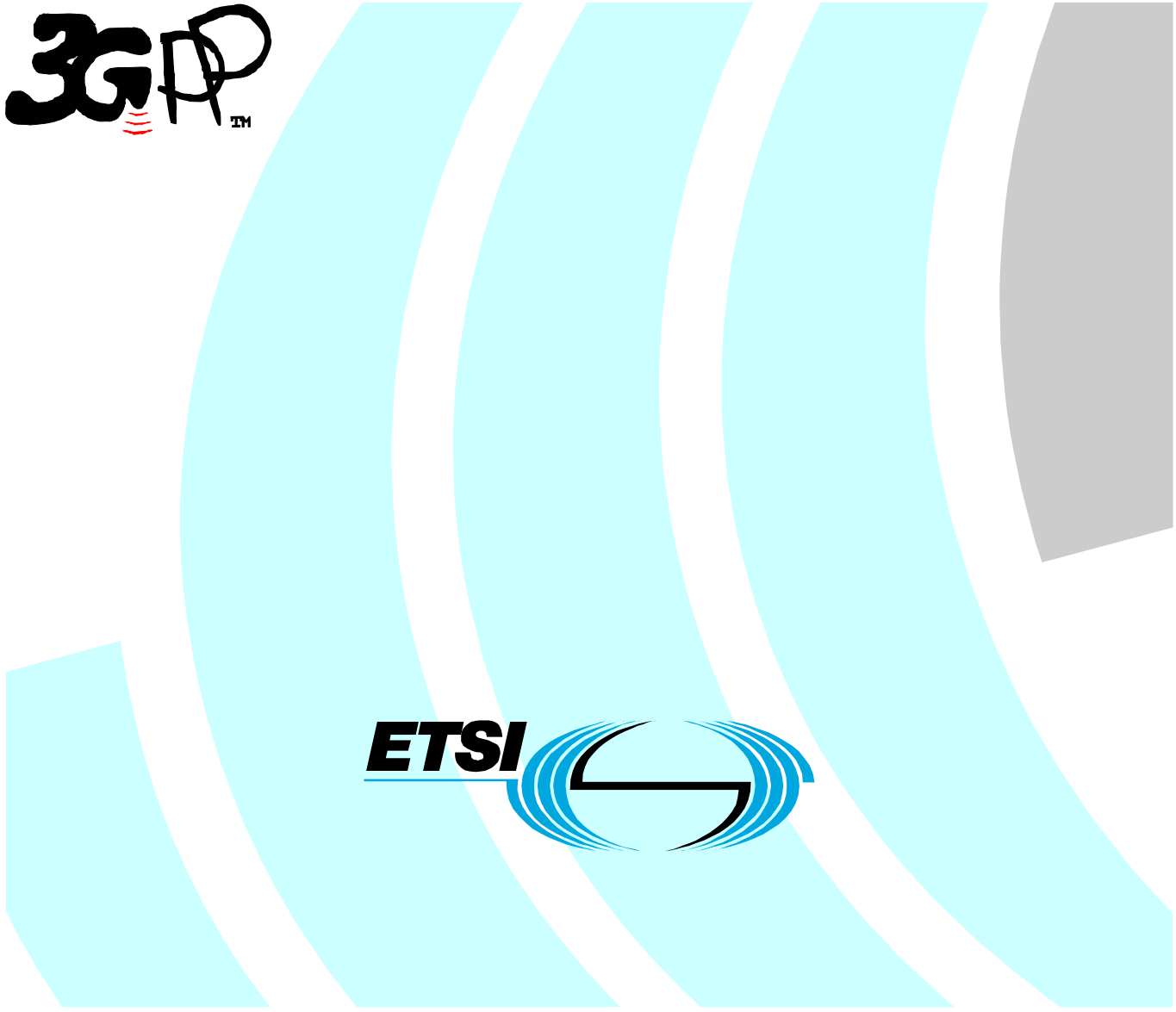
ETSI TS 129 198-13 V5.2.0 (2003-09)

Technical Specification

Universal Mobile Telecommunications System (UMTS);
Open Service Access (OSA);
Application Programming Interface (API);
Part 13: Policy management SCF
(3GPP TS 29.198-13 version 5.2.0 Release 5)



Reference
RTS/TSGN-0529198-13v520

Keywords
UMTS

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

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

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp .

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 SCF	
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 5)
Part 10:	Connectivity Manager SCF	(not part of 3GPP Release 5)
Part 11:	Account Management SCF	
Part 12:	Charging SCF	
Part 13 :	**Policy Management SCF**	(new in 3GPP Release 5)
Part 14 :	Presence and Availability Management SCF	(new in 3GPP Release 5)

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	29.998-04-1	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	User Status and User Location – MAP 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*

The present document is equivalent to ETSI ES 202 915-13 v1.1.1 (Parlay 4.0).

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 23.127 [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 CN WG5, ETSI SPAN 12 and the Parlay Consortium, in co-operation with a number of JAIN™ Community member companies.

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: "Stage 1 Service Requirement for the Open Service Access (OSA) (Release 5)".

[3] 3GPP TS 23.127: "Virtual Home Environment (Release 5)".

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.

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 and control access to it a policy management service is needed. Consistent with this, a policy management service interface manager, IpPolicyManager, has been defined. All policy management interfaces are accessible from IpPolicyManager.

A number of APIs have been defined to obtain services from a policy management service. These include APIs to create, update or view policy information. Additionally APIs have been defined to facilitate interactions between clients (e.g., a 3rd party application) and any policy enabled service. These include APIs to view policy events, to subscribe to policy events and for the generation of events by clients. All APIs conform to an underlying policy information model.

Clients that perform administrative tasks, e.g., create, update or delete policy information must obtain access to IpPolicyManager using the family of obtainInterface() methods supported by the IpAccess interface. Administrative tasks may be performed through methods supported by IpPolicyManager.

Clients that need to interact with a specific policy enabled service (for non-administrative tasks) can obtain access to that service's interface directly via the selectService() method supported by the IpAccess interface. It should be noted that specific policy enabled services may support additional interfaces and methods that are not defined below. 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.

The order is as follows:

- The Sequence diagrams give the reader a practical idea of how each of the SCF is implemented.
- The Class relationships clause show how each of the interfaces applicable to the SCF, relate to one another.
- The Interface specification clause describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the transition between states in the SCF. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.
- 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.

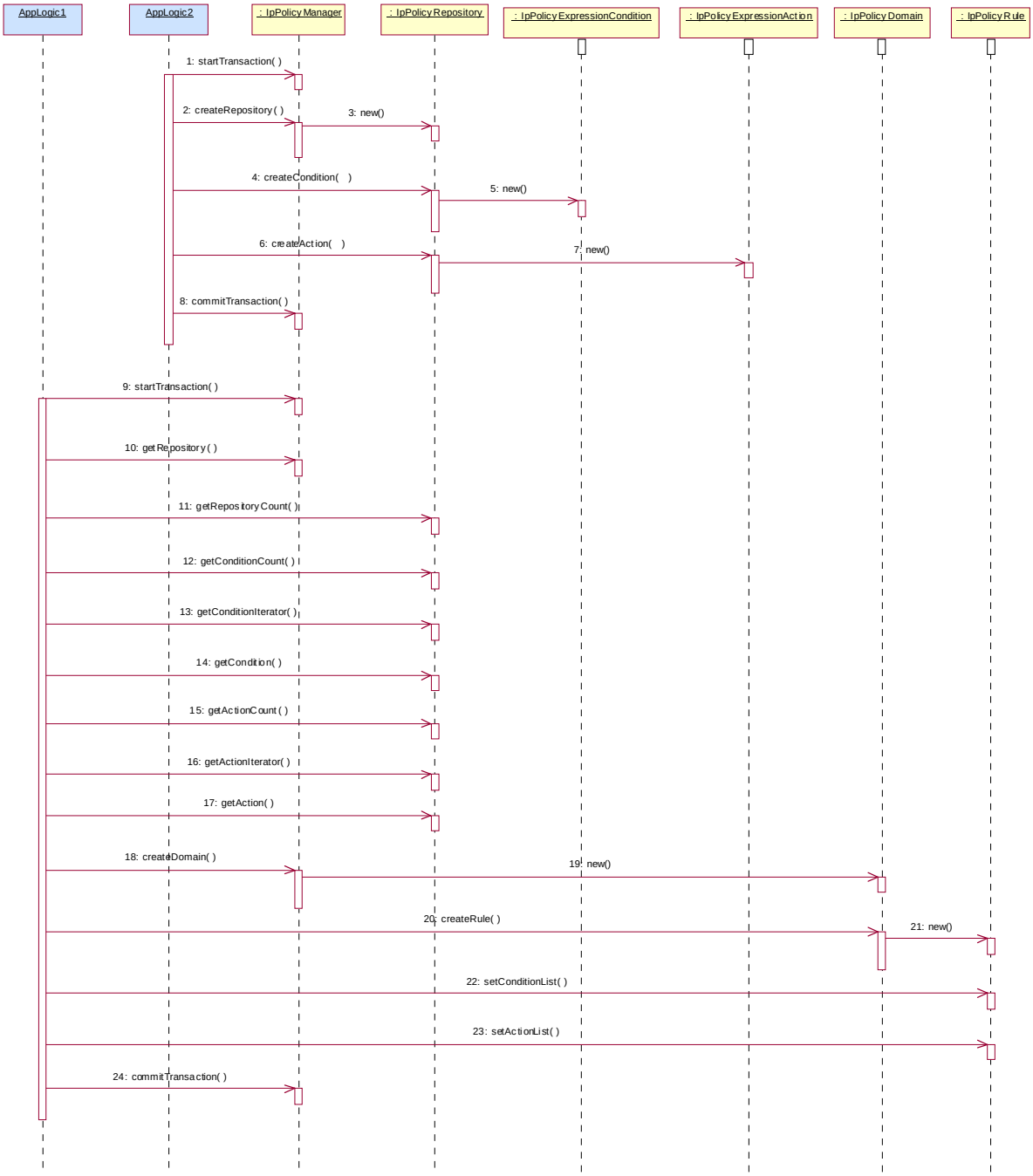
An implementation of this API which supports or implements a method described in the present document, shall support or implement the functionality described for that method, for at least one valid set of values for the parameters of that method. Where a method is not supported by an implementation of 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 AppLogic2 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 & 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.

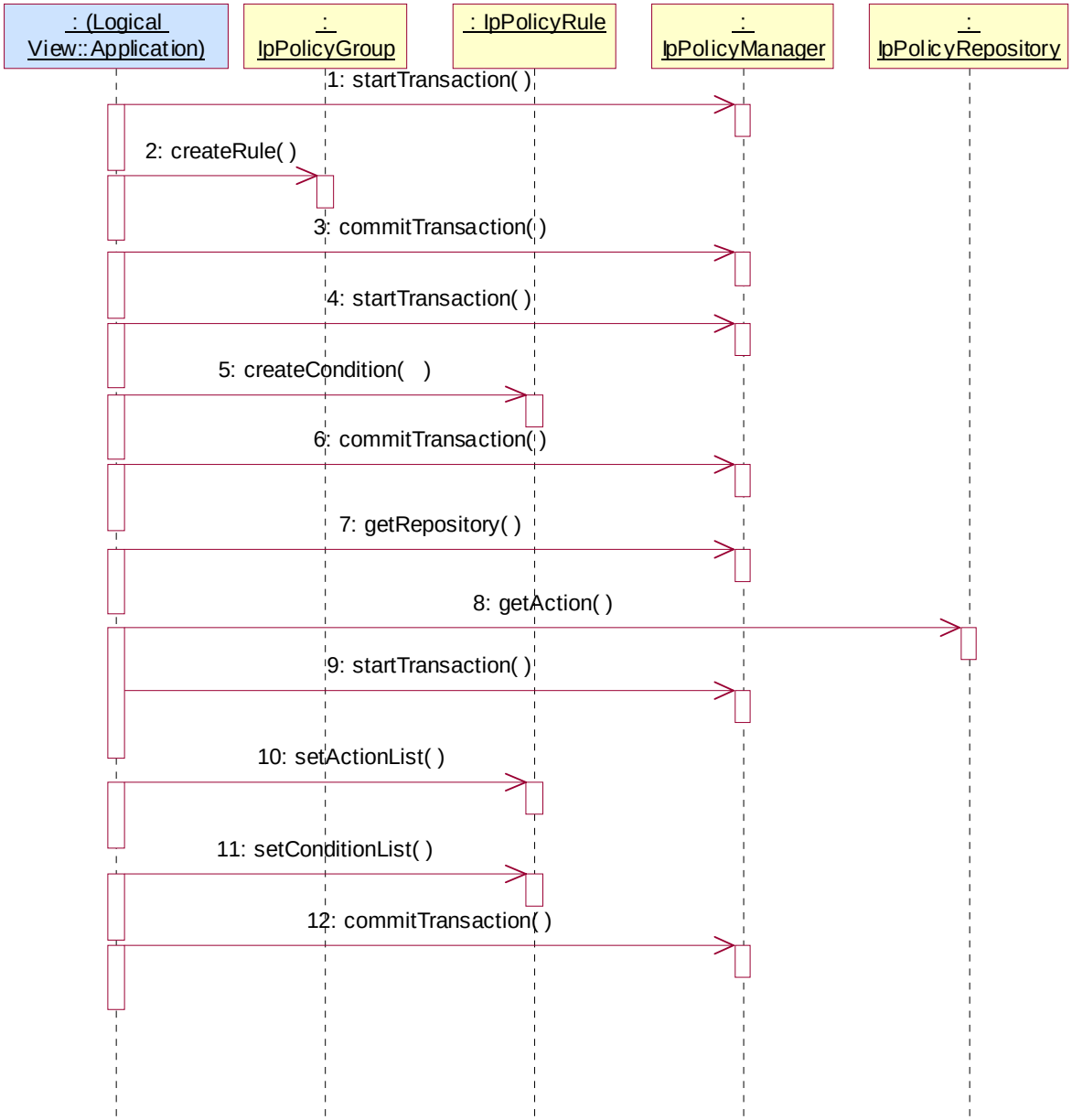
This sequence includes:

- 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&modify" domain.



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.

As preliminary to the invocation of "createCondition", the application should perform the following activities:

1) Create a TpAttribute, with AttributeName: "Expression", AttributeType: P_STRING, AttributeValue:

"<the condition expression to be evaluated>"

2) Add the TpAttribute from 1) to a new TpAttributeSet as its sole element

After having performed these steps the application can call the method createCondition() on the appropriate repository or rule, passing in the name of the condition, the type of the condition IpPolicyExpressionCondition, and the TpAttributeSet created in 2). Note that this call may throw an exception if the expression defined in 1) is not parsable according to the published BNF.

Creating IpPolicyExpressionAction is done similarly.

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 IpPolicyCondition 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 & receive an event

This sequence shows how policy events are used.

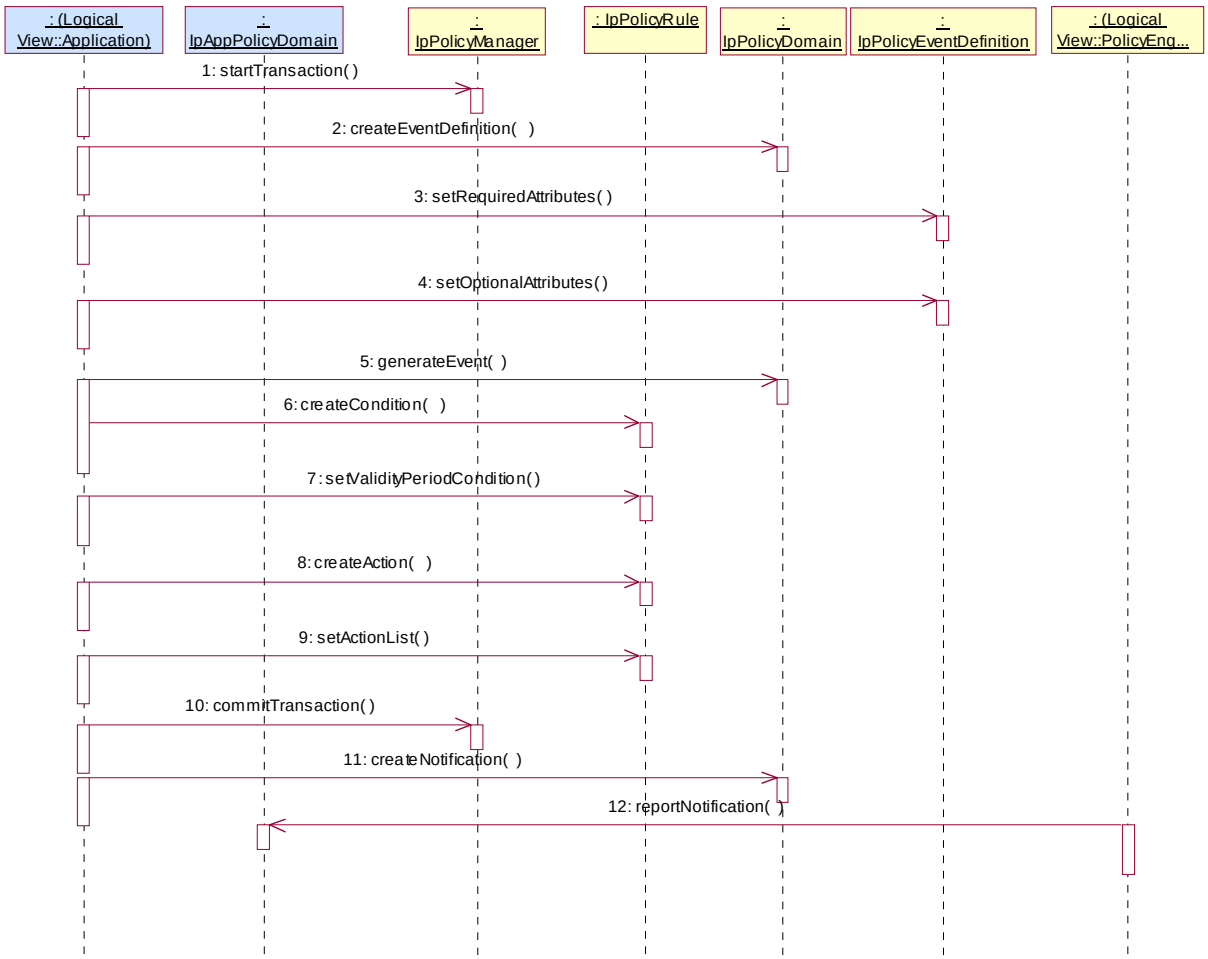
For clarification we list the different policy related objects used:

- 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 method can be used to test the newly created event by passing a attribute set and checking whether the expected event is generated.

6: This createCondition() method creates locally a PolicyTimePeriodCondition defining the validity period of this rule.

Returns a reference to the new IpPolicyTimePeriodCondition object.

As preliminary to the invocation of "createCondition", the application should perform the following activities:

1) Create a set of TpAttribute setting the different time and dates applying to this condition. For instance, one attribute might be defined as:

TpAttribute.AttributeName (type: TpString)=TimePeriod

TpAttribute.AttributeType= P_STRING

TpAttribute.AttributeValue= "20000101T080000/20000131T120000"

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

2) Add the set of TpAttributes from 1) to a new TpAttributeSet. This will be passed with createCondition().

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

8: 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.

9: 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.

10: This closes the transaction bracket.

11: Now -- independently of the activities before -- the application can register with the policy domain for events of a certain type. If such an event occurs (as a result of rule's action) the application is notified.

Passed parameters are the callback interface reference and the list of event types the application is interested in.

Returns a sessionID.

12: In the policy engine complex, a certain event action is performed leading to an event the application registered for. In that case, the application is notified via the callback interface whose reference has been sent with enablePolicyNotification().

Parameters are the sessionID relating the this notification to the specific enablePolicyNotification()-call and the policyEvent arising.

5.4 Create & 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 to be created domain object 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.

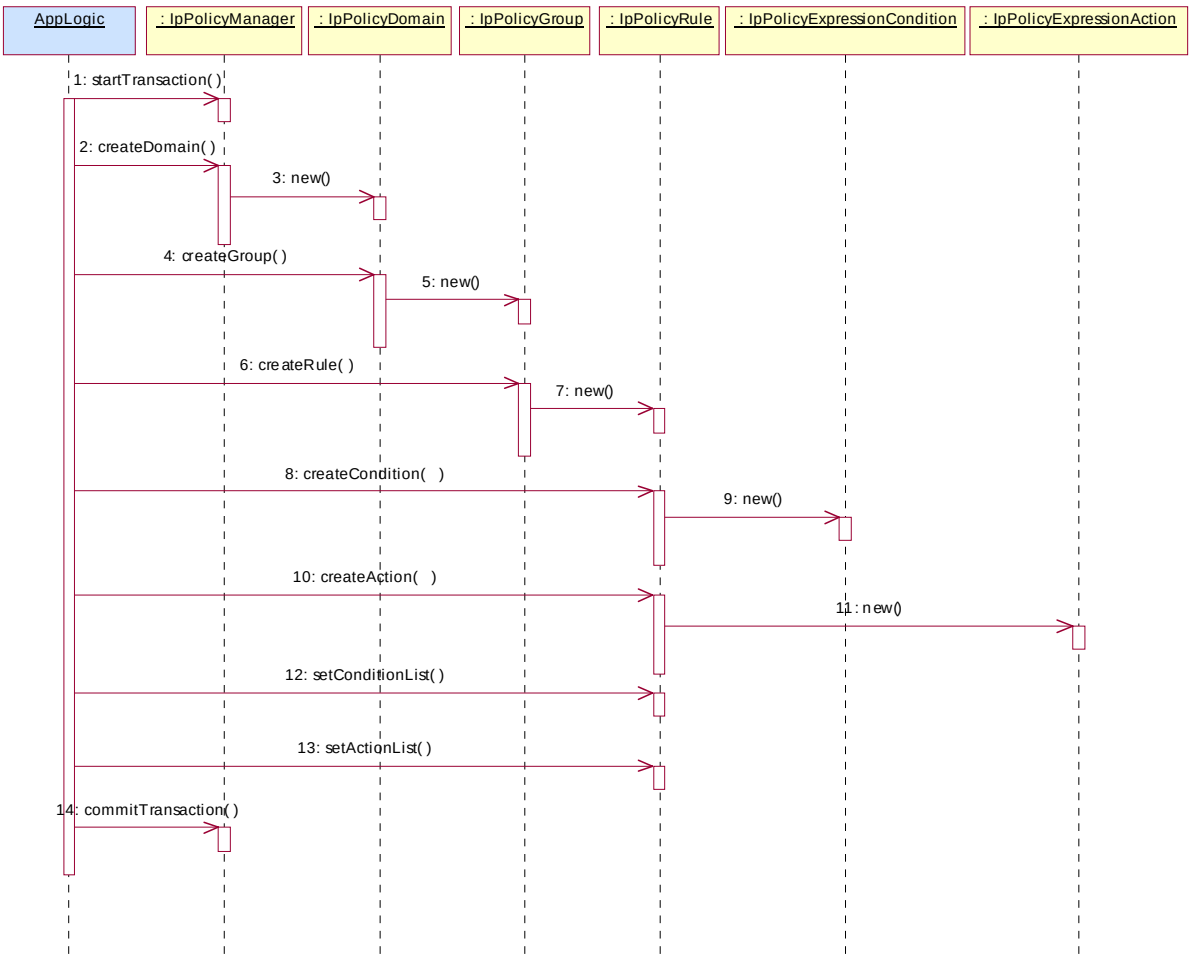
9: Another action is to remove a rule. We assume here that the name of the rule (which is passed as parameter) is already known. Otherwise one has to retrieve the name by using the IpRuleIterator interface (the reference is got with getRuleIterator()).

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 == NULL; no roles defined
- ConditionListType == P_PM_DNF; disjunctive normal form (DNF)
- SequencedActions == 3; don't 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.AttributeType = "P_STRING"
- ConditionAttribute.AttributeValue = "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.AttributeType = "P_STRING"
- ActionAttribute.AttributeValue = "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:

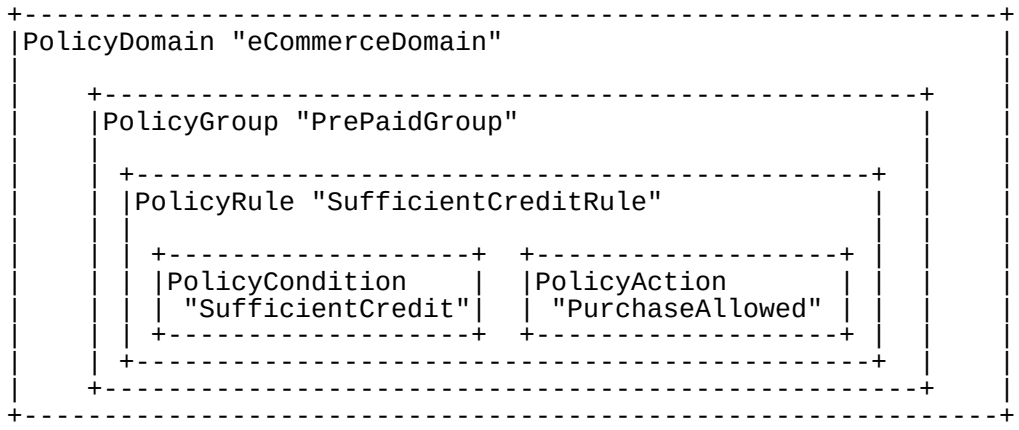
- actionList.Action == <reference to the IpPolicyAction interface returned by step 10>

- actionList.SequenceNumber == 1;

14: The "SufficientCreditRule" now exists in the "PrePaidGroup" of the "eCommerceDomain". 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 evaluate the rule.

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.



6 Class Diagrams

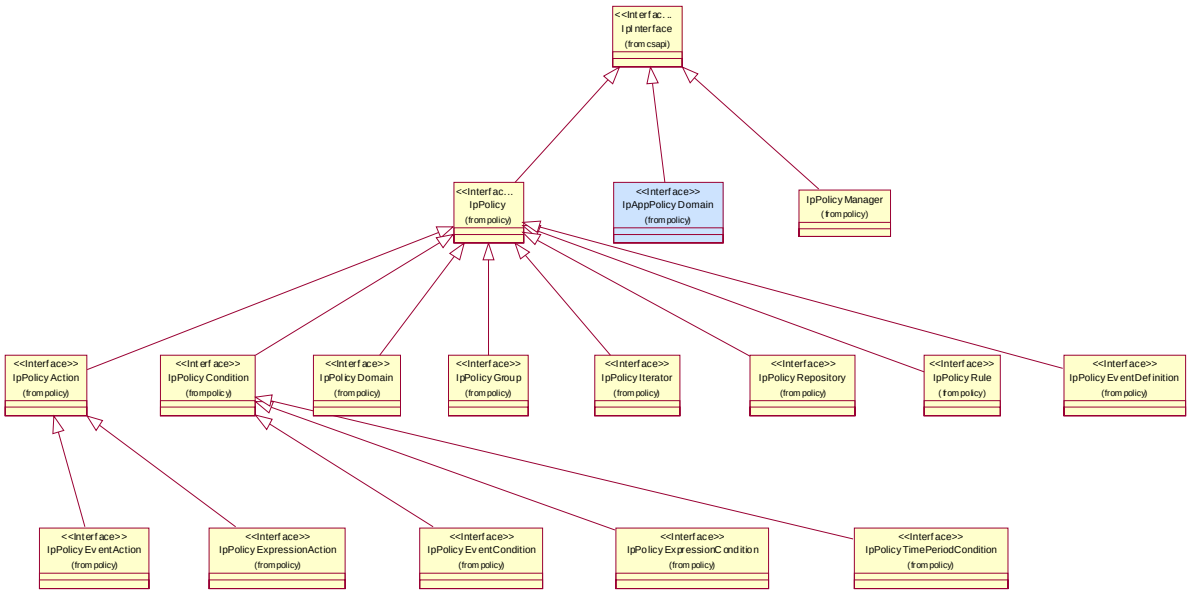


Figure: Policy Classes

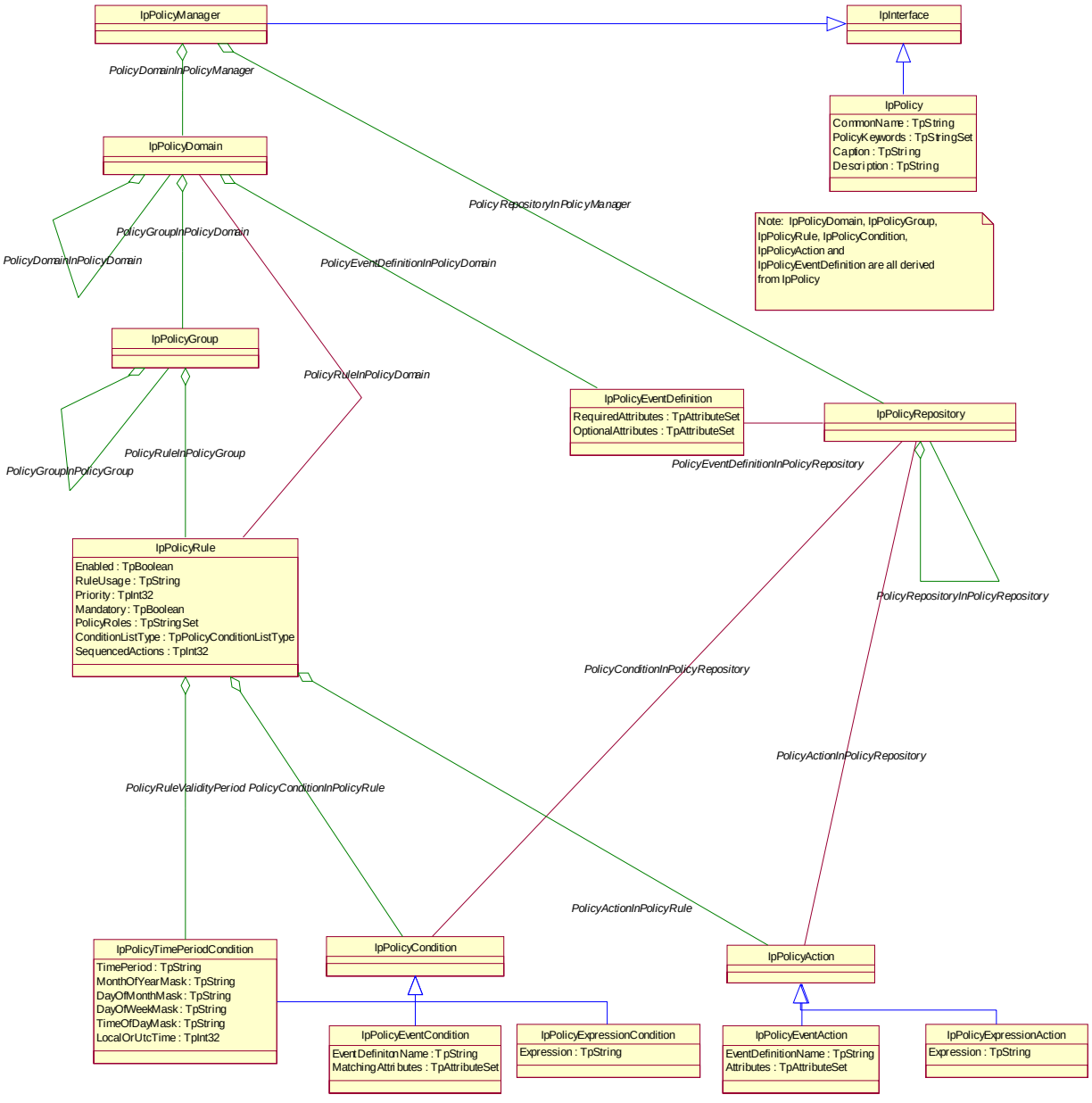


Figure: Policy Management Information Model

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.

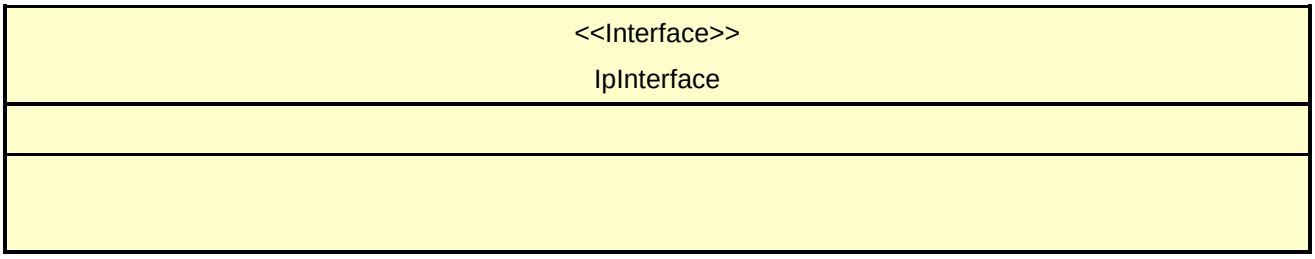
7.1.4 State Model

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.



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.

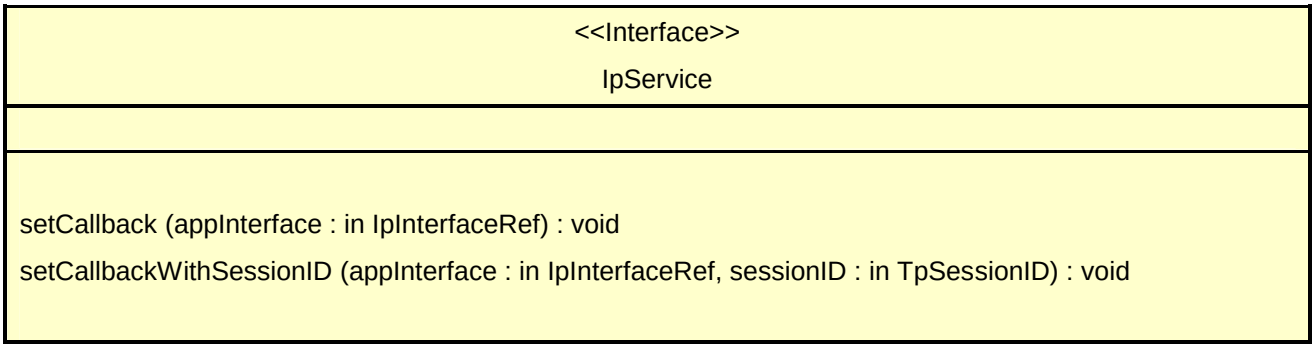
The interfaces that are implemented by the services are denoted as 'Service Interface'. The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as 'Application Interface'.

7.4 Generic Service Interface

7.4.1 Interface Class IpService

Inherits from: IpInterface

All service interfaces inherit from the following interface.



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.

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.

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 Interface Classes

The Policy Management APIs defined below 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.

· Publishing of policy events supported by a service.

· Subscription to policy events supported by a service.

· Generation of events.

· Obtaining statistics associated with the use of policies.

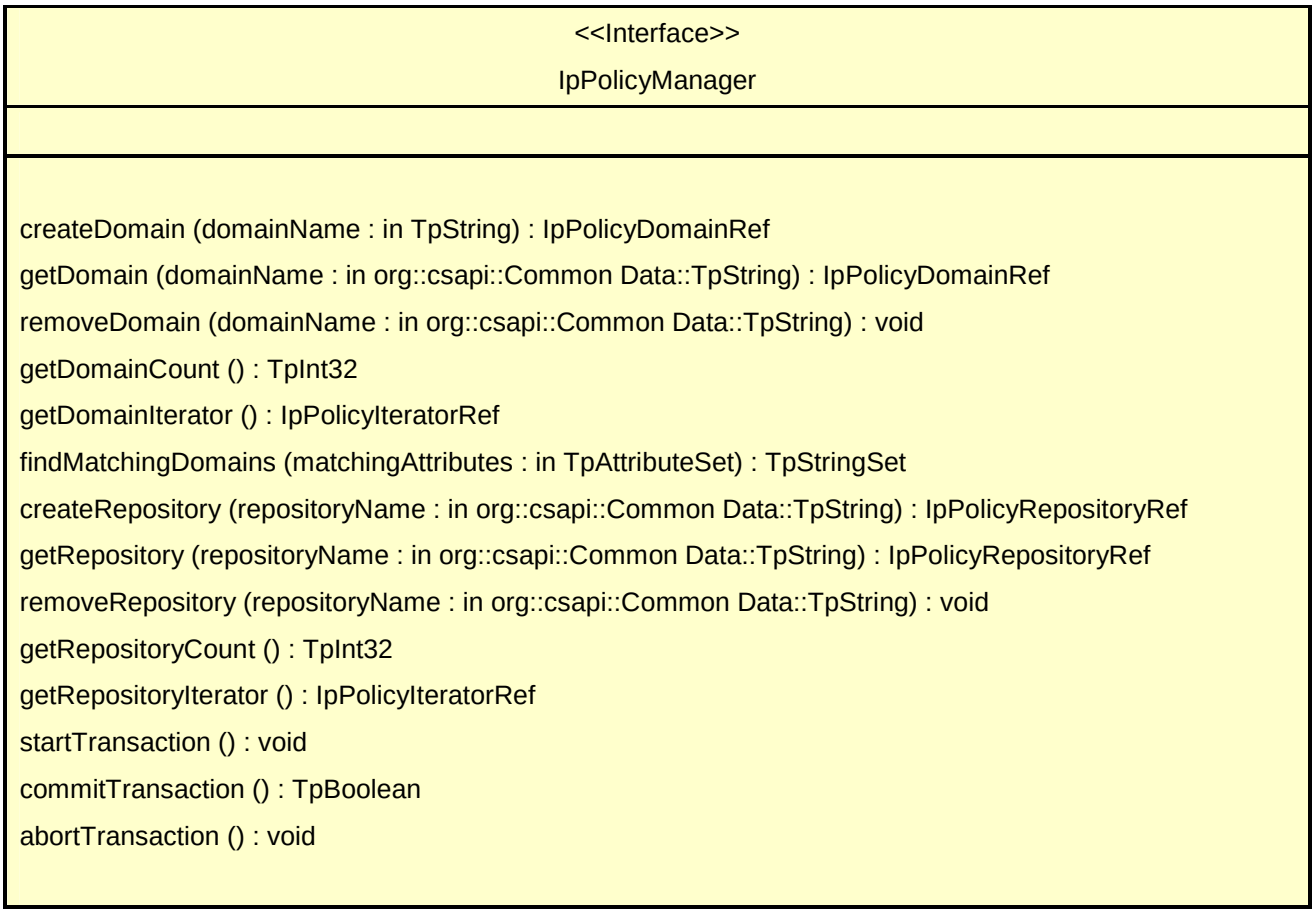
· Handling of service level agreements (SLA). SLAs may be used to convey authorisation for access or subscription to policy information or to modify or create policy information.

8.1 Interface Class IpPolicyManager

Inherits from: IpInterface.

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 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

8.1.2 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

8.1.3 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

8.1.4 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

8.1.5 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

8.1.6 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

8.1.7 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

8.1.8 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

8.1.9 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

8.1.10 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

8.1.11 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

8.1.12 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

8.1.13 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

8.1.14 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

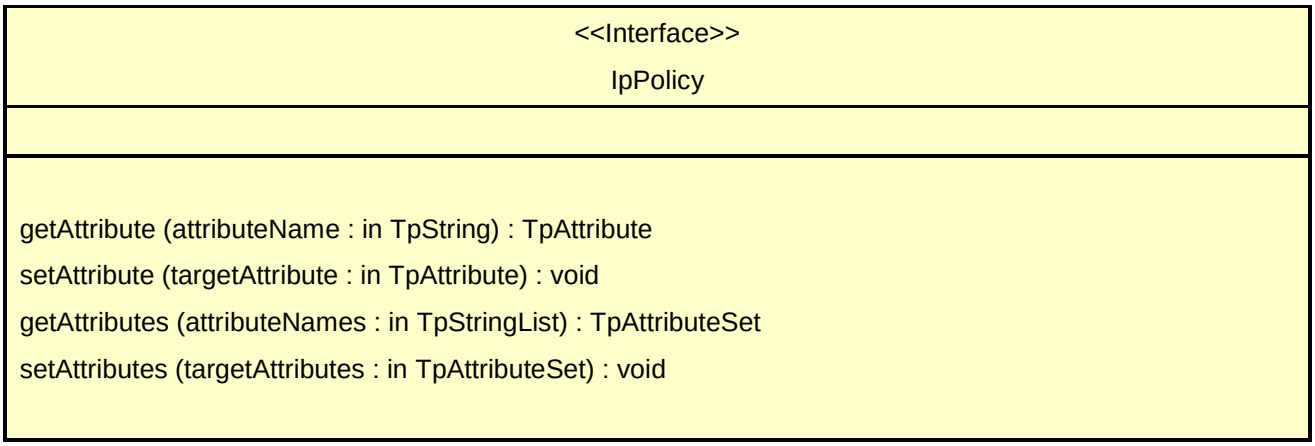
TpCommonExceptions, P_NO_TRANSACTION_IN_PROCESS

8.2 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.



8.2.1 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:

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

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

This 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 [PCIM].

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.

8.2.2 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

8.2.3 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

8.2.4 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

8.2.5 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

8.3 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 Figure 5 below. Note that a PolicyDomain can nest other PolicyDomains, and there is no restriction on the depth of the nesting in sibling PolicyDomains.

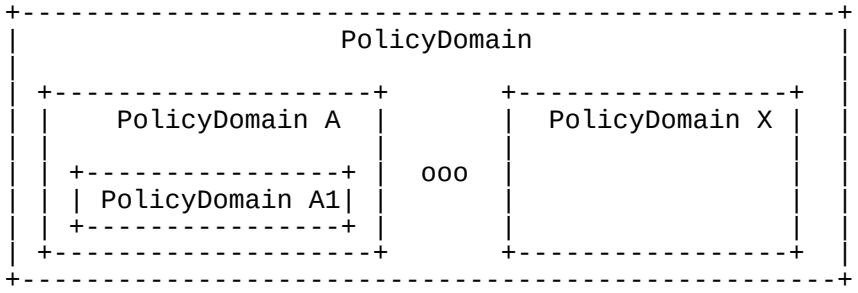


Figure Overview of the PolicyDomain class

As a simple example, think of the highest level PolicyDomain shown in Figure 5 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 Figure 5 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

generateEvent (eventDefinitionName : in TpString, attributes : in TpAttributeSet) : void

createNotification (appPolicyDomain : in IpAppPolicyDomainRef, events : in TpStringSet) : TpAssignmentID

destroyNotification (assignmentID : in TpAssignmentID, events : in TpStringSet) : void

createVariableSet (variableSetName : in TpString) : void

getVariableSet (variableSetName : in TpString) : TpAttributeSet

removeVariableSet (variableSetName : in TpString) : void

getVariableSetCount () : TpInt32

getVariableSetIterator () : IpPolicyIteratorRef

setVariable (variableSetName : in TpString, variable : in TpAttribute) : void

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

8.3.1 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:

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

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

This 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 [PCIM].

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).

8.3.2 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

8.3.3 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

8.3.4 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

8.3.5 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

8.3.6 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

8.3.7 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

8.3.8 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

8.3.9 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

8.3.10 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

8.3.11 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

8.3.12 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

8.3.13 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

8.3.14 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

8.3.15 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

8.3.16 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

8.3.17 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

8.3.18 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

8.3.19 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

8.3.20 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

8.3.21 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

8.3.22 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

8.3.23 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.

See also: IpPolicyEventAction

Parameters

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

8.3.24 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

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

8.3.25 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

8.3.26 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

8.3.27 Method getVariableSet()

Get a variable set.

Returns: A variable set.

Parameters

variableSetName : in TpString

The name of the variable set to get.

Returns

TpAttributeSet

Raises

TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR

8.3.28 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

8.3.29 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

8.3.30 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

8.3.31 Method setVariable()

Set a variable within a variable set.

Parameters

variableSetName : in TpString

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

variable : in TpAttribute

The variable to set.

Raises

TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR, P_NO_TRANSACTION_IN_PROCESS

8.3.32 Method getVariable()

Get a copy of a variable from a variable set.

Returns: A copy of the variable.

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

TpAttribute

Raises

TpCommonExceptions, P_ACCESS_VIOLATION, P_SYNTAX_ERROR, P_NAME_SPACE_ERROR

8.4 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 Figure 5 below. Note that a PolicyGroup can nest other PolicyGroups, and there is no restriction on the depth of the nesting in sibling PolicyGroups.

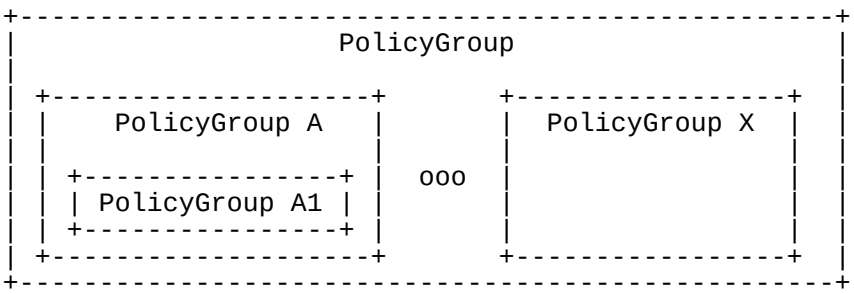


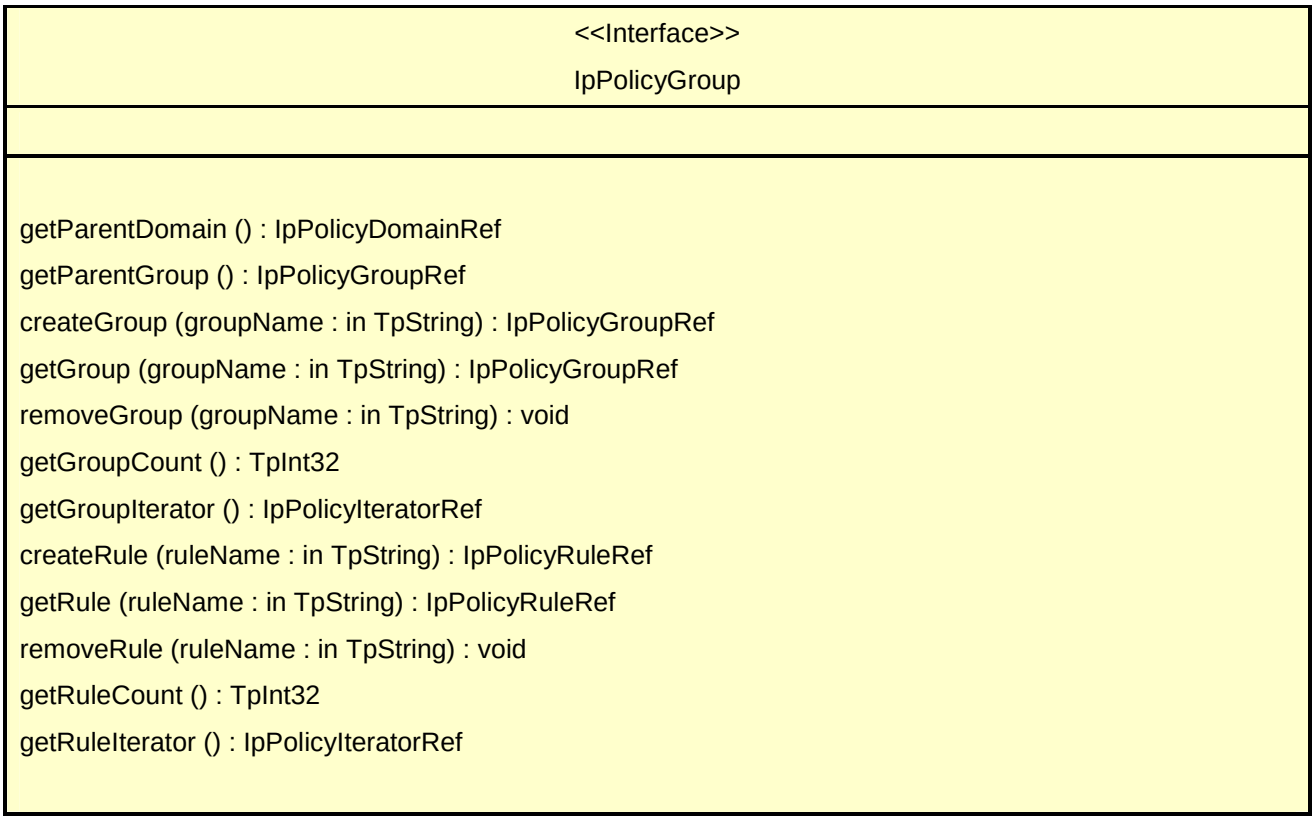
Figure Overview of the PolicyGroup class

As a simple example, think of the highest level PolicyGroup shown in Figure 5 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 Figure 5 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.



8.4.1 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:

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

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

This 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 [PCIM].

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.

8.4.2 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

8.4.3 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

8.4.4 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

8.4.5 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

8.4.6 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

8.4.7 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

8.4.8 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

8.4.9 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

8.4.10 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

8.4.11 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

8.4.12 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

8.4.13 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

8.5 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>>

IpPolicyRepository

getParentRepository () : IpPolicyRepositoryRef

createRepository (repositoryName : in TpString) : IpPolicyRepositoryRef

getRepository (repositoryName : in TpString) : IpPolicyRepositoryRef

removeRepository (repositoryName : in TpString) : void

getRepositoryCount () : TpInt32

getRepositoryIterator () : IpPolicyIteratorRef

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

8.5.1 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:

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

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

This 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 [PCIM].

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.

8.5.2 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

8.5.3 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

8.5.4 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

8.5.5 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

8.5.6 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

8.5.7 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

8.5.8 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

8.5.9 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

8.5.10 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

8.5.11 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

8.5.12 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

8.5.13 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

8.5.14 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

8.5.15 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

8.5.16 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

8.5.17 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

8.6 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.

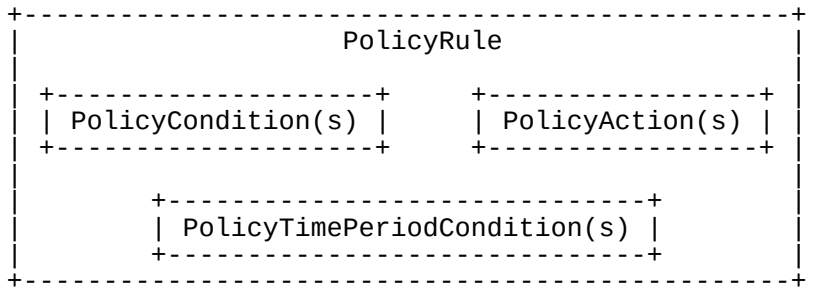


Figure Overview of the PolicyRule Class

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 a 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 "don't 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

8.6.1 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:

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

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

This 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 [PCIM].

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 [PCIM], 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:

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

o 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.

o dontCare(3): Do them -- I don't 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).

8.6.2 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`

8.6.3 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

8.6.4 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

8.6.5 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

8.6.6 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

8.6.7 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

8.6.8 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

8.6.9 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

8.6.10 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

8.6.11 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

8.6.12 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

8.6.13 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

8.6.14 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

8.6.15 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

8.6.16 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

8.6.17 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

8.6.18 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

8.6.19 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

8.6.20 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

8.6.21 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

8.7 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.

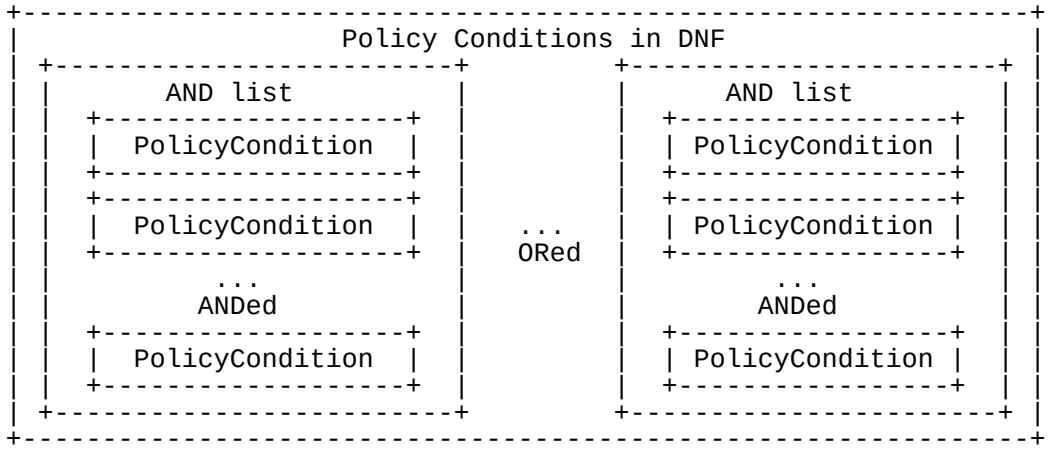


Figure Overview of Policy Conditions in DNF

This figure 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.

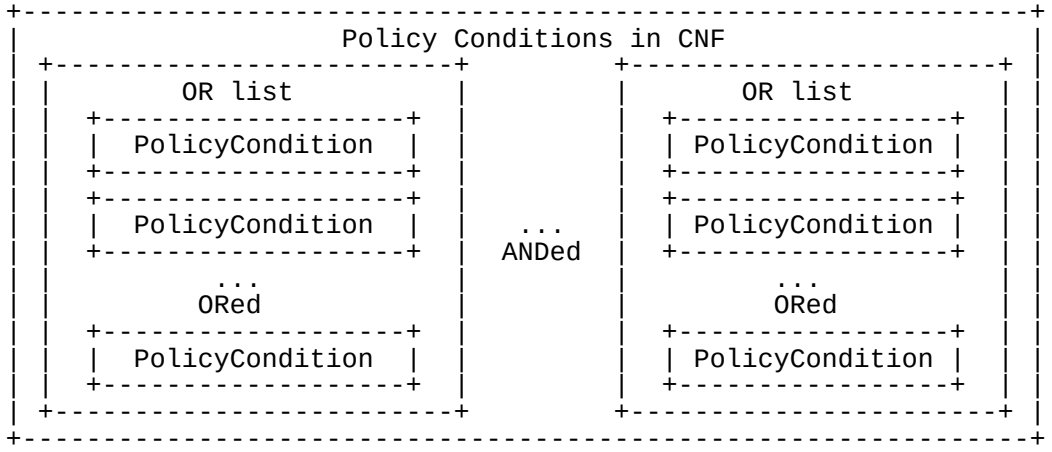
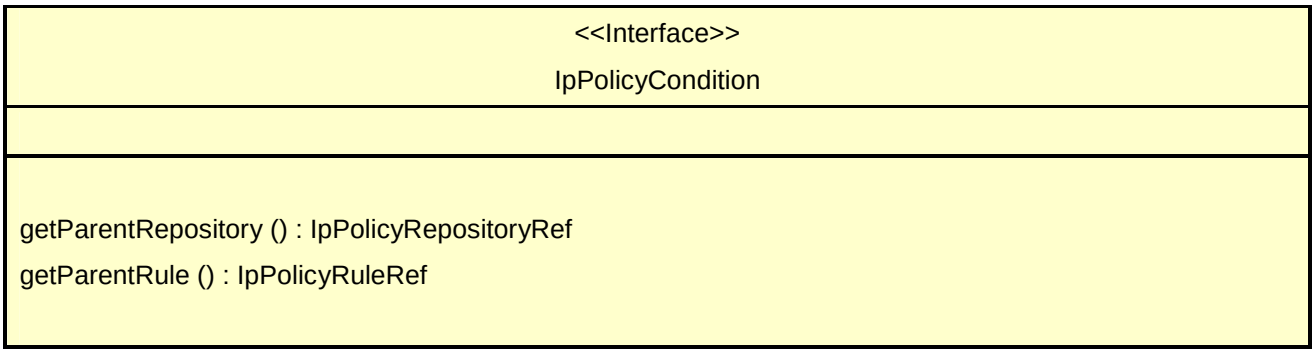


Figure 8. Overview of Policy Conditions in CNF

In this figure, 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.



8.7.1 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:

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

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

This 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 [PCIM].

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.

8.7.2 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

8.7.3 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

8.8 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 PCIM, 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.

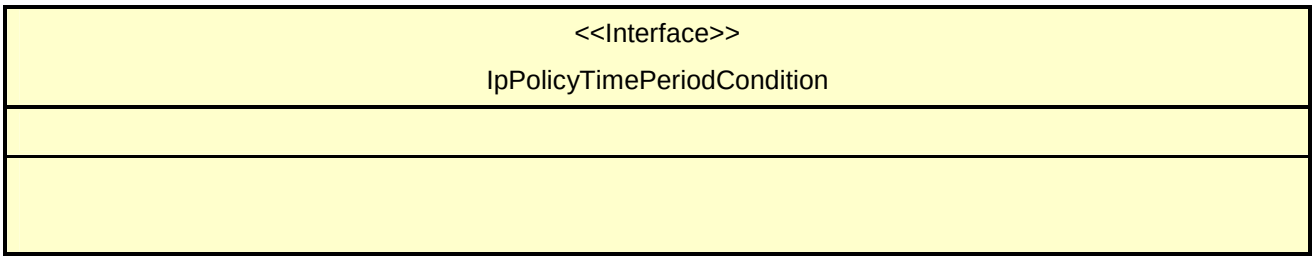
Instances of this class may have up to five attributes identifying time periods at different levels. The values of all the attributes present in an instance are ANDed together to determine the validity period(s) for the instance. For example, an instance with an overall validity range of January 1, 2000 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.



8.8.1 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:

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

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

This 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 [PCIM].

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 (reference [10]): 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.

o 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 '/'.

o 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 PCIM 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 section 5.4 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 reference [8] for more details on this object.

See section 5.4 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 section 5.4 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 [10]: 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.

8.9 Interface Class IpPolicyAction

Inherits from: IpPolicy.

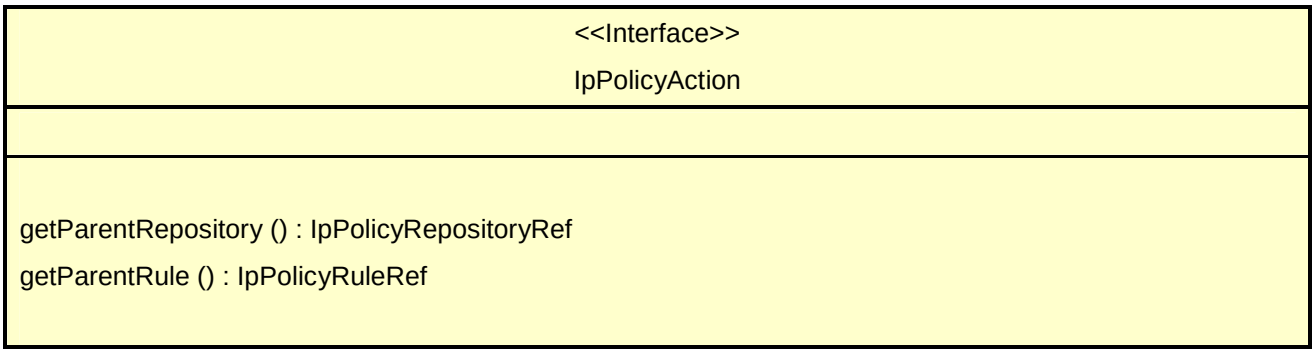
The purpose of a policy action is to execute one or more operations that will affect network traffic and/or systems, devices, etc., in order to achieve a desired state. This (new) state provides one or more (new) behaviours. A policy action ordinarily changes the configuration of one or more elements.

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.



8.9.1 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:

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

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

This 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 [PCIM].

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.

8.9.2 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

8.9.3 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

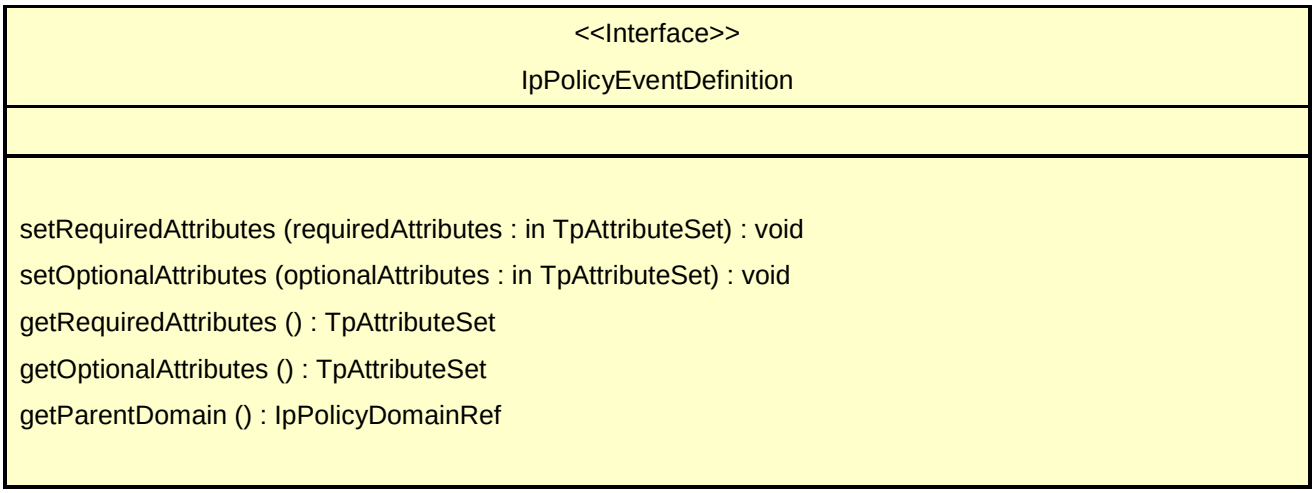
Raises

TpCommonExceptions

8.10 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.



8.10.1 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:

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

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

This 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 [PCIM].

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.

8.10.2 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

8.10.3 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

8.10.4 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

8.10.5 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`

8.10.6 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`

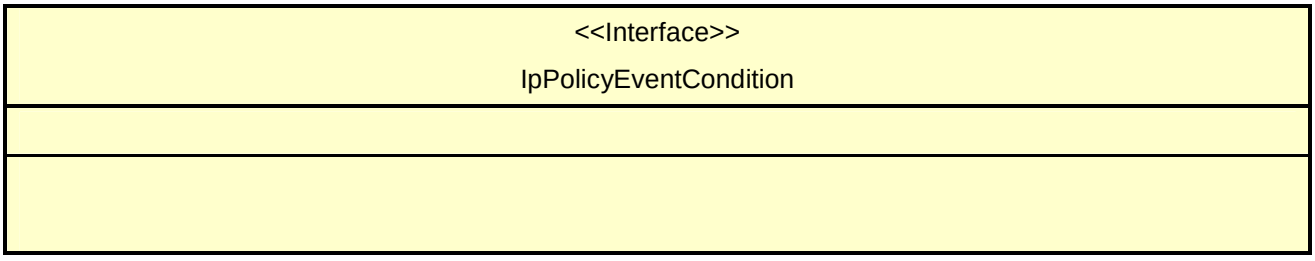
Raises

`TpCommonExceptions`

8.11 Interface Class IpPolicyEventCondition

Inherits from: IpPolicyCondition.

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



8.11.1 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:

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

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

This 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 [PCIM].

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.

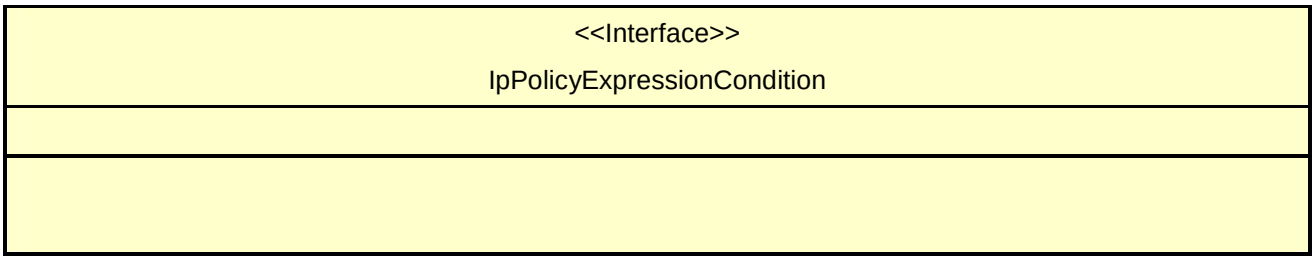
MatchingAttributes : TpAttributeSet

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.

8.12 Interface Class IpPolicyExpressionCondition

Inherits from: IpPolicyCondition.

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



8.12.1 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:

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

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

This 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 [PCIM].

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. The BNF describing the expression is defined as follows:

Expression:= VariableName <Comparison Operator> Constant or VariableName | VariableName <Arithmetic Operator> Constant or VariableName <Comparison Operator> Constant or VariableName | (VariableName<ArithmeticOperator>Constant or VariableName) <ArithmeticOperator> Constant or VariableName <Comparison Operator> Constant or VariableName

It is assumed that the Policy Engine is able to parse an expression defined in the above BNF. The BNF may be extended as appropriate.

Note that:

1. Variable is assumed to be one of type {TpInt32, TpFloat or TpString} and consistency of type is assumed when an expression is being defined.

2. Comparison Operator is one of: {==, !=, <=, >=}, and, Arithmetic Operator is one of {*, +, -, /}. These are reserved symbols. Note that when Variable is of type TpInt32 or TpFloat the Comparison and Arithmetic operators have the 'usual' meanings. When Variable is of type string, the comparison operators are the 'standard' string comparison operators. However, the only applicable Arithmetic operators are:

'*' := string concatention, e.g., abc*cde12 is the string abccde12

'-' := string (positional) difference, e.g., ABCD - ABCD is the null string but abcdef-abc is the string 'def'

'/' := string (positional) overlap, e.g., acbcd/acBCd is the string 'acd'

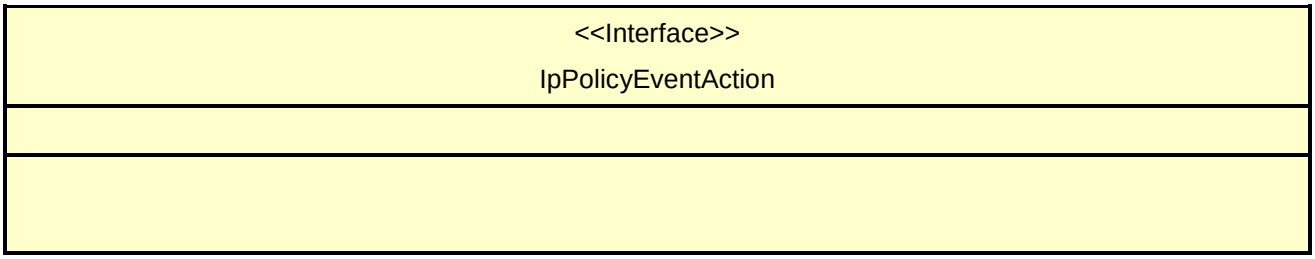
3. Example showing an expression formed using Variables of type TpFloat (or TpInt32): (bandwidth.allocated - bandwidth.used)/100 >= 36

Note that 'bandwidth' is assumed to be the name of a set of variables and 'allocated' & 'used' are variables (attributes) included in that set.

8.13 Interface Class IpPolicyEventAction

Inherits from: IpPolicyAction.

Generate an instance of a specified event.



8.13.1 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:

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

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

This 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 [PCIM].

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.

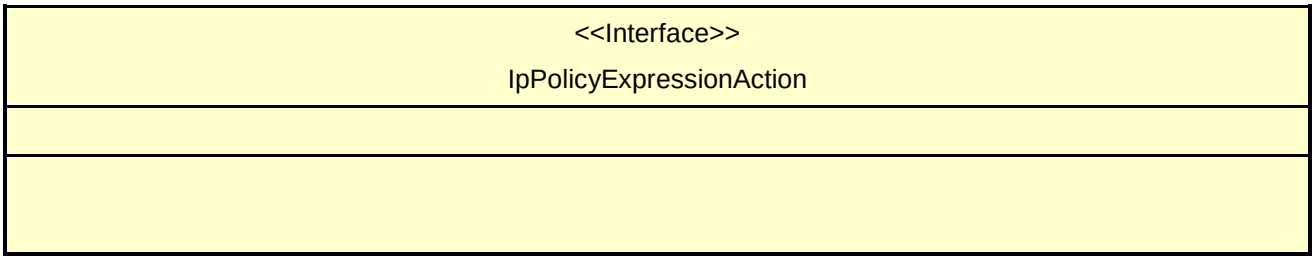
Attributes : TpAttributeSet

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.

8.14 Interface Class IpPolicyExpressionAction

Inherits from: IpPolicyAction.

Evaluate an expression.



8.14.1 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:

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

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

This 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 [PCIM].

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. The BNF describing the expression is defined as follows:

Expression:= VariableName<AssignmentOperator>Constant or VariableName<ArithmeticOperator> Constant or VariableName | VariableName<AssignmentOperator>Constant

It is assumed that the Policy Engine is able to parse an expression defined in the above BNF. The BNF may be extended as appropriate.

Note that:

1. Variable is assumed to be one of type {TpInt32, TpFloat or TpString} and consistency of type is assumed when an expression is being defined.

2. Assignment Operator is denoted by the symbol (within qoutes) '='. The assignment operator assigns the value of the 'right hand side' to the variable on the 'left hand side' -- see example below. Arithmetic Operator is one of {*, +, -, /}. All the above mentioned symbols are reserved symbols. Note that when Variable is of type TpInt32 or TpFloat the Arithmetic operators have the 'usual' meanings. When Variable is of type string the only applicable operators are the operators (within qoutes) '*' (concatenation), '-' (string difference) and '/' (string overlap).

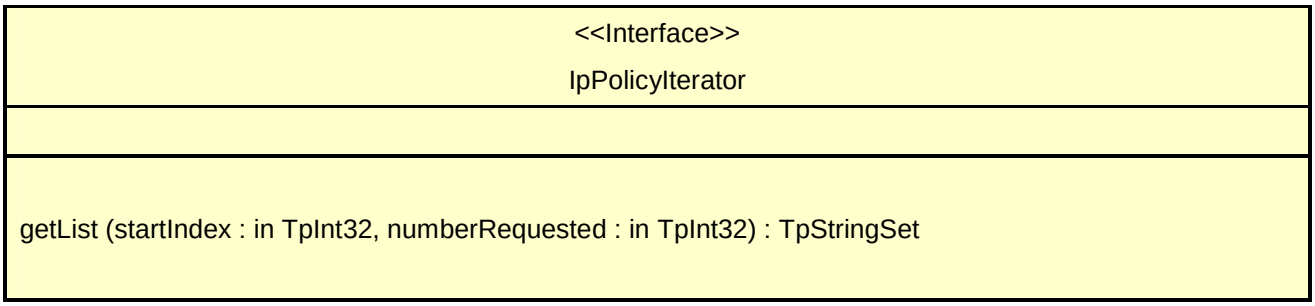
3. Example showing an assignment expression formed using Variables of type TpFloat (or TpInt32): content.charge = content.charge - 30

Note that 'content' is assumed to be the name of a set of variables and 'charge' is a variable (attribute) included in that set. In the above example, the value of content.charge is decremented by 30.

8.15 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.



8.15.1 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:

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

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

This 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 [PCIM].

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.

8.15.2 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`

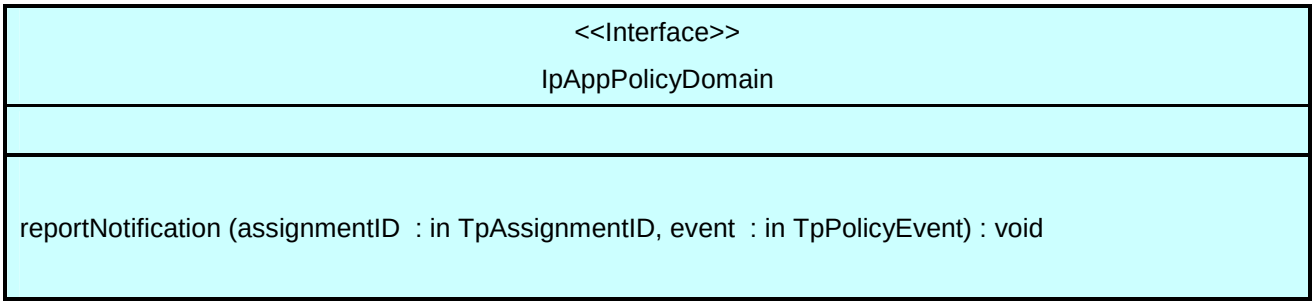
Raises

`TpCommonExceptions`

8.16 Interface Class IpAppPolicyDomain

Inherits from: IpInterface.

This interface is supported by the client. A reference to the interface is provided by the client by calling createNotification() on a given IpPolicyDomain. When notifications that the client has indicated interest in are available, they will be communicated to the client by calling the appropriate method on this interface.



8.16.1 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.

9 State Transition Diagrams

There are no State Transition Diagrams for the Policy Management SCF.

10 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.

10.1 Policy Management Data Definitions

This section 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.

10.1.1 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

10.1.2 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

10.1.3 TpPolicyConditionList

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

10.1.4 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

10.1.5 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

10.1.6 TpPolicyActionList

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

10.1.7 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

10.1.8 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

10.1.9 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.
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. 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.

10.1.10 TpPolicyKeywordSet

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

10.1.11 IpPolicyDomain

Defines the address of an `IpPolicyDomain` Interface.

10.1.12 IpPolicyDomainRef

Defines a `Reference` to an IpPolicyDomain

10.1.13 IpPolicyRepository

Defines the address of an `IpPolicyRepository` Interface.

10.1.14 IpPolicyRepositoryRef

Defines a `Reference` to an IpPolicyRepository

10.1.15 IpPolicyGroup

Defines the address of an `IpPolicyGroup` Interface.

10.1.16 IpPolicyGroupRef

Defines a `Reference` to an IpPolicyGroup

10.1.17 IpPolicyRule

Defines the address of an `IpPolicyRule` Interface.

10.1.18 IpPolicyRuleRef

Defines a `Reference` to an IpPolicyRule

10.1.19 IpPolicyEventDefinition

Defines the address of an `IpPolicyEventDefinition` Interface.

10.1.20 IpPolicyEventDefinitionRef

Defines a `Reference` to an IpPolicyEventDefinition

10.1.21 IpAppPolicyDomain

Defines the address of an `IpAppPolicyDomain` Interface.

10.1.22 IpAppPolicyDomainRef

Defines a `Reference` to an IpAppPolicyDomain

10.1.23 IpPolicyCondition

Defines the address of an `IpPolicyCondition` Interface.

10.1.24 IpPolicyConditionRef

Defines a `Reference` to an IpPolicyCondition

10.1.25 IpPolicyTimePeriodCondition

Defines the address of an `IpPolicyTimePeriodCondition` Interface.

10.1.26 IpPolicyTimePeriodConditionRef

Defines a `Reference` to an IpPolicyTimePeriodCondition

11 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 a text file (pm.idl contained in archive 2919813IDL.ZIP) which accompanies the present document.

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

The Java API realisation of this specification is produced in accordance with the Java Realisation rules defined in Part 1 of this specification series. 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 specification is provided as Java Code, contained in archive 2919813J2SE.ZIP that accompanies the present document.

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

Annex C (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
April 2002	--	--	--	--	Draft v100 submitted to TSG CN email list for Information	--	1.0.0
June 2002	CN_16	NP-020195	--	--	Draft v200 submitted to TSG CN#16 for Approval	2.0.0	5.0.0
Sep 2002	CN_17	NP-020439	001	--	Add text to clarify requirements on support of methods	5.0.0	5.1.0
Sep 2002	CN_17	NP-020395	002	--	Add text to clarify relationship between 3GPP and ETSI/Parlay OSA specifications	5.0.0	5.1.0
Sep 2003	CN_21	NP-030352	004	--	Correction to Java Realisation Annex	5.1.0	5.2.0

History

Document history		
V5.0.0	June 2002	Publication
V5.1.0	September 2002	Publication
V5.2.0	September 2003	Publication