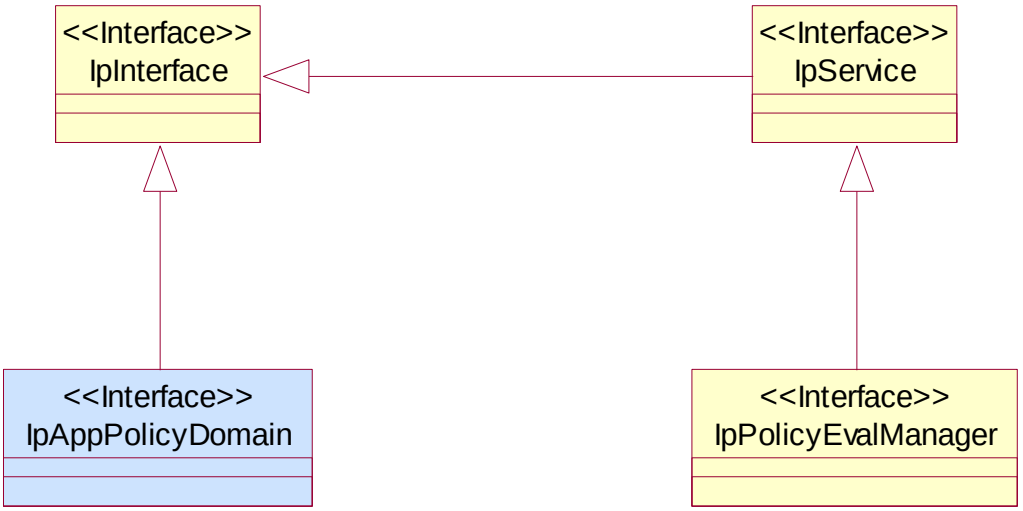



**Figure: Policy Evaluation Classes**

# 7 The Service Interface Specifications

## 7.1 Interface Specification Format

This clause defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

### 7.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name IpSvc<name>, while the Framework interfaces are denoted by classes with name IpFw<name>.

### 7.1.2 Method descriptions

Each method (API method "call") is described. Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a '`Req`' suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a '`Res`' or '`Err`' suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name>` or `IpSvc<name>` interfaces to provide the callback mechanism.

### 7.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as 'in' represent those that must have a value when the method is called. Those described as 'out' are those that contain the return result of the method when the method returns.

### 7.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

## 7.2 Base Interface

### 7.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.

| <<Interface>><br>IpInterface |
|---|
| |
| |

## 7.3 Service Interfaces

### 7.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

## 7.4 Generic Service Interface

### 7.4.1 Interface Class IpService

Inherits from: IpInterface.

All service interfaces inherit from the following interface.

| <<Interface>><br>IpService |
|---|
| |
| setCallback (appInterface : in IpInterfaceRef) : void<br>setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : void |

#### 7.4.1.1 Method setCallback()

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application. It is not allowed to invoke this method on an interface that uses SessionIDs. Multiple invocations of this method on an interface shall result in multiple callback references being specified. The SCS shall use the most recent callback interface provided by the application using this method. In the event that a callback reference fails or is no longer available, the next most recent callback reference available shall be used.

*Parameters*

**appInterface : in IpInterfaceRef**

Specifies a reference to the application interface, which is used for callbacks.

*Raises*

**TpCommonExceptions, P_INVALID_INTERFACE_TYPE**

#### 7.4.1.2 Method setCallbackWithSessionID()

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg. It is not allowed to invoke this method on an interface that does not use SessionIDs. Multiple invocations of this method on an interface shall result in multiple callback references being specified. The SCS shall use the most recent callback interface provided by the application using this method. In the event that a callback reference fails or is no longer available, the next most recent callback reference available shall be used.

*Parameters*

**appInterface : in IpInterfaceRef**

Specifies a reference to the application interface, which is used for callbacks.

**sessionID : in TpSessionID**

Specifies the session for which the service can invoke the application's callback interface.

*Raises*

**TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_INTERFACE_TYPE**

# 8 Policy Management (PM) Interface Classes

## 8.1 PM Provisioning SCF Interface Classes

The Policy Management provisioning APIs address the following :

· The creation, modification and viewing of policy information.

Generally, policy enabled services will be created by a network service provider. A policy service may also be created by an application service provider (ASP) and hosted in the network. Such services need not be based on published OSA specifications. However, they will be created using OSA policy management APIs, will conform to the OSA policy information model and will be accessible via OSA defined interfaces.

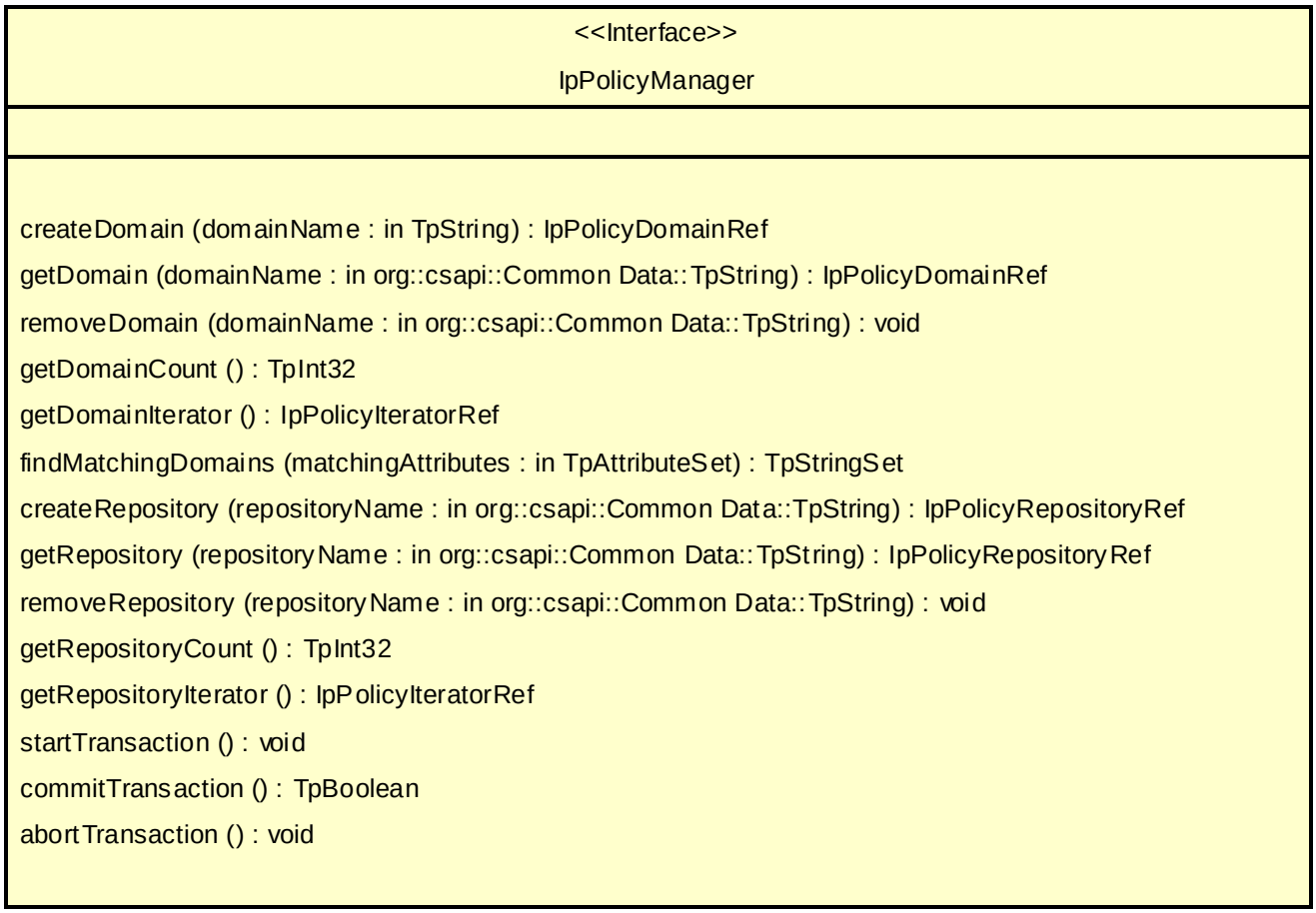
### 8.1.1 Interface Class IpPolicyManager

Inherits from: IpService.

Clients that wish to participate in Policy Management obtain a reference to an instance of the IpPolicyManager interface from the Framework. Using this reference, clients can obtain a reference to a policy domain of interest, iterate through the names of all policy domains, create a new policy domain, or remove an existing one. Clients can also obtain a reference to a policy repository, iterate through the names of all policy repositories, create a new policy repository or remove an existing one.

Note that all operations through Policy Management interfaces are subject to authorization checks - clients will only

have permission to invoke methods as are allowed by the client's privileges as established by a prior agreement between the owner of the client and the owner of the policy management complex. Similarly, methods will only return data that the client is authorized to see. For example, if the client is authorized to see some of the top-level domains and not others, the IpPolicyIterator returned by getDomainIterator() will only return those domains that the client is authorized for.




### 8.1.1.1 Method createDomain()

Create the specified top-level Policy Domain and get a reference to the new instance.

Returns a reference to the domain just created.

*Parameters*

**domainName : in TpString**

The name of the domain to create.

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

#### 8.1.1.2 Method getDomain()

Get a reference to the specified top-level Domain.

Returns the reference to the domain.

*Parameters*

**domainName : in org::csapi::Common Data::TpString**

The name of the domain.

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

#### 8.1.1.3 Method removeDomain()

Remove the specified top-level domain.

*Parameters*

**domainName : in org::csapi::Common Data::TpString**

The name of the top-level domain to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

#### 8.1.1.4 Method getDomainCount()

Returns the number of top-level Policy Domains contained by the PolicyManager that the client is authorized to see.

Returns the number of domains.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.1.5 Method getDomainIterator()

Obtain a reference to an iterator that will return the names of each of the top-level Policy Domains known to the PolicyManager that the client is authorized to see.

Returns the reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.1.6 Method findMatchingDomains()

Ask for the set of domains that contain attributes that match the specified set of attributes that the client is authorized to see. This could be used, for example, to get a list of all of the domains whose 'Role' is 'QOS'.

Returns the names of the matching top-level domains.

*Parameters*

**matchingAttributes : in TpAttributeSet**

*Returns*

**TpStringSet**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.1.7 Method createRepository()

Create the specified top-level Policy Repository and get a reference to the new instance.

Returns a reference to the repository just created.

*Parameters*

**repositoryName : in org::csapi::Common Data::TpString**

The name of the Repository to create.

*Returns*

**IpPolicyRepositoryRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.1.8 Method getRepository()

Get a reference to the specified top-level repository.

Returns a reference to the repository.

*Parameters*

**repositoryName : in org::csapi::Common Data::TpString**

The name of the repository.

*Returns*

**IpPolicyRepositoryRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.1.9 Method removeRepository()

Remove the specified top-level Policy Repository.

*Parameters*

**repositoryName : in org::csapi::Common Data::TpString**

The name of the top-level Repository to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.1.10 Method getRepositoryCount()

Returns the number of top-level Policy Repositories contained by the PolicyManager that the client is authorized to see.

Returns: The number of repositories.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.1.11 Method getRepositoryIterator()

Obtain a reference to an iterator that will return the names of each of the top-level Policy Repositories known to the PolicyManager that the client is authorized to see.

Returns: The reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.1.12 Method startTransaction()

Open a transaction. All modifications to the policy information base up to the call to either commitTransaction() or abortTransaction() will be treated as part of this transaction.

Note that transaction brackets consisting of startTransaction() and commitTransaction() are generally used to perform changes in an atomic way, i.e. to ensure that either all changes are made persistent or all changes are undone in case of failure of even a single action. Any other clients reading data modified by this transaction will see the existing data until commitTransaction() is called. Any timeouts of this transaction are implementation specific. If a transaction is timed out, any subsequent attempt to make requests that require a transaction will throw the exception P_NO_TRANSACTION_IN_PROCESS.

Note, however, that the scope of transaction brackets is extended here: Large transaction brackets can be also useful for efficiency reasons even if the different actions are not atomic. Creation of a transaction introduces a significant overhead, reduction of the number of separate transactions reduces this. It is up to the application implementation to reflect this fact.

Note that transactions can not be nested, that is, a second call to startTransaction() without calling commitTransaction() or abortTransaction() in between will result in the exception P_TRANSACTION_IN_PROCESS being thrown during the second call.

*Parameters*

No Parameters were identified for this method

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_TRANSACTION_IN_PROCESS**

#### 8.1.1.13 Method commitTransaction()

Commit a transaction. All modifications to the policy information base made since the last call to startTransaction() will be committed.

Returns: TRUE is returned if the commit succeeded and the policy information base has been updated, FALSE otherwise.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpBoolean**

*Raises*

**TpCommonExceptions, P_NO_TRANSACTION_IN_PROCESS**

#### 8.1.1.14 Method abortTransaction()

Abort a transaction. All modifications to the policy information base made since the last call to startTransaction() will be discarded.

*Parameters*

No Parameters were identified for this method

*Raises*

**TpCommonExceptions, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.2 Interface Class IpPolicy

Inherits from: IpInterface.

The base interface from which are derived all of the Policy interfaces (except IpPolicyManager). This interface documents four attributes for describing a policy-related instance. In the same way that the generic attribute accessor methods are defined in this base interface, these common attributes are documented here as well and each interface that is derived from IpPolicy will provide support for them.

Note that we could have defined dedicated get/set methods for each attribute, which would have the benefits of being potentially faster and safer, but this design approach was not taken, primarily to make it simpler to add additional attributes in the future without having to change the associated Interface.

| <<Interface>><br>IpPolicy |
| --- |
| |
| getAttribute (attributeName : in TpString) : TpAttribute<br>setAttribute (targetAttribute : in TpAttribute) : void<br>getAttributes (attributeNames : in TpStringList) : TpAttributeSet<br>setAttributes (targetAttributes : in TpAttributeSet) : void |

### 8.1.2.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

### 8.1.2.2 Method getAttribute()

Get a copy of the specified attribute from the policy object. Note that modifying the returned attribute will not update the actual attribute of the object. See setAttribute() for that functionality.

Returns: A copy of the attribute.

*Parameters*

**attributeName : in TpString**

The name of the attribute to retrieve.

*Returns*

**TpAttribute**

*Raises*

**TpCommonExceptions, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.2.3 Method setAttribute()

Set an attribute of a policy object.

*Parameters*

**targetAttribute : in TpAttribute**

The attribute to be set in this object.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.2.4 Method getAttributes()

Get a copy of the set of attributes for the policy object. Note that modifying the returned set will not update the actual attributes of the object. See setAttributes() for that functionality.

Returns: A copy of the attributes.

*Parameters*

**attributeNames : in TpStringList**

The list of names of the attributes to retrieve. In case the list of names is null or empty, all of the attributes will be returned.

*Returns*

**TpAttributeSet**

*Raises*

**TpCommonExceptions**

### 8.1.2.5 Method setAttributes()

Set one or more attributes of a policy object.

*Parameters*

**targetAttributes : in TpAttributeSet**

The attributes to be set in this object.

*Raises*

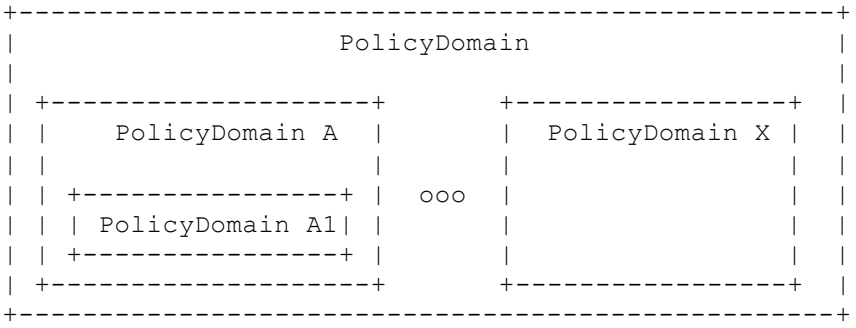
**TpCommonExceptions, P_ACCESS_VIOLATION, P_NO_TRANSACTION_IN_PROCESS**

## 8.1.3 Interface Class IpPolicyDomain

Inherits from: IpPolicy.

This class is a generalized aggregation container. It enables PolicyDomains, PolicyGroups, PolicyRules, or PolicyEventDefinitions to be aggregated in a single container. Loops, including the degenerate case of a PolicyDomain that contains itself, are not allowed when PolicyDomains contain other PolicyDomains.

PolicyDomains and their nesting capabilities are shown in the figure below. Note that a PolicyDomain can nest other PolicyDomains, and there is no restriction on the depth of the nesting in sibling PolicyDomains.




As a simple example, think of the highest level PolicyDomain shown in the figure above as a PolicyDomain for the Call Control Service. This PolicyDomain may be called CallControlPolicy, and may aggregate several PolicyDomains that provide specialized rules per client application.

Hence, PolicyDomain A in the figure above may define call control rules for a third party application from company A, while another PolicyDomain might define rules for third party application B (e.g. PolicyDomain X), and so forth.

Note also that the depth of each PolicyDomain does not need to be the same. Thus, the ApplicationAPolicyDomain might have several additional layers of PolicyDomains defined for any of several reasons (different locales, number of customers, etc.). The PolicyRules are therefore contained at n levels from the ApplicationAPolicyDomain. Compare this to the Application B PolicyDomain (PolicyDomain X), which might directly contain PolicyRules.

<<Interface>>

IpPolicyDomain

getParentDomain () : IpPolicyDomainRef

createDomain (domainName : in TpString) : IpPolicyDomainRef

getDomain (domainName : in TpString) : IpPolicyDomainRef

removeDomain (domainName : in TpString) : void

getDomainCount () : TpInt32

getDomainIterator () : IpPolicyIteratorRef

createGroup (groupName : in TpString) : IpPolicyGroupRef

getGroup (groupName : in TpString) : IpPolicyGroupRef

removeGroup (groupName : in TpString) : void

getGroupCount () : TpInt32

getGroupIterator () : IpPolicyIteratorRef

createRule (ruleName : in TpString) : IpPolicyRuleRef

getRule (ruleName : in TpString) : IpPolicyRuleRef

removeRule (ruleName : in TpString) : void

getRuleCount () : TpInt32

getRuleIterator () : IpPolicyIteratorRef

createEventDefinition (eventDefinitionName : in TpString, requiredAttributes : in TpStringSet, optionalAttributes : in TpStringSet) : IpPolicyEventDefinitionRef

getEventDefinition (eventDefinitionName : in TpString) : IpPolicyEventDefinitionRef

removeEventDefinition (eventDefinitionName : in TpString) : void

getEventDefinitionCount () : TpInt32

getEventDefinitionIterator () : IpPolicyIteratorRef

createVariableSet (variableSetName : in TpString) : void

getVariableSet (variableSetName : in TpString) : TpPolicyVarSet

removeVariableSet (variableSetName : in TpString) : void

getVariableSetCount () : TpInt32

getVariableSetIterator () : IpPolicyIteratorRef

createVariable (variableSetName : in TpString, variableName : in TpString, variableType : in TpPolicyType) : void

setVariableValue (variableSetName : in TpString, variableName : in TpString, variableValue : in TpAny) : void

getVariableType (variableSetName : in TpString, variableName : in TpString) : TpPolicyType

getVariableValue (variableSetName : in TpString, variableName : in TpString) : TpAny

getVariable (variableSetName : in TpString, variableName : in TpString) : TpPolicyVar

removeVariable (variablSetName : in TpString, variableName : in TpString) : void

createSignature (signatureName : in TpString) : IpPolicySignatureRef

getSignature (signatureName : in TpString) : IpPolicySignatureRef

removeSignature (signatureName : in TpString) : void

getSignatureCount () : TpInt32

getSignatureIterator () : IpPolicyIteratorRef

#### 8.1.3.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

**Role : TpString**

This attribute provides a way to specify higher-level context associated with a top-level domain, e.g. Role = Charging, Role = QOS, or Role = User Interaction, etc. This attribute can be used to search for domains that specify a particular Role by using the findMatchingDomains() method of the IpPolicyManager interface. This attribute must be explicitly set for each instance of an IpPolicyDomain. There is no default and values are not copied from the parent domain (if any).

**Owner : TpString**

This attribute provides a way to specify an owner of a top-level domain. This attribute can be used to search for domains that specify a particular Owner by using the findMatchingDomains() method of the IpPolicyManager interface. This attribute must be explicitly set for each instance of an IpPolicyDomain. There is no default and values are not copied from the parent domain (if any).

### 8.1.3.2 Method getParentDomain()

Return a reference to the domain that contains this one (if any). If this is a top-level domain, return a NULL reference.

Returns: A reference to the parent domain.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions**

### 8.1.3.3 Method createDomain()

Create the specified domain and get a reference to the new instance.

Returns: A reference to the domain just created.

*Parameters*

**domainName : in TpString**

The name of the domain to create.

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.4 Method getDomain()

Get a reference to the specified subdomain.

Returns: A reference to the domain.

*Parameters*

**domainName : in TpString**

The name of the subdomain to get.

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.5 Method removeDomain()

Remove the specified subdomain.

*Parameters*

**domainName : in TpString**

The name of the subdomain to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.6 Method getDomainCount()

Returns the number of subdomains contained by this one that the client is authorized to see.

Returns: The number of subdomains.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.3.7 Method getDomainIterator()

Obtain a reference to an iterator that will return the names of each of the subdomains contained by this one that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

***The following methods are for Rule Group Management in Domain :***

### 8.1.3.8 Method createGroup()

Create the specified group and get a reference to the new instance.

Returns: A reference to the group just created.

*Parameters*

**groupName : in TpString**

The name of the group to create.

*Returns*

**IpPolicyGroupRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.9 Method getGroup()

Get a reference to the specified group.

Returns: A reference to the group.

*Parameters*

**groupName : in TpString**

The name of the group to get.

*Returns*

**IpPolicyGroupRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.10 Method removeGroup()

Remove the specified group.

*Parameters*

**groupName : in TpString**

The name of the group to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.11 Method getGroupCount()

Returns the number of groups contained by this domain that the client is authorized to see.

Returns: The number of groups.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.3.12 Method getGroupIterator()

Obtain a reference to an iterator that will return the names of each of the groups contained by this domain that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

***The following methods are for rule Management within a Domain:***

### 8.1.3.13 Method createRule()

Create a rule with the specified name, and get a reference to the new instance.

Returns: A reference to the just created rule.

*Parameters*

**ruleName : in TpString**

The name of the rule to create.

*Returns*

**IpPolicyRuleRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.14 Method getRule()

Get a reference to the specified rule.

Returns: A reference to the rule.

*Parameters*

**ruleName : in TpString**

The name of the rule to get.

*Returns*

**IpPolicyRuleRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.15 Method removeRule()

Remove the specified rule.

*Parameters*

**ruleName : in TpString**

The name of the rule to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.16 Method getRuleCount()

Returns the number of rules contained by this domain that the client is authorized to see.

Returns: The number of rules.

*Parameters*
No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.3.17 Method getRuleIterator()

Obtain a reference to an iterator that will return the names of each of the rules contained by this domain that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*
No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

***The following methods are for event management within a Domain:***

### 8.1.3.18 Method createEventDefinition()

Define a new event type, specifying the definition's name and the required and optional attributes that must/may appear in an instance of that event.

Returns: A reference to the newly created definition.

*Parameters*

**eventDefinitionName : in TpString**

The name of the definition of the new event.

**requiredAttributes : in TpStringSet**

The set of attributes that MUST be included in any event of this type.

**optionalAttributes : in TpStringSet**

A set of attributes that MAY be included in any event of this type.

*Returns*

**IpPolicyEventDefinitionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.19 Method getEventDefinition()

Get a reference to the definition of an event type.

Returns: A reference to the definition.

*Parameters*

**eventDefinitionName : in TpString**

The name of the event definition to get.

*Returns*

**IpPolicyEventDefinitionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.20 Method removeEventDefinition()

Remove the definition for an event from the domain.

*Parameters*

**eventDefinitionName : in TpString**

The name of the definition to remove.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.21 Method getEventDefinitionCount()

Returns the number of event definitions contained by this domain that the client is authorized to see.

Returns: The number of event definitions.

*Parameters*
No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.3.22 Method getEventDefinitionIterator()

Obtain a reference to an iterator that will return the names of each of the definitions contained by this domain that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*
No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

***The following methods are for variable management within a Domain:***

### 8.1.3.23 Method createVariableSet()

Used by clients to define a named collection of variables. Variables are attributes that can be updated by the client to reflect the current 'state' of the client. Since variables can be referenced by name from expression conditions and actions, the act of updating a variable may have a side effect of satisfying conditions in rules that are currently active. Variables that are defined by the network operator may be dynamically updated by the policy engine to reflect the current 'state' of the modelled networks and services.

*Parameters*

**variableSetName : in TpString**

The name of the new variable set.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.24 Method getVariableSet()

Get a variable set.

Returns: A variable set.

*Parameters*

**variableSetName : in TpString**

The name of the variable set to get.

*Returns*

**TpPolicyVarSet**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.25 Method removeVariableSet()

Remove the variable set from the domain.

*Parameters*

**variableSetName : in TpString**

The name of the variable set to remove.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.26 Method getVariableSetCount()

Returns the number of variable sets contained by this domain that the client is authorized to see.

Returns: The number of variable sets.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.3.27 Method getVariableSetIterator()

Obtain a reference to an iterator that will return the names of each of the variable sets contained by this domain that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*
No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.3.28 Method createVariable()

Create a variable within a variable set.

*Parameters*

**variableSetName : in TpString**

The name of the variable set within which to set the specified variable.

**variableName : in TpString**

The name of the variable to being created.

**variableType : in TpPolicyType**

The type of the variable being created.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.29 Method setVariableValue()

Set a variable value within a variable set.

*Parameters*

**variableSetName : in TpString**

The name of the variable set within which to set the specified variable value.

**variableName : in TpString**

The name of the variable being set.

**variableValue : in TpAny**

The value of the variable being created.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.30 Method getVariableType()

Get a copy of the type of a variable from a variable set.

Returns: A copy of the variable type.

*Parameters*

**variableSetName : in TpString**

The name of the variable set in which to find the variable.

**variableName : in TpString**

The name of the variable whose type is to be retrieved.

*Returns*

**TpPolicyType**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.31 Method getVariableValue()

Get a copy of a variable value from a variable set.

Returns: A copy of the variable value.

*Parameters*

**variableSetName : in TpString**

The name of the variable set to find the variable in.

**variableName : in TpString**

The name of the variable whose value is to be retrieved.

*Returns*

**TpAny**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.32 Method getVariable()

Get a copy of a variable from a variable set.

Returns: A copy of the variable (i.e. a copy of its type and value).

*Parameters*

**variableSetName : in TpString**

The name of the variable set to find the variable in.

**variableName : in TpString**

The name of the variable to get a copy of.

*Returns*

**TpPolicyVar**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.33 Method removeVariable()

Remove a variable from a variable set.

*Parameters*

**variablSetName : in TpString**

The name of the variable set from where to remove the variable.

**variableName : in TpString**

The name of the variable to be removed.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

***The following methods are for signature management within a Domain:***

### 8.1.3.34 Method createSignature()

Define a new policy-evaluation method signature, specifying the signature's name.

Returns: A reference to the newly created definition.

*Parameters*

**signatureName : in TpString**

The name of the new policy-evaluation method signature.

*Returns*

**IpPolicySignatureRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.35 Method getSignature()

Get a reference to the signature for a policy-evaluation method signature.

Returns: A reference to the definition.

*Parameters*

**signatureName : in TpString**

The name of the policy-evaluation method signature to get.

*Returns*

**IpPolicySignatureRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.3.36 Method removeSignature()

Remove the policy-evaluation method signature from the domain.

*Parameters*

**signatureName : in TpString**

The name of the signature to remove.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.3.37 Method getSignatureCount()

Returns the number of policy-evaluation signatures contained in this domain that the client is authorized to see.

Returns: The number of signatures.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions**

### 8.1.3.38 Method getSignatureIterator()

Obtain a reference to an iterator that will return the names of each of the policy-evaluation signatures contained in this domain that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

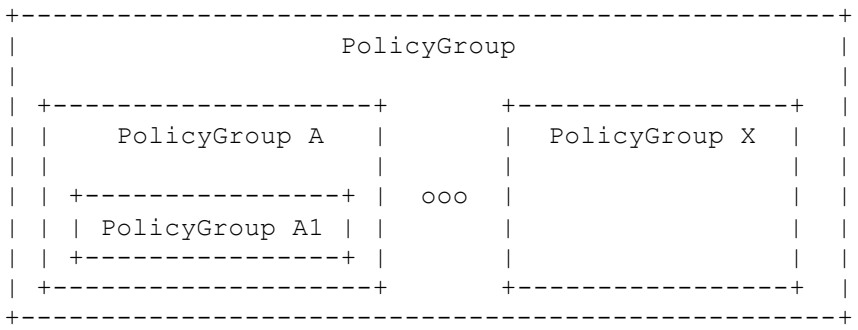
**TpCommonExceptions, P_ACCESS_VIOLATION**

## 8.1.4 Interface Class IpPolicyGroup

Inherits from: IpPolicy.

This class is a generalized aggregation container. It enables either PolicyRules or PolicyGroups to be aggregated in a single container. Loops, including the degenerate case of a PolicyGroup that contains itself, are not allowed when PolicyGroups contain other PolicyGroups.

PolicyGroups and their nesting capabilities are shown in the figure below. Note that a PolicyGroup can nest other PolicyGroups, and there is no restriction on the depth of the nesting in sibling PolicyGroups.




As a simple example, think of the highest level PolicyGroup shown in the figure above as a logon policy or US employees of a company. This PolicyGroup may be called USEmployeeLogonPolicy, and may aggregate several PolicyGroups that provide specialized rules per location.

Hence, PolicyGroup A in the figure above may define logon rules for employees on the West Coast, while another PolicyGroup might define logon rules for the Midwest (e.g. PolicyGroup X), and so forth.

Note also that the depth of each PolicyGroup does not need to be the same. Thus, the WestCoast PolicyGroup might have several additional layers of PolicyGroups defined for any of several reasons (different locales, number of subnets, etc.). The PolicyRules are therefore contained at n levels from the USEmployeeLogonPolicyGroup. Compare this to the Midwest PolicyGroup (PolicyGroup X), which might directly contain PolicyRules. No attributes are defined for this class since it inherits all its attributes from IpPolicy. The class exists to aggregate PolicyRules or other PolicyGroups.

<<Interface>>

IpPolicyGroup

getParentDomain () : IpPolicyDomainRef

getParentGroup () : IpPolicyGroupRef

createGroup (groupName : in TpString) : IpPolicyGroupRef

getGroup (groupName : in TpString) : IpPolicyGroupRef

removeGroup (groupName : in TpString) : void

getGroupCount () : TpInt32

getGroupIterator () : IpPolicyIteratorRef

createRule (ruleName : in TpString) : IpPolicyRuleRef

getRule (ruleName : in TpString) : IpPolicyRuleRef

removeRule (ruleName : in TpString) : void

getRuleCount () : TpInt32

getRuleIterator () : IpPolicyIteratorRef

#### 8.1.4.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

### 8.1.4.2 Method getParentDomain()

Get a reference to the domain that directly contains this group (if any). If this is a subgroup (whose immediate container is another group instead of a domain), return a NULL reference.

Returns: A reference to the containing domain.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions**

### 8.1.4.3 Method getParentGroup()

Return a reference to the group that contains this one (if any). If this is a top-level group, return a NULL reference.

Returns: A reference to the containing group.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyGroupRef**

*Raises*

**TpCommonExceptions**

### 8.1.4.4 Method createGroup()

Create the specified group and get a reference to the new instance.

Returns: A reference to the group just created.

*Parameters*

**groupName : in TpString**

The name of the group to create.

*Returns*

**IpPolicyGroupRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.4.5 Method getGroup()

Get a reference to the specified group.

Returns: A reference to the group.

*Parameters*

**groupName : in TpString**

The name of the group to get.

*Returns*

**IpPolicyGroupRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.4.6 Method removeGroup()

Remove the specified group.

*Parameters*

**groupName : in TpString**

The name of the group to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.4.7 Method getGroupCount()

Returns the number of groups contained by this group that the client is authorized to see.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.4.8 Method getGroupIterator()

Obtain a reference to an iterator that will return the names of each of the groups contained by this group that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.4.9 Method createRule()

Create a rule with the specified name, and get a reference to the new instance.

Returns: A reference to the just created rule.

*Parameters*

**ruleName : in TpString**

The name of the rule to create.

*Returns*

**IpPolicyRuleRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.4.10 Method getRule()

Get a reference to the specified rule.

Returns: A reference to the rule.

*Parameters*

**ruleName : in TpString**

The name of the rule to get.

*Returns*

**IpPolicyRuleRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.4.11 Method removeRule()

Remove the specified rule.

*Parameters*

**ruleName : in TpString**

The name of the rule to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.4.12 Method getRuleCount()

Returns the number of rules contained by this group that the client is authorized to see.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.4.13 Method getRuleIterator()

Obtain a reference to an iterator that will return the names of each of the rules contained by this group that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*
No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.5 Interface Class IpPolicyRepository

Inherits from: IpPolicy.

A class representing a container for reusable policy-related information. Instances of PolicyConditions and PolicyActions can be defined here and then referenced from one or more PolicyRules. Note that some instantiations of the Policy Management service will have Repositories that have been pre-defined by the Service Provider, with pre-defined PolicyConditions and PolicyActions. It may also be possible that clients with the appropriate authorizations will be able to define new Repositories and/or add new PolicyConditions and PolicyActions to existing Repositories.

| <<Interface>><br>IpPolicyRepository |
|---|
| |
| getParentRepository () : IpPolicyRepositoryRef<br>createRepository (repositoryName : in TpString) : IpPolicyRepositoryRef<br>getRepository (repositoryName : in TpString) : IpPolicyRepositoryRef<br>removeRepository (repositoryName : in TpString) : void<br>getRepositoryCount () : TpInt32<br>getRepositoryIterator () : IpPolicyIteratorRef<br>createCondition (conditionName : in TpString, conditionType : in TpPolicyConditionType, conditionAttributes : in TpAttributeSet) : IpPolicyConditionRef<br>getCondition (conditionName : in TpString) : IpPolicyConditionRef<br>removeCondition (conditionName : in TpString) : void<br>getConditionCount () : TpInt32<br>getConditionIterator () : IpPolicyIteratorRef<br>createAction (actionName : in TpString, actionType : in TpPolicyActionType, actionAttributes : in TpAttributeSet) : IpPolicyActionRef<br>getAction (actionName : in TpString) : IpPolicyActionRef<br>removeAction (actionName : in TpString) : void<br>getActionCount () : TpInt32<br>getActionIterator () : IpPolicyIteratorRef |

### 8.1.5.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

### 8.1.5.2 Method getParentRepository()

Return a reference to the repository that contains this one (if any). If this is a top-level repository, return a NULL reference.

Returns: A reference to the parent repository.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyRepositoryRef**

*Raises*

**TpCommonExceptions**

### 8.1.5.3 Method createRepository()

Create the specified repository and get a reference to the new instance.

Returns: A reference to the repository just created.

*Parameters*

**repositoryName : in TpString**

The name of the repository to create.

*Returns*

**IpPolicyRepositoryRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.5.4 Method getRepository()

Get a reference to the specified subrepository.

Returns: A reference to the repository.

*Parameters*

**repositoryName : in TpString**

The name of the subrepository to get.

*Returns*

**IpPolicyRepositoryRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.5.5 Method removeRepository()

Remove the specified subrepository.

*Parameters*

**repositoryName : in TpString**

The name of the subrepository to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.5.6 Method getRepositoryCount()

Returns the number of subrepositories contained by this repository that the client is authorized to see.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.5.7 Method getRepositoryIterator()

Obtain a reference to an iterator that will return the names of each of the subrepositories contained by this one that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*
No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.5.8 Method createCondition()

Create a reusable condition. References to the newly created condition can be used in one or more PolicyRules.

Returns: The reference to the newly created condition.

*Parameters*

**conditionName : in TpString**

The name uniquely identifying this condition within this repository.

**conditionType : in TpPolicyConditionType**

The type specifying which IpPolicyCondition class should be created. For this version of the Policy Management API, it must be one of P_PM_TIME_PERIOD_CONDITION, P_PM_EVENT_CONDITION, or P_PM_EXPRESSION_CONDITION.

**conditionAttributes : in TpAttributeSet**

The attributes specifying the condition.

*Returns*

**IpPolicyConditionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.5.9 Method getCondition()

Get a reference to the specified condition.

Returns: A reference to the specified condition.

*Parameters*

**conditionName : in TpString**

The name of the condition to get.

*Returns*

**IpPolicyConditionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.5.10 Method removeCondition()

Remove the specified condition.

*Parameters*

**conditionName : in TpString**

The name of the condition to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.5.11 Method getConditionCount()

Returns the number of conditions contained by this repository that the client is authorized to see.

Returns: The number of conditions.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.5.12 Method getConditionIterator()

Obtain a reference to an iterator that will return the names of each of the conditions contained by this repository that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.5.13 Method createAction()

Create a reusable action. References to the newly created action can be used in one or more PolicyRules.

Returns: The reference to the newly created action.

*Parameters*

**actionName : in TpString**

The name uniquely identifying this action within this repository.

**actionType : in TpPolicyActionType**

The type specifying which IpPolicyAction class should be created. For this version of the Policy Management API, it must be one of P_PM_EVENT_ACTION, or P_PM_EXPRESSION_ACTION.

**actionAttributes : in TpAttributeSet**

The attributes specifying the action.

*Returns*

**IpPolicyActionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.5.14 Method getAction()

Get a reference to the specified action.

Returns: A reference to the specified action.

*Parameters*

**actionName : in TpString**

The name of the action to get.

*Returns*

**IpPolicyActionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.5.15 Method removeAction()

Remove the specified action.

*Parameters*

**actionName : in TpString**

The name of the action to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.5.16 Method getActionCount()

Returns the number of actions contained by this repository that the client is authorized to see.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.5.17 Method getActionIterator()

Obtain a reference to an iterator that will return the names of each of the actions contained by this repository that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

## 8.1.6 Interface Class IpPolicyRule

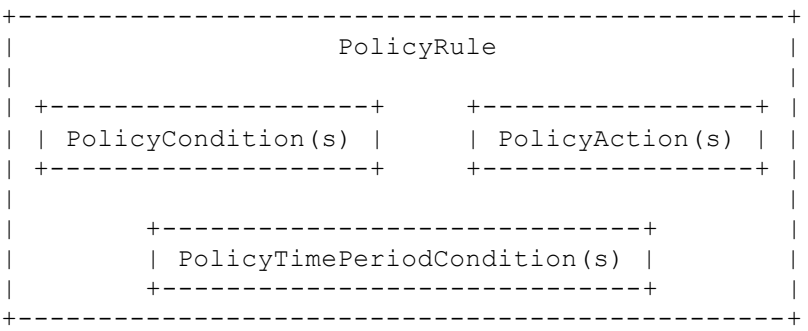
Inherits from: IpPolicy.

This class represents the "If Condition then Action" semantics associated with a policy. A PolicyRule condition, in the most general sense, is represented as either an ORed set of ANDed conditions (Disjunctive Normal Form, or DNF) or an ANDed set of ORed conditions (Conjunctive Normal Form, or CNF). Individual conditions may either be negated (NOT C) or unnegated (C). The actions specified by a PolicyRule are to be performed if and only if the PolicyRule condition (whether it is represented in DNF or CNF) evaluates to TRUE.

The conditions and actions associated with a policy rule are modelled, respectively, with subclasses of the classes PolicyCondition and PolicyAction. These condition and action objects are tied to instances of PolicyRule by the setConditionList() and setActionList() methods.

A policy rule may also be associated with one or more policy time periods, indicating the schedule according to which the policy rule is active and inactive. In this case it is the setValidityPeriodCondition() method that provides the linkage.

A policy rule is illustrated conceptually in the figure below.




The PolicyRule class uses the structure TpConditionList to specify the list of conditions for the rule and uses the attribute ConditionListType, to indicate whether the conditions for the rule are in DNF or CNF. The TpConditionList is a list of structures, each element of which contains a reference to a condition and two additional attributes to complete the representation of the rule's conditional expression. The first of these attributes is an integer to partition the referenced conditions into one or more groups, and the second is a Boolean to indicate whether the referenced condition is negated. An example shows how TpConditionList and these two additional attributes provide a unique representation of a set of conditions in either DNF or CNF.

Suppose we have a TpConditionList that aggregates five PolicyConditions C1 through C5, with the following values in the attributes of the five elements of the list:

C1: GroupNumber = 1, ConditionNegated = FALSE
C2: GroupNumber = 1, ConditionNegated = TRUE
C3: GroupNumber = 1, ConditionNegated = FALSE
C4: GroupNumber = 2, ConditionNegated = FALSE
C5: GroupNumber = 2, ConditionNegated = FALSE

If ConditionListType = P_PM_DNF, then the overall condition for the PolicyRule is:

(C1 AND (NOT C2) AND C3) OR (C4 AND C5)

On the other hand, if ConditionListType = P_PM_CNF, then the overall condition for the PolicyRule is:

(C1 OR (NOT C2) OR C3) AND (C4 OR C5)

In both cases, there is an unambiguous specification of the overall condition that is tested to determine whether to perform the actions associated with the PolicyRule.

Similarly, The PolicyRule class uses the structure TpPolicyActionList to specify the list of actions for the rule and uses the attribute SequencedActions to indicate whether the actions for the rule MUST be executed in the order

specified in the TpActionList, SHOULD be executed in the order specified, or it does not matter. The TpActionList is a list of structures, each element of which contains a reference to an action and an attribute sequenceNumber. This attribute provides an unsigned integer 'n' that indicates the relative position of an action in the sequence of actions associated with a policy rule. When 'n' is a positive integer, it indicates a place in the sequence of actions to be performed, with smaller integers indicating earlier positions in the sequence. The special value '0' indicates "do not care". If two or more actions have the same non-zero sequence number, they may be performed in any order, but they must all be performed at the appropriate place in the overall action sequence.

A series of examples will make ordering of actions clearer:

- If all actions have the same sequence number, regardless of whether it is '0' or non-zero, any order is acceptable.
- The values

1: ACTION A
2: ACTION B
1: ACTION C
3: ACTION D

indicate two acceptable orders: A, C, B, D or C, A, B, D, since A and C can be performed in either order, but only at the '1' position.

- The values

0: ACTION A
2: ACTION B
3: ACTION C
3: ACTION D

require that B,C, and D occur either as B, C, D or as B, D, C. Action A may appear at any point relative to B, C, and D. Thus the complete set of acceptable orders is: A, B, C, D; B, A, C, D; B, C, A, D; B, C, D, A; A, B, D, C; B, A, D, C; B, D, A, C; B, D, C, A.

Note that the non-zero sequence numbers need not start with '1', and they need not be consecutive. All that matters is their relative magnitude.

<<Interface>>

IpPolicyRule

getParentGroup () : IpPolicyGroupRef

getParentDomain () : IpPolicyDomainRef

createCondition (conditionName : in TpString, conditionType : in TpPolicyConditionType, conditionAttributes : in TpAttributeSet) : IpPolicyConditionRef

getCondition (conditionName : in TpString) : IpPolicyConditionRef

removeCondition (conditionName : in TpString) : void

getConditionCount () : TpInt32

getConditionIterator () : IpPolicyIteratorRef

createAction (actionName : in TpString, actionType : in TpPolicyActionType, actionAttributes : in TpAttributeSet) : IpPolicyActionRef

getAction (actionName : in TpString) : IpPolicyActionRef

removeAction (actionName : in TpString) : void

getActionCount () : TpInt32

getActionIterator () : IpPolicyIteratorRef

setValidityPeriodConditionByName (conditionName : in TpString) : void

setValidityPeriodCondition (conditionReference : in IpPolicyTimePeriodConditionRef) : void

getValidityPeriodCondition () : IpPolicyTimePeriodConditionRef

unsetValidityPeriodCondition () : void

setConditionList (conditionList : in TpPolicyConditionList) : void

getConditionList () : TpPolicyConditionList

setActionList (actionList : in TpPolicyActionList) : void

getActionList () : TpPolicyActionList

### 8.1.6.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL",

"P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

### Caption : TpString

This attribute provides a one-line description of a policy-related object.

### Description : TpString

This attribute provides a longer description than that provided by the caption attribute.

### Enabled : TpBoolean

This attribute indicates whether a policy rule is currently enabled, from an administrative point of view. Its purpose is to allow a policy administrator to enable or disable a policy rule without having to add it to, or remove it from, the policy repository.

Note that unlike RFC 3460, this attribute does not support the value 'enabledForDebug'. It was considered confusing that Enabled was not a boolean attribute. Support for debugging, including the ability to specify that the entity evaluating the policy condition(s) is being told to evaluate the conditions for the policy rule, but not to perform the actions if the conditions evaluate to TRUE, will be considered for a later release.

### RuleUsage : TpString

This attribute is a free-form string that recommends how this policy should be used.

### Priority : TpInt32

This attribute provides a non-negative integer for prioritising policy rules relative to each other. Larger integer values indicate higher priority. Since one purpose of this attribute is to allow specific, ad hoc policy rules to temporarily override established policy rules, an instance that has this attribute set has a higher priority than all instances that use or set the default value of zero.

Prioritisation among policy rules provides a basic mechanism for resolving policy conflicts.

### Mandatory : TpBoolean

This attribute indicates whether evaluation (and possibly action execution) of a PolicyRule is mandatory or not. Its concept is similar to the ability to mark packets for delivery or possible discard, based on network traffic and device load.

The evaluation of a PolicyRule MUST be attempted if the Mandatory attribute value is TRUE. If the Mandatory attribute value of a PolicyRule is FALSE, then the evaluation of the rule is "best effort" and MAY be ignored.

### PolicyRoles : TpStringSet

This attribute represents the roles and role combinations associated with a policy rule. Each value represents one role combination. Since this is a multi-valued attribute, more than one role combination can be associated with a single policy rule. Each value is a string of the form:

<RoleName>[&&<RoleName>]*

where the individual role names appear in alphabetical order.

### ConditionListType : TpPolicyConditionListType

This attribute is used to specify whether the list of policy conditions associated with this policy rule is in Disjunctive Normal Form (DNF) or Conjunctive Normal Form (CNF). If this attribute is not present, the list type defaults to DNF.

**SequencedActions : TpInt32**

This attribute gives a policy administrator a way of specifying how the ordering of the policy actions associated with this PolicyRule is to be interpreted. Three values are supported:

- mandatory(1): Do the actions in the indicated order, or do not do them at all.

- recommended(2): Do the actions in the indicated order if you can, but if you cannot do them in this order, do them in another order if you can.

- dontCare(3): Do them -- I do not care about the order.

When error / event reporting is addressed for the Policy Framework, suitable codes will be defined for reporting that a set of actions could not be performed in an order specified as mandatory (and thus were not performed at all), that a set of actions could not be performed in a recommended order (and moreover could not be performed in any order), or that a set of actions could not be performed in a recommended order (but were performed in a different order).

### 8.1.6.2 Method getParentGroup()

Return a reference to the PolicyGroup that directly contains this Rule (if any). If this Rule is contained by a PolicyDomain, return a NULL reference.

Returns: The reference to the PolicyGroup.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyGroupRef**

*Raises*

**TpCommonExceptions**

### 8.1.6.3 Method getParentDomain()

Return a reference to the PolicyDomain that directly contains this Rule (if any). If this Rule is contained by a PolicyGroup, return a NULL reference.

Returns: The reference to the PolicyDomain to get.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions**

### 8.1.6.4 Method createCondition()

Create a new condition local to this Rule. Conditions created local to a Rule can only be referenced from that Rule. For reusable conditions, see IpPolicyRepository.

Returns: The reference to the newly created condition.

*Parameters*

**conditionName : in TpString**

The name uniquely identifying this condition within this rule.

**conditionType : in TpPolicyConditionType**

The type specifying which IpPolicyCondition class should be created. For this version of the Policy Management API, it must be one of P_PM_TIME_PERIOD_CONDITION, P_PM_EVENT_CONDITION, or P_PM_EXPRESSION_CONDITION.

**conditionAttributes : in TpAttributeSet**

The initial attributes for this condition.

*Returns*

**IpPolicyConditionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.5 Method getCondition()

Get a reference to the specified condition.

Returns: A reference to the specified condition.

*Parameters*

**conditionName : in TpString**

The name of the condition to get.

*Returns*

**IpPolicyConditionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.6.6 Method removeCondition()

Remove the specified condition.

*Parameters*

**conditionName : in TpString**

The name of the condition to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.7 Method getConditionCount()

Returns the number of conditions contained by this rule that the client is authorized to see.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.6.8 Method getConditionIterator()

Obtain a reference to an iterator that will return the names of each of the conditions contained by this rule that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.6.9 Method createAction()

Create a new action local to this Rule. Actions created local to a Rule can only be referenced from that Rule. For reusable actions, see IpPolicyRepository.

Returns: The reference to the newly created action.

*Parameters*

**actionName : in TpString**

The name uniquely identifying this action within this rule.

**actionType : in TpPolicyActionType**

The type specifying which IpPolicyAction class should be created. For this version of the Policy Management API, it must be one of P_PM_EVENT_ACTION, or P_PM_EXPRESSION_ACTION.

**actionAttributes : in TpAttributeSet**

The attributes specifying the action.

*Returns*

**IpPolicyActionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.10 Method getAction()

Get a reference to the specified action.

Returns: A reference to the specified action.

*Parameters*

**actionName : in TpString**

The name of the action to get.

*Returns*

**IpPolicyActionRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.1.6.11 Method removeAction()

Remove the specified action.

*Parameters*

**actionName : in TpString**

The name of the action to delete.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.12 Method getActionCount()

Returns the number of actions contained by this rule that the client is authorized to see.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpInt32**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.6.13 Method getActionIterator()

Obtain a reference to an iterator that will return the names of each of the actions contained by this rule that the client is authorized to see.

Returns: A reference to the iterator.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyIteratorRef**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.6.14 Method setValidityPeriodConditionByName()

Set the validity period for the rule, specifying the name of a condition of type IpValidityPeriodCondition. Since the condition is specified by name, the condition must be defined local to this rule.

*Parameters*

**conditionName : in TpString**

Name identifying a condition local to this rule.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.15 Method setValidityPeriodCondition()

Set the validity period for the rule, providing a reference to a condition of type IpValidityPeriodCondition. Since the condition is specified by reference, the condition may be defined local to rule or may be a condition defined in a PolicyRepository.

*Parameters*

**conditionReference : in IpPolicyTimePeriodConditionRef**

Reference to the condition to be used to set the validity period condition.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.16 Method getValidityPeriodCondition()

Get a reference to the condition used to set the validity period condition for this rule.

Returns: The reference to the condition. This will be a NULL reference if the validity period condition is not set.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyTimePeriodConditionRef**

*Raises*

**TpCommonExceptions**

### 8.1.6.17 Method unsetValidityPeriodCondition()

Unset the validity period condition for this rule. When the validity period condition is not set, the rule is always active.

*Parameters*

No Parameters were identified for this method

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.18 Method setConditionList()

Set the condition list of this rule, specifying each triple of condition, Group Number and Negated attributes. See the text under IpPolicyRule above for a description of the use of these two attributes. Note that although a condition may be contained by a rule (by creating the condition within the rule using createCondition(), it is not evaluated as part of the rule's condition list until it is included in the list specified by this method.

*Parameters*

**conditionList : in TpPolicyConditionList**

List of (Condition reference, Group Number, Negated) triples and the value ConditionListType indicating whether the conditions are in DNF or CNF.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.19 Method getConditionList()

Get the condition list set for the rule.

Returns: The condition list currently set for this rule.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpPolicyConditionList**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION**

### 8.1.6.20 Method setActionList()

Set the list of actions for this rule, specifying each pair of Action and SequenceNumber. See the text under IpPolicyRule above for a description of the use of this attribute. Note that although an action may be contained by a rule (by creating the action within the rule using createAction()), it is not evaluated as part of the rule's actions until it is included in the list specified by this method.

*Parameters*

**actionList : in TpPolicyActionList**

List of (Action Reference, Sequence Number) pairs.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.6.21 Method getActionList()

Get the action list set for the rule.

Returns: The action list currently set for this rule.

*Parameters*
No Parameters were identified for this method

*Returns*

**TpPolicyActionList**

*Raises*

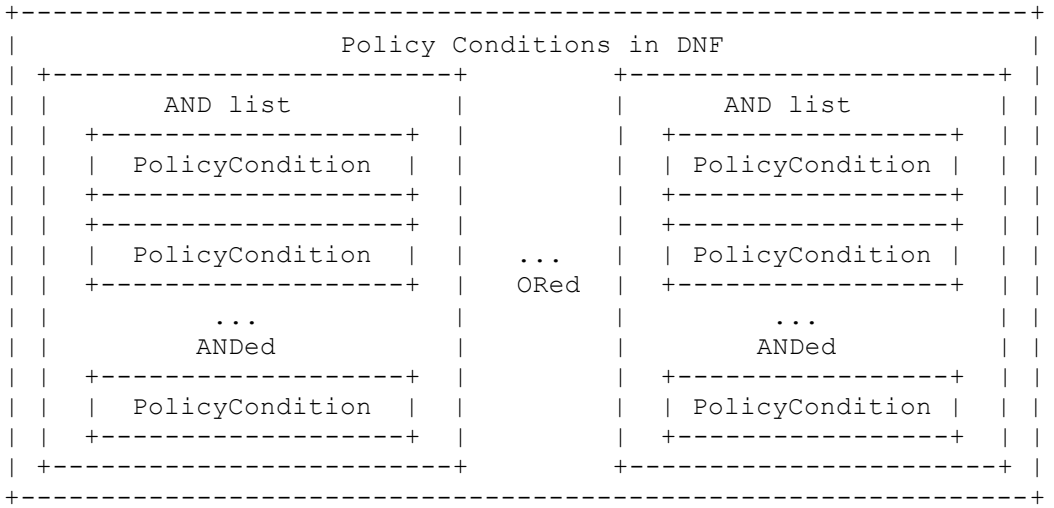
**TpCommonExceptions, P_ACCESS_VIOLATION**

## 8.1.7 Interface Class IpPolicyCondition

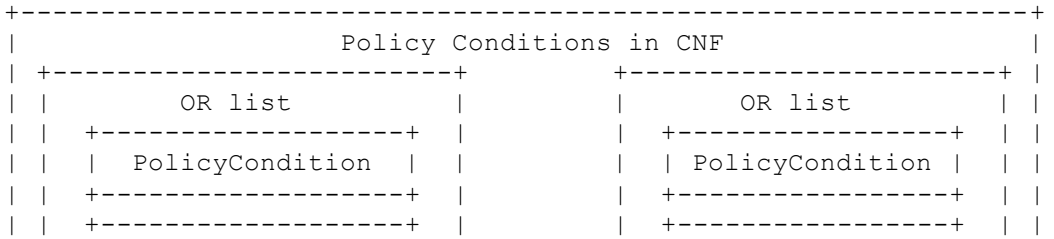
Inherits from: IpPolicy.

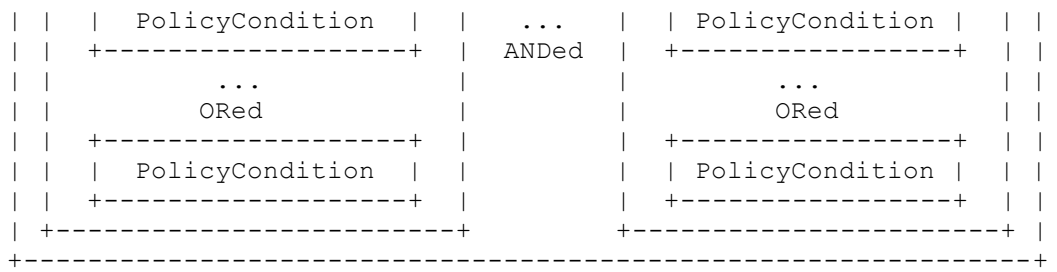
The purpose of a policy condition is to determine whether or not the set of actions (aggregated in the PolicyRule that the condition applies to) should be executed or not. For the purposes of the Policy Core Information Model, all that matters about an individual PolicyCondition is that it evaluates to TRUE or FALSE. (The individual PolicyConditions associated with a PolicyRule are combined to form a compound expression in either DNF or CNF, but this is accomplished via the ConditionList, discussed above. A logical structure within an individual PolicyCondition may also be introduced, but this would have to be done in a subclass of PolicyCondition.

Because it is general, the PolicyCondition class does not itself contain any "real" conditions. These will be represented by attributes of the domain-specific subclasses of PolicyCondition.



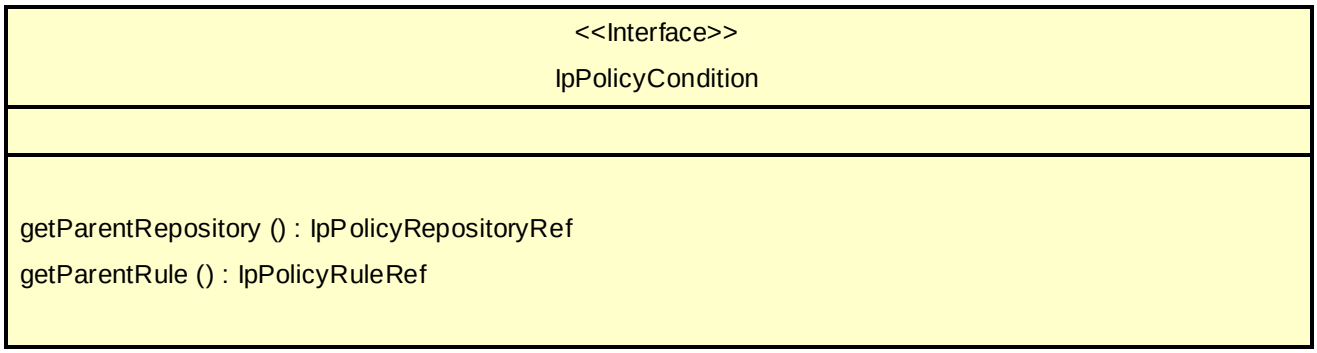

The figure above illustrates that when policy conditions are in DNF, there are one or more sets of conditions that are ANDed together to form AND lists. An AND list evaluates to TRUE if and only if all of its constituent conditions evaluate to TRUE. The overall condition then evaluates to TRUE if and only if at least one of its constituent AND lists evaluates to TRUE.






In the figure above, the policy conditions are in CNF. Consequently, there are one or more OR lists, each of which evaluates to TRUE if and only if at least one of its constituent conditions evaluates to TRUE. The overall condition then evaluates to TRUE if and only if ALL of its constituent OR lists evaluate to TRUE.

When identifying and using the PolicyCondition class, it is necessary to remember that a condition can be rule-specific or reusable. This was discussed above. The distinction between the two types of policy conditions lies in the associations in which an instance can participate, and in how the different instances are named. Conceptually, a reusable policy condition resides in a policy repository, and is named within the scope of that repository. On the other hand, a rule-specific policy condition is, as the name suggests, named within the scope of the single policy rule to which it is related.




### 8.1.7.1 Attributes

**`CommonName : TpString`**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**`PolicyKeywords : TpStringSet`**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

### 8.1.7.2 Method getParentRepository()

Return a reference to the repository that contains this condition (if any). If this condition is contained by a rule, return a NULL reference.

Returns: A reference to the parent repository.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyRepositoryRef**

*Raises*

**TpCommonExceptions**

### 8.1.7.3 Method getParentRule()

Return a reference to the rule that contains this condition (if any). If this condition is contained by a PolicyRepository, return a NULL reference.

Returns: A reference to the parent rule.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyRuleRef**

*Raises*

**TpCommonExceptions**

## 8.1.8 Interface Class IpPolicyTimePeriodCondition

Inherits from: IpPolicyCondition.

This class provides a means of representing the time periods during which a policy rule is valid, i.e., active. At all times that fall outside these time periods, the policy rule has no effect. A policy rule is treated as valid at all times if it does not specify a PolicyTimePeriodCondition.

In some cases a PDP may need to perform certain setup / cleanup actions when a policy rule becomes active / inactive. For example, sessions that were established while a policy rule was active might need to be taken down when the rule becomes inactive. In other cases, however, such sessions might be left up: in this case, the effect of

deactivating the policy rule would just be to prevent the establishment of new sessions. Setup / cleanup behaviours on validity period transitions are not currently addressed by the RFC 3460, and must be specified in 'guideline' documents, or via subclasses of PolicyRule, PolicyTimePeriodCondition or other concrete subclasses of Policy. If such behaviours need to be under the control of the policy administrator, then a mechanism to allow this control must also be specified in the subclass.

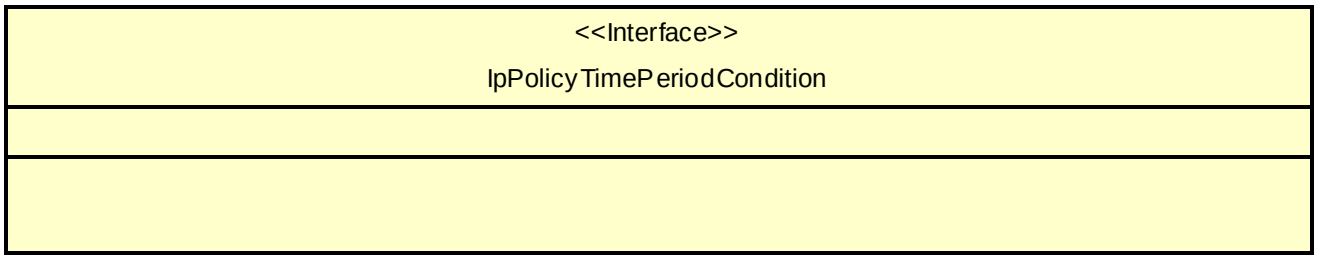
PolicyTimePeriodCondition is defined as a subclass of PolicyCondition. This is to allow the inclusion of time-based criteria in the AND/OR condition definitions for a PolicyRule.

Instances of this class may have up to five attributes identifying time periods at different levels. The values of all the attributes present in an instance are ANDed together to determine the validity period(s) for the instance. For example, an instance with an overall validity range of January 1, 2000 through December 31, 2000; a month mask that selects March and April; a day-of-the-week mask that selects Fridays; and a time of day range of 0800 through 1600 would represent the following time periods:

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

Attributes not present in an instance of PolicyTimePeriodCondition are implicitly treated as having their value "always enabled". Thus, in the example above, the day-of-the-month mask is not present, and so the validity period for the instance implicitly includes a day-of-the-month mask that selects all days of the month. If we apply this "missing attribute" rule to its fullest, we see that there is a second way to indicate that a policy rule is always enabled: have it point to an instance of PolicyTimePeriodCondition whose only attributes are its naming attributes.

The attribute LocalOrUtcTime indicates whether the times represented in the other five time-related attributes of an instance of PolicyTimePeriodCondition are to be interpreted as local times for the location where a policy rule is being applied, or as UTC times.




### 8.1.8.1 Attributes

**`CommonName : TpString`**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**`PolicyKeywords : TpStringSet`**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

### Caption : TpString

This attribute provides a one-line description of a policy-related object.

### Description : TpString

This attribute provides a longer description than that provided by the caption attribute.

### TimePeriod : TpString

This attribute identifies an overall range of calendar dates and times over which a policy rule is valid. It reuses the format for an explicit time period defined in RFC 2445: a string representing a starting date and time, in which the character 'T' indicates the beginning of the time portion, followed by the solidus character '/', followed by a similar string representing an end date and time. The first date indicates the beginning of the range, while the second date indicates the end. Thus, the second date and time must be later than the first. Date/times are expressed as substrings of the form "yyyymmddThhmmss". For example:

20000101T080000/20000131T120000

January 1, 2000, 0800 through January 31, 2000, noon

There are also two special cases in which one of the date/time strings is replaced with a special string defined in RFC 2445.

- If the first date/time is replaced with the string "THISANDPRIOR", then the attribute indicates that a policy rule is valid [from now] until the date/time that appears after the '/'.

- If the second date/time is replaced with the string "THISANDFUTURE", then the attribute indicates that a policy rule becomes valid on the date/time that appears before the '/', and remains valid from that point on.

Note that RFC 2445 does not use these two strings in connection with explicit time periods. Thus the RFC 3460 is combining two elements from RFC 2445 that are not combined in the RFC itself.

### MonthOfYearMask : TpString

The purpose of this attribute is to refine the definition of the valid time period that is defined by the TimePeriod attribute, by explicitly specifying the months when the policy is valid. These attributes work together, with the TimePeriod used to specify the overall time period during which the policy might be valid, and the MonthOfYearMask used to pick out the specific months within that time period when the policy is valid.

This attribute is formatted as an octet string of size 2, consisting of 12 bits identifying the 12 months of the year, beginning with January and ending with December, followed by 4 bits that are always set to '0'. For each month, the value '1' indicates that the policy is valid for that month, and the value '0' indicates that it is not valid. The value X'08 30', for example, indicates that a policy rule is valid only in the months May, November, and December.

See clause 5.4 of RFC 3060 for details of how CIM represents a single-valued octet string attribute such as this one. (Basically, CIM prepends a 4-octet length to the octet string.)

If this attribute is omitted, then the policy rule is treated as valid for all twelve months.

### DayOfMonthMask : TpString

The purpose of this attribute is to refine the definition of the valid time period that is defined by the TimePeriod attribute, by explicitly specifying the days of the month when the policy is valid. These attributes work together, with the TimePeriod used to specify the overall time period during which the policy might be valid, and the DayOfMonthMask used to pick out the specific days of the month within that time period when the policy is valid.

This attribute is formatted as an octet string of size 8, consisting of 31 bits identifying the days of the month counting from the beginning, followed by 31 more bits identifying the days of the month counting from the end, followed by 2 bits that are always set to '0'. For each day, the value '1' indicates that the policy is valid for that day, and the value '0' indicates that it is not valid.

The value X'80 00 00 01 00 00 00 00', for example, indicates that a policy rule is valid on the first and last days of the month.

For months with fewer than 31 days, the digits corresponding to days that the months do not have (counting in both directions) are ignored.

The encoding of the 62 significant bits in the octet string matches that used for the schedDay object in the DISMAN-SCHEDULE-MIB. See RFC 2591 for more details on this object.

See clause 5.4 of RFC 3060 for details of how CIM represents a single-valued octet string attribute such as this one. (Basically, CIM prepends a 4-octet length to the octet string.)

**DayOfWeekMask : TpString**

The purpose of this attribute is to refine the definition of the valid time period that is defined by the TimePeriod attribute by explicitly specifying the days of the week when the policy is valid. These attributes work together, with the TimePeriod used to specify the overall time period when the policy might be valid, and the DayOfWeekMask used to pick out the specific days of the week in that time period when the policy is valid.

This attribute is formatted as an octet string of size 1, consisting of 7 bits identifying the 7 days of the week, beginning with Sunday and ending with Saturday, followed by 1 bit that is always set to '0'. For each day of the week, the value '1' indicates that the policy is valid for that day, and the value '0' indicates that it is not valid.

The value X'7C', for example, indicates that a policy rule is valid Monday through Friday.

See clause 5.4 of RFC 3060 for details of how CIM represents a single-valued octet string attribute such as this one. (Basically, CIM prepends a 4-octet length to the octet string.)

**TimeOfDayMask : TpString**

The purpose of this attribute is to refine the definition of the valid time period that is defined by the TimePeriod attribute by explicitly specifying a range of times in a day the policy is valid for. These attributes work together, with the TimePeriod used to specify the overall time period that the policy is valid for, and the TimeOfDayMask used to pick out which range of time periods in a given day of that time period the policy is valid for.

This attribute is formatted in the style of RFC 2445: a time string beginning with the character 'T', followed by the solidus character '/', followed by a second time string. The first time indicates the beginning of the range, while the second time indicates the end. Times are expressed as substrings of the form "Thhmmss".

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

When a range spans midnight, it by definition includes parts of two successive days. When one of these days is also selected by either the MonthOfYearMask, DayOfMonthMask, and/or DayOfWeekMask, but the other day is not, then the policy is active only during the portion of the range that falls on the selected day. For example, if the range extends from 2100 until 0800, and the day of week mask selects Monday and Tuesday, then the policy is active during the following three intervals:

From midnight Sunday until 0800 Monday;

From 2100 Monday until 0800 Tuesday;

From 2100 Tuesday until 23:59:59 Tuesday.

**LocalOrUtcTime : TpInt32**

This attribute indicates whether the times represented in the TimePeriod attribute and in the various Mask attributes represent local times or UTC times. There is no provision for mixing of local times and UTC times: the value of this

attribute applies to all of the other time-related attributes. Note that LocalTime is designated by the integer 1 and UtcTime by the integer 2. If no value is specified the default value is 2, i.e. UtcTime is used.

## 8.1.9 Interface Class IpPolicyAction

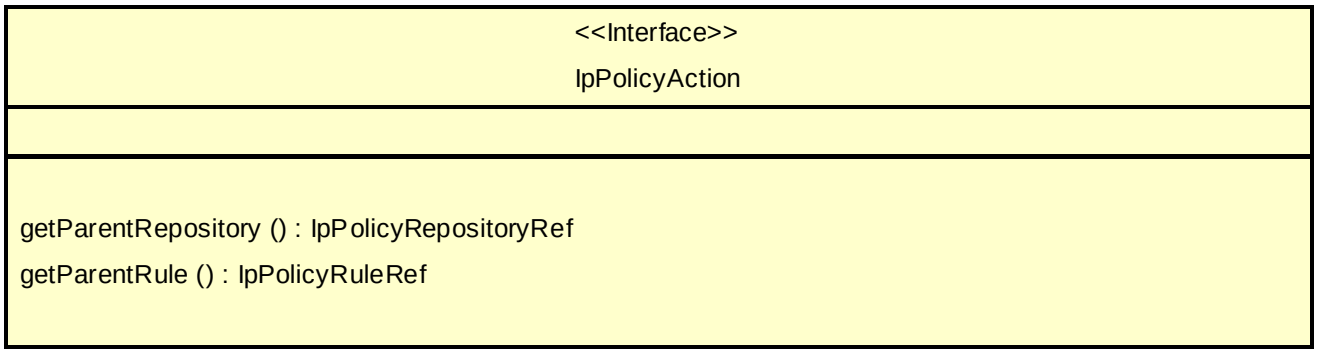
Inherits from: IpPolicy.

The purpose of a policy action is to execute one or more operations that will affect network traffic and/or systems, devices, etc., in order to achieve a desired state. This (new) state provides one or more (new) behaviours. A policy action ordinarily changes the configuration of one or more elements.

A PolicyRule contains one or more policy actions. A policy administrator can assign an order to the actions associated with a PolicyRule, complete with an indication of whether the indicated order is mandatory, recommended, or of no significance. Ordering of the actions associated with a PolicyRule is accomplished via the setActionList() method.

The actions associated with a PolicyRule are executed if and only if the overall condition(s) of the PolicyRule evaluates to TRUE.

When identifying and using the PolicyAction class, it is necessary to remember that an action can be rule-specific or reusable. This was discussed above. The distinction between the two types of policy actions lies in the associations in which an instance can participate, and in how the different instances are named. Conceptually, a reusable policy action resides in a policy repository, and is named within the scope of that repository. On the other hand, a rule-specific policy action is named within the scope of the single policy rule to which it is related.




### 8.1.9.1 Attributes

**`CommonName : TpString`**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**`PolicyKeywords : TpStringSet`**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional key words to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

### 8.1.9.2 Method getParentRepository()

Return a reference to the repository that contains this action (if any). If this action is contained by a rule, return a NULL reference.

Returns: A reference to the parent repository.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyRepositoryRef**

*Raises*

**TpCommonExceptions**

### 8.1.9.3 Method getParentRule()

Return a reference to the rule that contains this action (if any). If this action is contained by a PolicyRepository, return a NULL reference.

Returns: A reference to the parent rule.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyRuleRef**

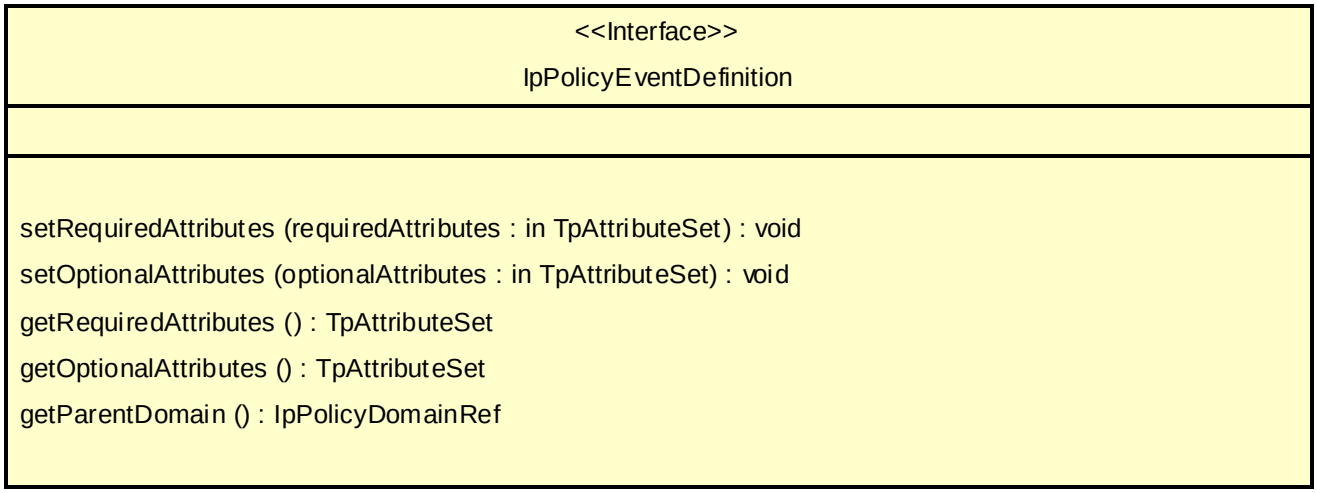
*Raises*

**TpCommonExceptions**

## 8.1.10 Interface Class IpPolicyEventDefinition

Inherits from: IpPolicy.

Instances of IpPolicyEventDefinition specify the required and optional attributes of events that can be subscribed to, specified as conditions, and generated by clients or actions.




### 8.1.10.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

**RequiredAttributes : TpAttributeSet**

The names and types of the attributes that generated events must include.

**OptionalAttributes : TpAttributeSet**

The names and types of the attributes that generated events may include.

### 8.1.10.2 Method setRequiredAttributes()

Specify the names and types of the attributes that generated events must include.

*Parameters*

**requiredAttributes : in TpAttributeSet**

The names and types of the attributes.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.10.3 Method setOptionalAttributes()

Specify the names and types of the attributes that may be included in a generated event.

*Parameters*

**optionalAttributes : in TpAttributeSet**

The names and types of the attributes.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.10.4 Method getRequiredAttributes()

Get the names and types of the attributes that a generated event is required to include.

Returns: A copy of the set of names and types.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpAttributeSet**

*Raises*

**TpCommonExceptions**

### 8.1.10.5 Method getOptionalAttributes()

Get the names and types of the attributes that a generated event may optionally include.

Returns: A copy of the set of names and types.

*Parameters*
No Parameters were identified for this method

*Returns*

**TpAttributeSet**

*Raises*

**TpCommonExceptions**

### 8.1.10.6 Method getParentDomain()

Return a reference to the domain that contains this event definition.

Returns: A reference to the containing domain.

*Parameters*
No Parameters were identified for this method

*Returns*

**IpPolicyDomainRef**

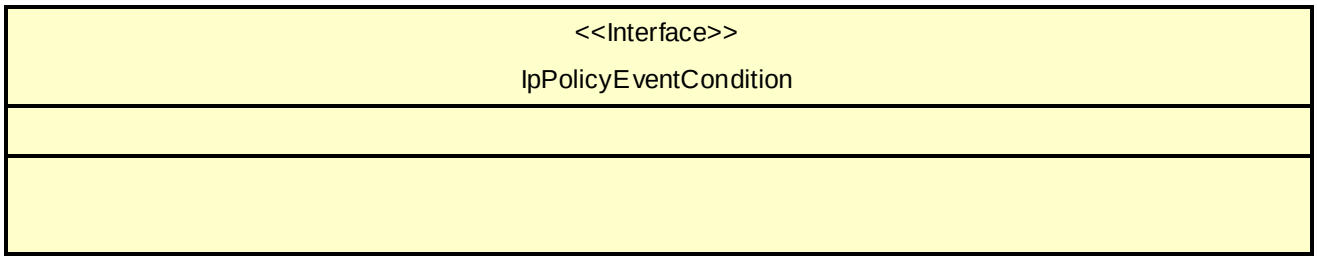
*Raises*

**TpCommonExceptions**

## 8.1.11 Interface Class IpPolicyEventCondition

Inherits from: IpPolicyCondition.

A PolicyCondition that is satisfied when the specified event, with the matching attributes, is generated.




### 8.1.11.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

**EventDefinitonName : TpString**

The EventDefinition that defines the event this condition is waiting on.

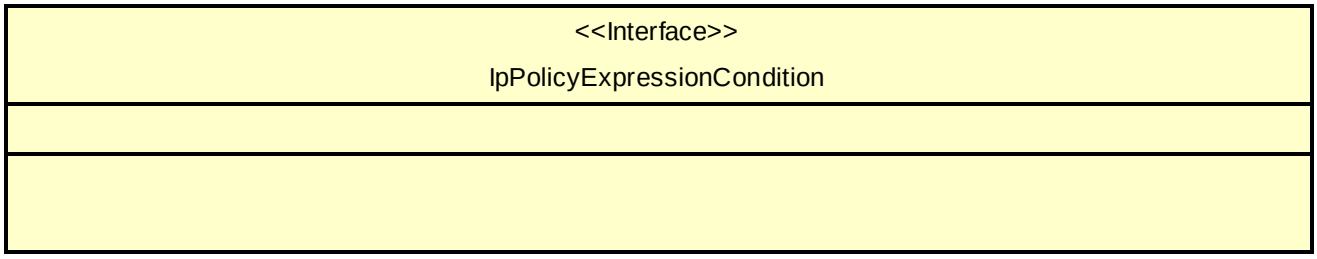
**MatchingAttributes : TpAttributeSet**

The set of attributes that must match (name and value) for the condition to be satisfied. If this set is empty, then the generation of the event is enough to satisfy the condition.

## 8.1.12 Interface Class IpPolicyExpressionCondition

Inherits from: IpPolicyCondition.

A PolicyCondition that is satisfied when the specified event, with the matching attributes, is generated.




### 8.1.12.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

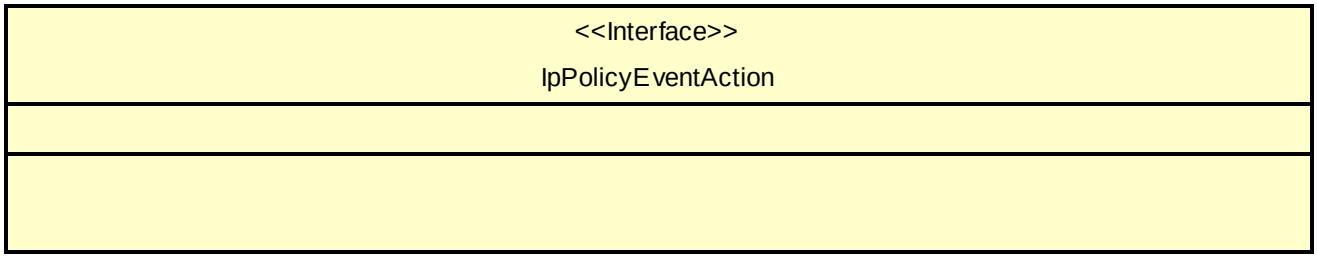
**Expression : TpString**

The expression to be evaluated as the condition. In case this SCF supports both eBNF and XML, then the TpAttributeTagInfo of the TpAttribute that populated this expression is used to distinguish between XML and eBNF string contents. A TpAttributeTagInfo value of P_XML_TYPE indicates XML as contents of the Expression attribute and a TpAttributeTagInfo value of P_SIMPLE_TYPE indicates eBNF as contents of Expression attribute. The eBNF definition can be found in clause 11.3.

## 8.1.13 Interface Class IpPolicyEventAction

Inherits from: IpPolicyAction.

Generate an instance of a specified event.




### 8.1.13.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

**EventDefinitionName : TpString**

The name of the EventDefinition that should be used to define the desired event.

**Attributes : TpAttributeSet**

The set of attributes that should be included with the generated event. Note that this set must contain all of the attributes in the RequiredAttributes attribute of the specified EventDefinition and any remaining attributes must be included in the OptionalAttributes attribute.

## 8.1.14 Interface Class IpPolicyExpressionAction

Inherits from: IpPolicyAction.

Evaluate an expression.

| <<Interface>><br>IpPolicyExpressionAction |
| --- |
| |
| |

### 8.1.14.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

**Expression : TpString**

The expression that should evaluated. In case this SCF supports both eBNF and XML, then the TpAttributeTagInfo of the TpAttribute that populated this expression is used to distinguish between XML and eBNF string contents. A TpAttributeTagInfo value of P_XML_TYPE indicates XML as contents of the Expression attribute and a TpAttributeTagInfo value of P_SIMPLE_TYPE indicates eBNF as contents of Expression attribute. The eBNF definition can be found in clause 11.3.

## 8.1.15 Interface Class IpPolicyIterator

Inherits from: IpPolicy.

This interface supports paging through the names of the appropriate objects within a container. Rather than retrieving one name at a time, this interface specifically allows the caller to specify how many names to retrieve on each call.

| <<Interface>><br>IpPolicyIterator |
|---|
| |
| getList (startIndex : in TpInt32, numberRequested : in TpInt32) : TpStringSet |

### 8.1.15.1 Attributes

**`CommonName : TpString`**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**`PolicyKeywords : TpStringSet`**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**`Caption : TpString`**

This attribute provides a one-line description of a policy-related object.

**`Description : TpString`**

This attribute provides a longer description than that provided by the caption attribute.

### 8.1.15.2 Method getList()

Return at most numberRequested names starting at location startLocation.

Returns: The list of names returned. The list can be examined to determine how many entries were actually returned.

*Parameters*

**startIndex : in TpInt32**

The index (starting at 0) of the first name to be returned.

**numberRequested : in TpInt32**

The maximum number of names expected to be returned by this call.

*Returns*

**TpStringSet**

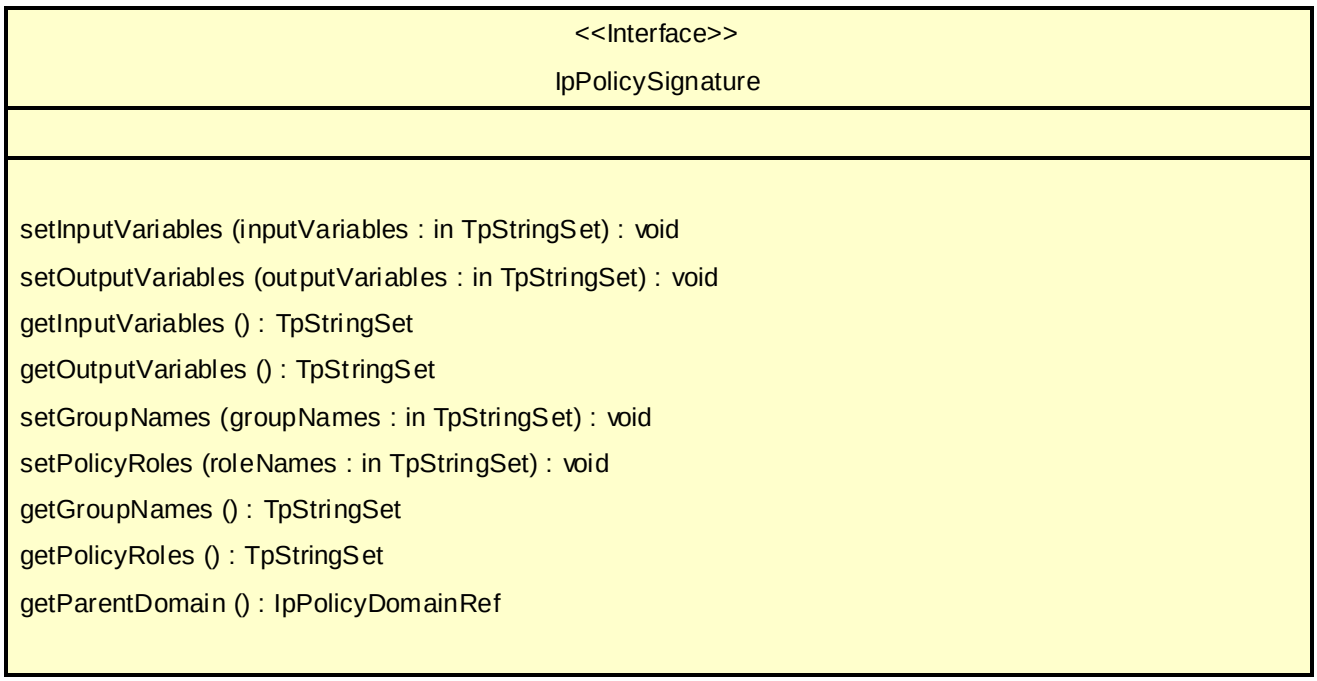
*Raises*

**TpCommonExceptions**

## 8.1.16 Interface Class IpPolicySignature

Inherits from: IpPolicy.

IpPolicySignature specifies the attributes needed to completely specify the 'context' of an evaluation request - also see definitions of createSignature(), evalPolicy(). The input and output variable names referenced below must correspond to variables whose names, types and initial values have been set via the setVariable() method which have been created via the createVariable() method.




### 8.1.16.1 Attributes

**CommonName : TpString**

The identifier used to distinguish instances of a give class of objects within a container. It is defined and referenced by the 'name' parameter used in most API methods.

**PolicyKeywords : TpStringSet**

This attribute provides a set of one or more keywords that a policy administrator may use to assist in characterizing or categorizing a policy object. Keywords are of one of two types:

- Keywords defined in the present document, or in documents that define subinterfaces of the interfaces defined in the present document. These keywords provide a vendor-independent, installation-independent way of characterizing policy objects.

- Installation-dependent keywords for characterizing policy objects. Examples include "Engineering", "Billing", and "Review in December 2000".

The present document defines the following keywords: "P_PM_KEYWORD_UNKNOWN", "P_PM_KEYWORD_CONFIGURATION", "P_PM_KEYWORD_USAGE", "P_PM_KEYWORD_SECURITY", "P_PM_KEYWORD_SERVICE", "P_PM_KEYWORD_MOTIVATIONAL", "P_PM_KEYWORD_INSTALLATION", and "P_PM_KEYWORD_EVENT". These concepts were originally defined in RFC 3460.

One additional keyword is defined: "P_PM_KEYWORD_POLICY". The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations.

Documents that define subinterfaces of the Policy Information Model interfaces SHOULD define additional keywords to characterize instances of these subinterfaces. By convention, keywords defined in conjunction with interface definitions are in uppercase. Installation-defined keywords can be in any case.

**Caption : TpString**

This attribute provides a one-line description of a policy-related object.

**Description : TpString**

This attribute provides a longer description than that provided by the caption attribute.

**inputVariables : in TpStringSet**

The names of input variables whose values are required to be available for decision request. This must not be an empty set.

**outputVariables : in TpStringSet**

The names of output variables whose values are to be sent back to a client after a decision has been rendered. This must not be an empty set.

**groupNames : in TpStringSet**

The set of names of the rule groups that must be included for policy evaluation. A group name is identical to the value of the CommonName attribute of a rule group (see clause 8.1.4.1). The set groupNames may be empty or may contain one or more group names. If the set is empty then all groups under the relevant policy domain (see clause 8.1.3) are to be considered in the evaluation. Also see roleNames below.

In general, a rule belongs to a group (of rules) with which it shares common characteristics. See clauses 8.1.4 and 8.1.6 for detailed definitions of a rule group and rule respectively.

**roleNames : in TpStringSet**

A roleName corresponds to a policy role (see clause 8.1.6.1 for the defining syntax for the attribute 'PolicyRoles' and the use of roleName therein). A roleName names the special role (or roles) of a rule within a group. Thus, e.g. roleName = content_streaming_charge && IP in a rule may be used to signify a combination of 2 roles. In this example the rule is used to compute charges for a content streaming service that is IP based . A roleName may transcend groups. Thus, e.g. 2 distinct rules in 2 distinct groups may have identical values for their policyRoles attribute. The set of roleNames may be empty or may have one or more elements. Also see the following:

a. groupNames = Null & roleNames = Null. In this case all rules under the relevant policy domain must be considered for the evaluation request.

b. groupNames != Null & roleNames = Null. In this case all rules within the named groups must be considered for the evaluation request.

c. groupNames != Null & roleNames != Null. In this case all rules from the named groups with the designated roleNames must be considered for the evaluation request.

d. groupNames = Null & policyRoles != Null. In this case all rules with the designated roleNames under the relevant policy domain must be considered.

### 8.1.16.2 Method setInputVariables()

Specify the names of the input variables that a policy-evaluation must include - also see the definition of the inputAttributes parameter for the method evalPolicy(). The types and names and initial values of these variables must be defined apriori via the setVariableType() and (if necessary) setVariableValue() methods.

*Parameters*

**inputVariables : in TpStringSet**

The names of the variables. This must not be an empty set.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.16.3 Method setOutputVariables()

Specify the names of the output variables that must be included in the output resulting from a policy-evaluation method call. These are names of variables whose values are to be returned back to the client by the evalPolicy() method. Also see the definition of the method evalPolicy(). The types and names and initial values of these variables must be set apriori via the setVariableType() and (if necessary) setVariableValue() methods.

*Parameters*

**outputVariables : in TpStringSet**

The names of the variables. This must not be an empty set.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.16.4 Method getInputVariables()

Get the names of the input variables associated with this signature.

Returns: A copy of the set of input variable names.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpStringSet**

*Raises*

**TpCommonExceptions**

### 8.1.16.5 Method getOutputVariables()

Get the names of the output variables associated with this signature.

Returns: A copy of the set of output variable names.

*Parameters*

No Parameters were identified for this method

*Returns*

**TpStringSet**

*Raises*

**TpCommonExceptions**

### 8.1.16.6 Method setGroupNames()

Specify the names of the groups that a policy-evaluation must include. A group name coincides with the value of the CommonName attribute of a relevant group (see clause 8.1.4.1).

*Parameters*

**groupNames : in TpStringSet**

The names of the groups. Elements of groupNames take values from of the CommonName attribute relevant groups. This may be NULL.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.16.7 Method setPolicyRoles()

Specify the names of the roles that a policy-evaluation must include.

*Parameters*

**roleNames : in TpStringSet**

The names of the roles. This may be NULL.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS**

### 8.1.16.8 Method getGroupNames()

Get the names of the groups associated with this signature.

Returns: A copy of the set of group names (this may be NULL).

*Parameters*

No Parameters were identified for this method

*Returns*

**TpStringSet**

*Raises*

**TpCommonExceptions**

### 8.1.16.9 Method getPolicyRoles()

Get the names of the roles associated with this signature.

Returns: A copy of the set of role names (this may be NULL).

*Parameters*

No Parameters were identified for this method

*Returns*

**TpStringSet**

*Raises*

**TpCommonExceptions**

### 8.1.16.10 Method getParentDomain()

Return a reference to the domain that contains this policy-evaluation signature.

Returns: A reference to the containing domain.

*Parameters*

No Parameters were identified for this method

*Returns*

**IpPolicyDomainRef**

*Raises*

**TpCommonExceptions**

## 8.2 PM Policy Evaluation SCF Interface Classes

The Policy Management policy evaluation APIs address the following:

- Evaluation of policy rules on request of a client application.
- Subscription to and receiving notification of policy events.
- The ability for authorized clients to generate events.

A client using the PM policy evaluation APIs should be aware that the underlying policy information, e.g. signatures, rules, policy events, variables, etc, is defined (and viewable) using the PM provisioning interface. It is therefore assumed that when a client obtains access to the IpPolicyEvalManager (for policy evaluation) it is aware of the parameters used in the methods that are supported by this interface. Note that it is possible for a client to obtain access to both the IpPolicyEvalManager and IpPolicyManager interfaces through the Parlay Framework.

### 8.2.1 Interface Class IpPolicyEvalManager

Inherits from: IpService.

An authorized client may access this interface to request evaluation of policy rules, subscribe to and receive notification of events and to generate policy events.

| <<Interface>><br>IpPolicyEvalManager |
|---|
| |
| evalPolicy (domainName : in TpString, signatureName : in TpString, inputVariables : in TpPolicyNameValueList) : TpPolicyNameValueList<br>evalPolicyReq (domainName : in TpString, signatureName : in TpString, inputVariables : in TpPolicyNameValueList, appPolicyDomain : in IpAppPolicyDomainRef) : TpAssignmentID<br>abortEvalPolicyReq (domainName : in TpString, assignmentID : in TpAssignmentID) : void<br>generateEvent (domainName : in TpString, eventDefinitionName : in TpString, attributes : in TpAttributeSet) : void<br>createNotification (domainName : in TpString, appPolicyDomain : in IpAppPolicyDomainRef, events : in TpStringSet) : TpAssignmentID<br>destroyNotification (assignmentID : in TpAssignmentID, events : in TpStringSet) : void |

#### 8.2.1.1 Method evalPolicy()

Invoke an evaluation of policy rules. Note that an evalPolicy() request is associated with a signature name that is specified through the attribute signatureName. This is to ensure that a 'context' is established for the evaluation request. This also allows for cross validation of the names of the input variables that are specified below via the attribute inputAttributes.

Returns: The output values included in the associated output structure, TpNameValueList.

*Parameters*

**domainName : in TpString**

The name of relevant domain. The name of a relevant domain may also be obtained from the policy management SCF by invoking the appropriate methods for that SCF.

**signatureName : in TpString**

The name of the signature that is to be used for this request. Must be a valid signature name in the relevant domain, i.e. the value of signatureName must correspond to the CommonName attribute of an IpPolicySignature created under the relevant IpPolicyDomain.

**inputVariables : in TpPolicyNameValueList**

The input variable name-value pairs that will be included in this request. Note that the collection of variable name specified in inputVariables must correspond to (a subset of) variables names set in the inputVariables attribute of the signature 'signatureName' in the policy management SCF.

*Returns*

**TpPolicyNameValueList**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.2.1.2 Method evalPolicyReq()

The synchronous version of evalPolicy().

Returns: TpAssignmentID contains the ID that is assigned to the asynchronous request.

Note that if any exception occurs the client will be notified synchronously and no assignment ID will be issued.

*Parameters*

**domainName : in TpString**

The name of relevant domain. The name of a relevant domain may also be obtained from the policy management SCF by invoking the appropriate methods for that SCF.

**signatureName : in TpString**

The name of the signature that is to be used for this request. Must be a valid signature name in the relevant domain, i.e. the value of signatureName must correspond to the CommonName attribute of an IpPolicySignature created under the relevant IpPolicyDomain.

**inputVariables : in TpPolicyNameValueList**

The input variable name-value pairs that will be included in this request. Note that the collection of variable name specified in inputVariables must correspond to (a subset of) variables names set in the inputVariables attribute of the signature 'signatureName' in the policy management SCF.

**appPolicyDomain : in IpAppPolicyDomainRef**

Call back reference to the client's address.

*Returns*

**TpAssignmentID**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

### 8.2.1.3 Method abortEvalPolicyReq()

This method is invoked to abort a specific asynchronous request made via an evalPolicyReq() invocation and identified by its assignment ID.

*Parameters*

**domainName : in TpString**

The name of relevant domain. The name of a relevant domain may also be obtained from the policy management SCF by invoking the appropriate methods for that SCF.

**assignmentID : in TpAssignmentID**

assignmentID: in TpAssignmentID.

*Raises*

**TpCommonExceptions, P_INVALID_ASSIGNMENT_ID**

### 8.2.1.4 Method generateEvent()

Generate an event using the attributes specified. Validate the attributes against the instance of IpPolicyEventDefinition specified by the eventDefinitionName parameter. Validation includes verifying that all of the attributes specified as required by the IpPolicyEventDefinition are included in the supplied attributes and that the supplied attributes do not include any attributes that are not specified as either required or optional by the IpPolicyEventDefinition.

*Parameters*

**domainName : in TpString**

The name of relevant domain. The name of a relevant domain may also be obtained from the policy management SCF by invoking the appropriate methods for that SCF.

**eventDefinitionName : in TpString**

The name of the definition of the event that will be used to validate attributes.

**attributes : in TpAttributeSet**

The attributes that will be included in the event instance that is generated.

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

#### 8.2.1.5 Method createNotification()

Allows a client to specify a set of events that they are interested in receiving. Once successfully subscribed, the client will receive copies of all generated events on the callback provided by the appPolicyDomain parameter.

Returns: An identifier for this subscription. When the client is no longer interested in receiving these events, it should call destroyNotification() with this identifier.

*Parameters*

**domainName : in TpString**

The name of relevant domain. The name of a relevant domain may also be obtained from the policy management SCF by invoking the appropriate methods for that SCF.

**appPolicyDomain : in IpAppPolicyDomainRef**

The callback to be used to send generated events to the client.

**events : in TpStringSet**

The set of names of event definitions specifying the events the client wishes to subscribe to.

*Returns*

**TpAssignmentID**

*Raises*

**TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR**

#### 8.2.1.6 Method destroyNotification()

Allows a client to indicate that it is no longer interested in receiving events that it previously subscribed to.

*Parameters*

**assignmentID : in TpAssignmentID**

The identifier the client received when it subscribed for the events.

**events : in TpStringSet**

If non-NULL and non-empty, this indicates the particular events that the client no longer wishes to receive. If NULL or empty, then the client is unsubscribing from all events associated with the specified identifier.

*Raises*

**TpCommonExceptions, P_SYNTAX_ERROR**

### 8.2.2 Interface Class IpAppPolicyDomain

Inherits from: IpInterface.

This interface is supported by the client. Return values or error messages resulting from asynchronous method calls are sent to this interface.

| <<Interface>><br>IpAppPolicyDomain |
|---|
| |
| reportNotification (assignmentID : in TpAssignmentID, event : in TpPolicyEvent) : void<br>evalPolicyRes (assignmentID : in TpAssignmentID, outputVariables : in TpPolicyNameValueList) : void<br>evalPolicyErr (assignmentID : in TpAssignmentID, error : in TpPolicyError) : void |

### 8.2.2.1 Method reportNotification()

Notify the client about the specified event.

*Parameters*

**assignmentID : in TpAssignmentID**

The assignmentID returned by the call to createNotification that enabled notification for the specified event.

**event : in TpPolicyEvent**

The event whose occurrence is being reported.

### 8.2.2.2 Method evalPolicyRes()

This method is invoked to pass back the results of an evalPolicyReq() invocation. The results are directed to the assignment ID that is obtained by the client upon invoking evalPolicyReq().

*Parameters*

**assignmentID : in TpAssignmentID**

AssignmentID is the unique ID that was assigned when a client invoked evalPolicyReq().

**outputVariables : in TpPolicyNameValueList**

TpNameValueList contains name-value pairs that are returned to the client.

### 8.2.2.3 Method evalPolicyErr()

This method is invoked to pass back any error resulting from the invocation of evalPolicyReq().

*Parameters*

**assignmentID : in TpAssignmentID**

AssignmentID is the unique ID that was assigned when a client invoked evalPolicyReq().

**error : in TpPolicyError**

Specifies the error, which led to the original request to fail.

# 9 State Transition Diagrams

## 9.1 PM Provisioning SCF State Transition Diagrams

There are no State Transition Diagrams for the PM Provisioning SCF.

## 9.2 PM Policy Evaluation SCF State Transition Diagrams

There are no State Transition Diagrams for the PM Policy Evaluation SCF.

# 10 PM Service Properties

The following table lists properties relevant to all the PM SCFs

| Property | Type | Description |
|---|---|---|
| P_SUPPORTED_ATTRIBUTE_TAGS | STRING_SET | Lists the supported attribute tags defined by TpAttributeTagInfo |
| P_SUPPORTED_VARIABLE_TAGS | STRING_SET | Lists the supported variable tags defined by TpPolicyTypeInfo |
| P_SUPPORTED_SIMPLE_ATTRIBUTE_TYPES | STRING_SET | Lists the supported attribute types defined by TpSimpleAttributeTypeInfo |
| P_SUPPORTED_SIMPLE_VARIABLE_TYPES | STRING_SET | Lists the supported variable types defined by TpSimpleAttributeTypeInfo |
| P_SUPPORTED_STRUCTURED_ATTRIBUTE_TYPES | STRING_SET | Lists the supported attribute types defined by TpStructuredAttributeType, e.g. P_org/csapi/TpAddress. |
| P_SUPPORTED_STRUCTURED_VARIABLE_TYPES | STRING_SET | Lists the supported variable types defined by TpStructuredAttributeType, e.g. P_org/csapi/TpAddress. |
| P_SUPPORTED_XML | STRING_SET | Lists the supported versions of XML specifications such as XML schema specifications (e.g. through URLs), XML versions (e.g. version 1.0) or XPath (e.g. version 1.0) |

Implementations of the PM APIs shall have the Service Properties set to the indicated values at a minimum:

```
P_SUPPORTED_ATTRIBUTE_TAGS = {
P_SIMPLE_TYPE
}

P_SUPPORTED_SIMPLE_ATTRIBUTE_TYPES = {
P_STRING,
P_FLOAT,
P_INT32,
P_BOOLEAN
}

P_SUPPORTED_VARIABLE_TAGS = {
P_SIMPLE_TYPE
P_PM_TYPE_RECORD,
P_PM_TYPE_LIST
}

P_SUPPORTED_SIMPLE_VARIABLE_TYPES = {
P_STRING,
P_FLOAT,
P_INT32,
P_BOOLEAN
}
```

# 11 Data Definitions

All data types referenced in this document but not defined in this clause are common data definitions which may be found in 3GPP TS 29.198-2.

## 11.1 Policy Management Data Definitions

This clause provides the Policy Management specific data definitions necessary to support the OSA interface specification.

The general format of a data definition specification is the following:

- Data type, that shows the name of the data type.
- Description, that describes the data type.
- Tabular specification, that specifies the data types and values of the data type.
- Example, if relevant, shown to illustrate the data type.

### 11.1.1 TpPolicyConditionListType

This data type defines the type condition list in a policy rule.

| Name | Value | Description |
|---|---|---|
| P_PM_DNF | 0 | Disjunctive normal form |
| P_PM_CNF | 1 | Conjunctive normal form |

### 11.1.2 TpPolicyConditionListElement

This data type is a `Sequence of Data Elements` which describes one element of a condition list. It is a structured data type consisting of the following {condition, groupNumber, negated} tuple:

| Sequence Element Name | Sequence Element Type |
|---|---|
| Condition | IpPolicyCondition |
| GroupNumber | TpInt32 |
| Negated | TpBoolean |

### 11.1.3 TpPolicyConditionList

This data type is a `Numbered Set of Data Elements` of type TpPolicyConditionListElement.

### 11.1.4 TpPolicyConditionType

This data type defines the condition type in a policy rule.

| Name | Value | Description |
|---|---|---|
| P_PM_TIME_PERIOD_CONDITION | 0 | IpPolicyTimePeriodCondition |
| P_PM_EVENT_CONDITION | 1 | IpPolicyEventCondition |
| P_PM_EXPRESSION_CONDITION | 2 | IpPolicyExpressionCondition |

### 11.1.5 TpPolicyActionListElement

This data type is a `Sequence of Data Elements` which describes one element of a action list. It is a structured data type consisting of the following {action, sequenceNumber ) pair:

| Sequence Element Name | Sequence Element Type |
|---|---|
| Action | IpPolicyAction |
| SequenceNumber | TpInt32 |

### 11.1.6 TpPolicyActionList

This data type is a `Numbered Set of Data Elements` of type TpPolicyActionListElement.

### 11.1.7 TpPolicyActionType

This data type defines the action type in a policy rule.

| Name | Value | Description |
|---|---|---|
| P_PM_EVENT_ACTION | 0 | IpPolicyEventAction |
| P_PM_EXPRESSION_ACTION | 1 | IpPolicyExpressionAction |

### 11.1.8 TpPolicyEvent

This data type is a `Sequence of Data Elements` which describes a generic "event". Events can be generated in response to network activity, as a result of clients calling the generateEvent() method of IpPolicyDomain, or as a result of the evaluation of an IpPolicyEventAction action. Each instance of a generated event is identified by a unique EventID, a 32-bit integer. The time the event was generated is captured in the attribute TimeGenerated. All of the attributes in the RequiredAttributes list of the EventDefinition associated with the given EventDefinitionName must be present in Attributes. Any other attributes must be in the OptionalAttributes list of the same EventDefinition.

It is a structured data type consisting of the following fields:

| Sequence Element Name | Sequence Element Type |
|---|---|
| EventID | TpInt32 |
| TimeGenerated | TpDateAndTime |
| Attributes | TpAttributeSet |
| EventDefinitionName | TpString |
| EventDomainName | TpString |

### 11.1.9 TpPolicyKeyword

This data type is identical to a TpString, and is defined as a string of characters that identify the Policy Keywords that are to be supported by the Policy Management API. Other Network operator specific keywords may also be used, but should be preceded by the string "`SP_`". The following values are defined.

| Name | Description |
|---|---|
| P_PM_KEYWORD_UNKNOWN | To be used when none of the defined values apply. |
| P_PM_KEYWORD_CONFIGURATION | Configuration Policies define the default (or generic) setup of a managed entity (for example, a network service). Examples of Configuration Policies are the setup of a network forwarding service or a network-hosted print queue. |

| Name | Description |
|---|---|
| P_PM_KEYWORD_USAGE | Usage Policies control the selection and configuration of entities based on specific "usage" data. Configuration Policies can be modified or simply re-applied by Usage Policies. Examples of Usage Policies include upgrading network forwarding services after a user is verified to be a member of a "gold" service group, or reconfiguring a printer to be able to handle the next job in its queue. |
| P_PM_KEYWORD_SECURITY | Security Policies deal with verifying that the client is actually who the client purports to be, permitting or denying access to resources, selecting and applying appropriate authentication mechanisms, and performing accounting and auditing of resources. |
| P_PM_KEYWORD_SERVICE | Service Policies characterize network and other services (not use them). For example, all wide-area backbone interfaces shall use a specific type of queuing.<br>Service policies describe services available in the network. Usage policies describe the particular binding of a client of the network to services available in the network. |
| P_PM_KEYWORD_MOTIVATIONAL | Motivational Policies are solely targeted at whether or how a policy's goal is accomplished. Configuration and Usage Policies are specific kinds of Motivational Policies. Another example is the scheduling of file backup based on disk write activity from 8am to 3pm, M-F. |
| P_PM_KEYWORD_INSTALLATION | Installation Policies define what can and cannot be put on a system or component, as well as the configuration of the mechanisms that perform the install. Installation policies typically represent specific administrative permissions, and can also represent dependencies between different components (e.g. to complete the installation of component A, components B and C must be previously successfully installed or uninstalled). |
| P_PM_KEYWORD_EVENT | Error and Event Policies. For example, if a device fails between 8am and 9pm, call the system administrator, otherwise call the Help Desk. |
| P_PM_KEYWORD_POLICY | The role of this keyword is to identify policy-related instances that would not otherwise be identifiable as being related to policy. It may be needed in some repository implementations. |

## 11.1.10 TpPolicyKeywordSet

This data type defines a `Numbered Set of Data Elements` of type TpPolicyKeyword .

### 11.1.11 TpPolicyError

| Name | Value | Description |
|---|---|---|
| P_PM_ERROR_UNDEFINED | 0 | Undefined Error |
| P_PM_ERROR_INSUFFICIENT_INPUTS | 1 | Required input variable values not available |
| P_PM_ERROR_INVALID_INPUT_NAME | 2 | Invalid input variable name |
| P_PM_ERROR_INVALID_INPUT_VALUE | 3 | Invalid input variable value |
| P_PM_ERROR_DB_ERROR | 4 | Error reading required rules from DB |
| P_PM_ERROR_EVALUATION_ERROR | 5 | Run-time error in evaluation of rule conditions/actions |
| NOTE: TpPolicyError is of type TpInt32. The table is an enumeration of all error codes. | | |

### 11.1.12 IpPolicyDomain

Defines the address of an `IpPolicyDomain` Interface.

### 11.1.13 IpPolicyDomainRef

Defines a `Reference` to an IpPolicyDomain.

### 11.1.14 IpPolicyRepository

Defines the address of an `IpPolicyRepository` Interface.

### 11.1.15 IpPolicyRepositoryRef

Defines a `Reference` to an IpPolicyRepository.

### 11.1.16 IpPolicyGroup

Defines the address of an `IpPolicyGroup` Interface.

### 11.1.17 IpPolicyGroupRef

Defines a `Reference` to an IpPolicyGroup.

### 11.1.18 IpPolicyRule

Defines the address of an `IpPolicyRule` Interface.

### 11.1.19 IpPolicyRuleRef

Defines a `Reference` to an IpPolicyRule.

### 11.1.20 IpPolicyEventDefinition

Defines the address of an `IpPolicyEventDefinition` Interface.

### 11.1.21 IpPolicyEventDefinitionRef

Defines a `Reference` to an IpPolicyEventDefinition.

### 11.1.22 IpAppPolicyDomain

Defines the address of an `IpAppPolicyDomain` Interface.

### 11.1.23 IpAppPolicyDomainRef

Defines a `Reference` to an IpAppPolicyDomain .

### 11.1.24 IpPolicyCondition

Defines the address of an `IpPolicyCondition` Interface.

### 11.1.25 IpPolicyConditionRef

Defines a `Reference` to an IpPolicyCondition .

### 11.1.26 IpPolicyTimePeriodCondition

Defines the address of an `IpPolicyTimePeriodCondition` Interface.

### 11.1.27 IpPolicyTimePeriodConditionRef

Defines a `Reference` to an IpPolicyTimePeriodCondition .

## 11.2 Data Types for PM Variables

### 11.2.1 TpPolicyVar

This defines the TpPolicyVar type. It is analogous to the TpAttribute type.

TpPolicyVar is a `Sequence of Data Elements` of variable name, type and value.

| Sequence Element Name | Sequence Element Type | Notes |
|---|---|---|
| VarName | TpString | Name of variable. |
| VarType | TpPolicyType | Type of variable. Could be atomic or complex type. |
| VarValue | TpAny | ValPue of variable. Note that depending on context, the AttributeValue may be NULL. |

The following types of variables are defined: Atomic types, Record types (having named fields), and List types (where list elements could be complex types as well).

### 11.2.2 TpPolicyVarSet

TpPolicyVarSet is a `Numbered Set of Data Elements` of type TpPolicyVar.

### 11.2.3 TpPolicyRecordType

Records have named fields, each field being a TpPolicyType itself. This allows nested structures to be defined. This contains the following data members:

| Sequence Element Name | Sequence Element Type | Notes |
|---|---|---|
| Names | Sequence of TpString | Name of record fields. |
| Types | Sequence of TpPolicyType | Type of record fields. |

## 11.2.4 TpPolicyListType

This defines a homogeneous list type. This contains the following data member:

| Sequence Element Name | Sequence Element Type | Notes |
|---|---|---|
| ElementType | TpPolicyType | Type of the elements of the list. |

## 11.2.5 TpPolicyTypeInfo

TpPolicyTypeInfo is an enumerated type used as a discriminator for the TpPolicyType structure, and can contain the following values:

| Name | Value | Description |
|---|---|---|
| P_PM_SIMPLE_TYPE | 0 | Simple type |
| P_PM_TYPE_RECORD | 1 | Record type |
| P_PM_TYPE_LIST | 2 | List type |
| P_PM_STRUCTURED_TYPE | 3 | Structured type |
| P_PM_XML_TYPE | 4 | XML type |

## 11.2.6 TpPolicyType

This is a Tagged Choice of Data Elements with a TpPolicyTypeInfo discriminator, and can be one of the following:

| | Tag Element Type | |
|---|---|---|
| | TpPolicyTypeInfo | |

| Tag Element Value | Choice Element Type | Choice Element Name |
|---|---|---|
| P_PM_SIMPLE_TYPE | TpSimpleAttributeTypeInfo | SimpleType |
| P_PM_TYPE_RECORD | TpPolicyRecordType | RecordType |
| P_PM_TYPE_LIST | TpPolicyListType | ListType |
| P_PM_STRUCTURED_TYPE | TpStructuredAttributeType | StructuredType |
| P_PM_XML_TYPE | TpXMLString | XMLString |

TpPolicyType allows us to define arbitrarily nested complex types as shown below. The level of nested data types actually supported is implementation specific.

The choice elements represent the following:

SimpleType: Defines an atomic type.

RecordType: Defines a record type with named fields.

ListType: Defines a homogeneous list type. Heterogeneous lists are not supported.

StructuredType Defines an object of the specified, fully qualified class.

XMLString Defines a data type that contains well-formed XML.

## 11.2.7 TpPolicyNameValue

This data structure is used to pass in a variable name-value pair in an evalPolicy() request. It is a Sequence of Data Elements of a variable name and value.

| Sequence Element Name | Sequence Element Type | Notes |
|---|---|---|
| Name | TpString | Name of Variable. |
| Value | TpAny | Value of variable. |

### 11.2.8 TpPolicyNameValueList

This type defines a `Numbered Set of Data Elements` of type TpPolicyNameValue.

## 11.3 eBNF for Condition and Action expressions

The eBNF for the action/condition expressions follows – note that these express constraints on the Expression attribute of IpPolicyCondition and IpPolicyExpression. The eBNF specifies rules for conditions/action expressions only (i.e. condition groups, negation of conditions are assumed to be handled at a higher level). Moreover, rules are given only for a single action expression, whereas a rule can contain multiple action expressions.

### 11.3.1 Basic Definition

We define some basic tokens that are used in the rest of the eBNF. The "…" used below indicate a range of corresponding characters. For example, the "…" in `letter` corresponds to all letters between `b` and `z`, both lower and uppercase). Similarly, the "…" in `char` corresponds to *printable* characters.

```
digit               ::= "0" | "1" | ... | "9";
letter              ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z";
alphanumeric        ::= digit | letter;
char                ::= alphanumeric | "\"" | "\'" | "." | "+" | ...;
identifier          ::= letter {[alphanumeric | "_"]}*;
```

Note 1: For a complete definition of the char type, see clauses 3.10.1.3 and 3.10.1.4 of the CORBA 2.4.2 Architecture and Specification document dated Feb 2001.

Note 2: The variable name syntax must conform to the eBNF specified by the identifier non-terminal above.

### 11.3.2 Definitions of Constant (Literals)

The following define the basic literals allowed. Examples include boolean literals (`true` and `false`), character literals (e.g. 'x', 'a'), string literals (e.g. "Parlay", "CORBA"), integer constants (e.g., -4, +23, 45, 05), float constants (e.g., -2.3, 4.0, 5.6e-23). We also define a `number` to be either an integer or a float, and a `const` to be any of the these constant types.

```
bool_const          ::= "true" | "false";
string_const        ::= '"' {char}* '"';
int_const           ::= {digit}+;
float_const         ::= (({digit}* "." {digit}+)
                      | ({digit}+ "." {digit}*))([eE][-+]?{digit}+)?
```

NOTE: For a complete definition of the char type, see clauses 3.10.1.3 and 3.10.1.4 of the CORBA 2.4.2 Architecture and Specification document dated Feb 2001.

```
number::=
      int_const
    | float_const
    ;

const::=
      bool_const
    | string_const
    | number
    ;
```

### 11.3.3 Definition of Operators

These define the unary arithmetic operators, binary arithmetic operators, as well as the boolean operators. "%" is the modulo operator, "in" is a list containment operator (e.g. can be used to check if a element is within a list, or if a list is contained within another). Note that the standard operator precedence will be enforced on top of this grammar.

```
unary_arith_op      ::= "+" | "-";
binary_arith_op     ::= "+" | "-" | "*" | "/" | "%";
boolean_op          ::= "<=" | "<" | "==" | ">" | ">=" | "!=" | "in";
```

### 11.3.4 Allowable arithmetic expressions & predicates

These following define complex arithmetic expressions and predicates.

```
arith_expr::=
      number
    | unary_arith_op arith_expr
    | arith_expr binary_arith_op arith_expr
    | "(" arith_expr ")"
    | var_access
    ;

predicate::=
      bool_const
    | arith_expr boolean_op arith_expr
    | (arith_expr | const) ("==" | "!=") (arith_expr | const)
    | "(" predicate ")"
    | var_access
    ;
```

Examples of arithmetic expressions include:

```
2 + 2
((4 + interval) % 100 - 42)
a + b.c / d + list[i+j].f
```

Examples of predicates include:

```
true
(interval > 100)
((4 + interval) % 100 - 42) > list[j].c * 2
(caller in buddy_list)
```

### 11.3.5 Accessing Variables

These following definitions specify how variables (simple or complex typed – see clause 10.1) can be accessed in rules. List (array) elements are accessed via a standard index ("[]") operator, and record fields are accessed via the dot (".") operator. Examples include `x`, `rec.b`, `list[42].a`, etc.

```
var_access::=
          identifier
            | var_access "." identifier
            | var_access "[" arith_expr "]"
            ;
```

### 11.3.6 Allowable Condition and Action Expressions

The following complete the definition of condition and action expressions. The condition expression corresponds exactly to the `predicate` mentioned above, while an action expression can be one of a simple assignment operation (=), or list append/delete operations (+= and −=). These specify the syntax of the Expression attribute in IpPolicyExpressionCondition and IpPolicyExpressionAction objects. Additional methods such as setConditionList() and setActionList() in IpPolicyRule interface need to be invoked in order to create a complete rule definition.

```
expr::=
          const
        | arith_expr
        | predicate
        | "!" predicate
        | predicate "&&" predicate
        | predicate "||" predicate'
        | "(" expr ")"
        ;

condition::= predicate;

action::= simple_var_access "=" expr
            | identifier "+=" expr
            | identifier "-=" expr
            ;
```

Examples of action expressions include:

```
            i = j+k
            can_insert = (! is_empty)

            // the following appends element 5 to the end of a list of integers
            L1 += 5

            // the following deletes all occurrences of element rec from the list
            L2 -= rec
```

## 11.4 Example Scenarios

We now present a high-level scenario that illustrates how all the different extensions are tied together. The rulegroup that we will use contains only one rule, which uses two variables x, and y, which are of the type:

```
x: struct {
   a: TpInt32;
   b: TpFloat;
}
y: TpInt32;
```

Moreover, let us assume that there is only one rulegroup ("testgroup") associated with the domain we are considering, and the rulegroup contains only one rule of the form (it is easy to extend this scenario to the general case):

```
if (x.b < 3)
then
   y = x.a;
end
```

Finally, assume that the value of x is to be supplied for rule evaluation, and the value of y is to be returned back to the client. The steps that need to be performed are as follows given below (we will give pseudo-code for all the steps):). Note that the actual implementations (e.g. CORBA, Java etc.) corresponding to these may differ slightly from that presented below.

1) Provision variables:

```
// get the manager
IpPolicyManagerRef manager = …;

// start transaction
manager.startTransaction();

// get the domain
IpPolicyDomainRef domain = manager.getDomain("testdomain");

// create a variable set
domain.createVariableSet("vset");

// define the type of x
// note that we can use the int_type defined as part of this
// process, for the type of y as well
TpPolicyType int_type = TpPolicyType(TpSimpleAttributeTypeInfo(P_INT32));
```

```
TpPolicyType float_type = TpPolicyType(TpSimpleAttributeTypeInfo(P_FLOAT));
Vector<TpString> field_names = ["a", "b"];
Vector<TpPolicyType> field_types = [int_type, float_type];
TpPolicyType x_type = TpPolicyType(TpRecordType(field_names,field_types));

// define the type of y
TpPolicyType y_type = TpPolicyType(TpSimpleAttributeTypeInfo(P_INT32));


// create the variables in the variable set
domain.createVariable("vset", "x", x_type);
domain.createVariable("vset", "y", y_type);

// set the values of x and y
TpAny x_value = {1, 2.5};
TpAny y_value = 3;
domain.setVariableValue("vset", "x", x_value);
domain.setVariableValue("vset", "y", y_value);
```

2) Create signature:

```
IpPolicySignatureRef sig = domain.createSignature("test_sig");

// set input and output variables
TpStringSet input_vars = ["x"];
TpStringSet output_vars = ["y"];
sig.setInputVariables(input_vars);
sig.setOutputVariables(output_vars);

// set groups and roles
TpStringSet groups = ["testgroup"];
TpStringSet roles = []; // no roles specified
sig.setGroupNames(groups);
sig.setRoleNames(roles);
```

3) Provision the rules:

The given rule is provisioned with the rulegroup. The variable declarations provisioned in (1) of the parent domain of the rulegroup need to be utilized to verify that the rule being provisioned is valid. For example, the condition (x.b < 3) can be verified as being valid, since "x" has a record type, and has "b" as a field, and "x.b" is a TpFloat. As an example, if the type of "x.b" had been TpString, then during provisioning, the rule condition would have been determined to be as invalid, and an exception thrown. The steps for creating the group are not shown in this example.

```
// commit transaction
manager.commitTransaction();
```

4) Sending a decision request:

The first three steps happen during provisioning time. In this step, we describe how the client may use the IpPolicyDomain.evalPolicy() method, as well as the notion of signatures, to request a decision to be rendered. We consider two scenarios: 1) where the value of x is explicitly specified by the client, and 2) where it is not.

- Case 1:

```
TpAny x_value = {4, 2.7};
TpPolicyNameValue x_name_val = {"x", x_value};
TpPolicyNameValueList inputs = [x_name_value]; // input values

TpPolicyNameValueList outputs = domain.evalPolicy("test_sig", inputs);
```

Here, the explicit value of x overrides the value of x set via setVariableValue(). Hence, before rules are evaluated for this decision, the value of x is set to {4, 2.7}. The rule condition will then be true, and the value of z will be set to 4. Hence the outputs list will contain the value of y as being 4.

Note that if the value of x was specified as:

```
TpAny x_value = {4, 9.0};
```

The rule condition would not be true, which implies that the rule action would not be executed. However, the signature "sig_test" specified that y was an output variable and hence its value was to be sent back to the client. However (as

mentioned earlier in our assumptions about variable semantics), y started out as being uninitialized, and hence an exception would be returned back to the client.

- Case 2:

```
TpPolicyNameValueList inputs = []; // input values

TpPolicyNameValue outputs = domain.evalPolicy("test_sig", inputs);
```

Here, the explicit value of x is not set. Hence the value of x set via setVariableValue() is used during rule evaluation, which implies that y will be set to the value 1. As in the first case, the outputs list will contain one element, which would be the value of variable y.

## 11.5 Example XML Scenarios

We now present a high-level scenario that illustrates how the XML extensions are tied together. The rulegroup that we will use contains only one rule and is part of a domain named "testdomain". The rule is given below in a pseudo-language:

```
if (SIPAddress inDomain "parlay.org")
then
    setCallLegProperty(P_CALL_LEG_PROPERTY_INFO,"http://www.parlay.org")
end
```

The example rule above invokes the operator "inDomain". We assume that this operation compares the domain part of an URI. It evaluates to "true" if the URI operand is part of a given domain. If the condition holds, the call leg property named P_CALL_LEG_PROPERTY_INFO will be set to "http://www.parlay.org ".

The action and condition part of this rule are expressed in pseudo XML below (i.e. namespaces are omitted, etc.). XML schema reference is not shown which would define XML structure, types and operations.

```
Action (to be passed in a ConditionAttribute.AttributeValue)

    <condition operator="inDomain">
        <operand>
            <variable name="SIPAddress" type="anyURI"/>
        </operand>
        <operand>
            <constant value="www.parlay.org" type="string"/>
        </operand>
    </condition>

Condition (to be passed in an ActionAttribute.AttributeValue)

    <action>
        <setCallLegProperty>
            <callLegProperty value="P_CALL_LEG_PROPERTY_INFO" type="CallLegProperties"/>
            <constant value="http://www.parlay.org" type=""string"/>
        </setCallLegProperty>
    </action>
```

Now, assume that the value of the variable with the name "SIPAddress" is to be supplied for rule evaluation, and the value of XML element is to be returned back to the client. The steps that need to be performed are as follows given below (we will give pseudo-code for all the steps):).

5) Provision variables:

```
// get the manager
IpPolicyManagerRef manager = …;

// start transaction
manager.startTransaction();

// get the domain
IpPolicyDomainRef domain = manager.getDomain("testdomain");

// create a variable set
domain.createVariableSet("vset");

// define the type of the variable named "SIPAddress"
TpPolicyType URI_type = TpPolicyType(TpStructuredAttributeTypeInfo("P_com/vendor/TpURI"));
```

```
// define the type of the action
TpPolicyType action_type = TpPolicyType(TpXMLString);

// create the variables in the variable set
domain.createVariable("vset", "SIPAddress", URI_type);
domain.createVariable("vset", "setCallLegProperty", action_type);

// set the values of x and y
TpAny URI_value = "sip:jdoe@parlay.org";
TpAny action_value = "<setCallLegProperty/>";
domain.setVariableValue("vset", "SIPAddress", URI_value);
domain.setVariableValue("vset", "setCallLegProperty", action_value);
```

6) Create signature:

```
IpPolicySignatureRef sig = domain.createSignature("test_sig");

// set input and output variables
TpStringSet input_vars = ["SIPAddress"];
TpStringSet output_vars = ["setCallLegProperty"];
sig.setInputVariables(input_vars);
sig.setOutputVariables(output_vars);

// set groups and roles
TpStringSet groups = ["testgroup"];
TpStringSet roles = []; // no roles specified
sig.setGroupNames(groups);
sig.setRoleNames(roles);
```

Provisioning and decision requests go much the same way as in steps 7 and further in clause 11.4.

# 12 Policy Management Exception Classes

The following are the list of exception classes which are used in this interface of the API.

| Name | Description |
|---|---|
| P_ACCESS_VIOLATION | Thrown if the client does not have authorization to invoke this method on this object with these parameters. |
| P_SYNTAX_ERROR | Thrown if the specified name is formatted improperly. |
| P_NAME_SPACE_ERROR | Thrown if the specified name matches or does not match the name of an existing object of the appropriate type within this container. |
| P_NO_TRANSACTION_IN_PROCESS | Thrown if there is currently no transaction in process. |
| P_TRANSACTION_IN_PROCESS | Thrown if there is currently a transaction in process. Note that transactions can not be nested, that is, a second call to startTransaction() without calling commitTransaction() or abortTransaction() in between will result in this exception being thrown during the second call. |

Each exception class contains the following structure:

| Structure Element Name | Structure Element Type | Structure Element Description |
|---|---|---|
| ExtraInformation | TpString | Carries extra information to help identify the source of the exception, e.g. a parameter name. |

# Annex A (normative): OMG IDL Description of Policy Management SCF

The OMG IDL representation of this interface specification is contained in text files (policy_data.idl and policy_interfaces.idl contained in archive 2919813V800IDL.ZIP) which accompany the present document.

# Annex B (informative): W3C WSDL Description of the Policy Management SCF

The W3C WSDL representation of this interface specification is contained in zip file 2919813V800WSDL.ZIP, which accompanies the present document.

# Annex C (informative): Java API Description of the Policy Management SCF

The Java API realisation of this interface specification is produced in accordance with the Java Realisation rules defined in Part 1 of this specification. These rules aim to deliver for Java, a developer API, provided as a realisation, supporting a Java API that represents the UML specifications. The rules support the production of both J2SE and J2EE versions of the API from the common UML specifications.

The J2SE representation of this interface specification is provided as Java Code, contained in archive 2919813V800J2SE.ZIP that accompanies the present document.

The J2EE representation of this interface specification is provided as Java Code, contained in archive 2919813V800J2EE.ZIP that accompanies the present document.

# Annex D (informative): Description of Policy Management for 3GPP2 cdma2000 networks

This annex is intended to define the OSA API Stage 3 interface definitions and it provides the complete OSA specifications. It is an extension of OSA API specifications capabilities to enable operation in cdma2000 systems environment. They are in alignment with 3GPP2 Stage 1 requirements and Stage 2 architecture defined in

[1] 3GPP2 P.S0001-B: "Wireless IP Network Standard", Version 1.0, September 2000;

[2] 3GPP2 S.R0037-0: "IP Network Architecture Model for cdma2000 Spread Spectrum Systems", Version 2.0, May 14, 2002;

[3] 3GPP2 X.S0013: "All-IP Core Network Multimedia Domain", December 2003.

These requirements are expressed as additions to and/or exclusions from the 3GPP specification. The information given here is to be used by developers in 3GPP2 cdma2000 network architecture to interpret the 3GPP OSA specifications.

## D.1 General Exceptions

The term UMTS is not applicable for the cdma2000 family of standards. Nevertheless these terms are used (3GPP TR 21.905) mostly in the broader sense of "3G Wireless System". If not stated otherwise there are no additions or exclusions required.

CAMEL and CAP mappings are not applicable for cdma2000 systems.

## D.2 Specific Exceptions

### D.2.1 Clause 1: Scope

There are no additions or exclusions.

### D.2.2 Clause 2: References

Normative references on 3GPP TS 23.078 and on 3GPP TS 29.078 are not applicable for cdma2000 systems.

### D.2.3 Clause 3: Definitions and abbreviations

There are no additions or exclusions.

### D.2.4 Clause 4: Policy Management SCF

There are no additions or exclusions.

### D.2.5 Clause 5: Sequence Diagrams

There are no additions or exclusions.

### D.2.6 Clause 6 Class Diagrams

There are no additions or exclusions.

### D.2.7 Clause 7: The Service Interface Specifications

There are no additions or exclusions.

### D.2.8 Clause 8: Policy Management Interface Classes

There are no additions or exclusions.

### D.2.9 Clause 9: State Transition Diagrams

There are no additions or exclusions.

### D.2.10 Clause 10: Data Definitions

There are no additions or exclusions.

### D.2.11 Clause 11: Policy Management Exception Classes

There are no additions or exclusions.

### D.2.12 Annex A (normative): OMG IDL Description of Policy Management SCF

There are no additions or exclusions.

# Annex E (informative): Change history

| Change history | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Date** | **TSG #** | **TSG Doc.** | **CR** | **Rev** | **Subject/Comment** | **Old** | **New** |
| Mar 2007 | CT_35 | CP-070047 | 0016 | -- | Update document for conversion to Release 7 | 6.4.1 | 7.0.0 |
| Dec 2008 | CT_42 | | | | Upgraded unchanged from Rel-7 | 7.0.0 | 8.0.0 |
| | | | | | | | |
| | | | | | | | |
| 2009-12 | - | - | - | - | Update to Rel-9 version (MCC) | 8.0.0 | 9.0.0 |