**3GPP TSG CN Plenary Meeting #22** **NP-030548**
**10 - 12 December 2003, Maui, Hawaii, USA**

**Source:** **CN5 (OSA)**

**Title:** **Rel-5 CRs 29.198-02/13/14 OSA API**

**Agenda item:** **8.2**

**Document for:** **APPROVAL**

---

| Doc-1st-Level | Spec | CR | R | Ph | Subject | Cat | Version-Current | Doc-2nd-Lev | WI |
|---|---|---|---|---|---|---|---|---|---|
| NP-030548 | 29.198-02 | 042 | - | Rel-5 | Correction of datatypes supported by TpAttribute | F | 5.4.0 | N5-030643 | OSA2 |
| NP-030548 | 29.198-13 | 006 | - | Rel-5 | Correction of standard datatypes supported by TpPolicy - Align with 29.198-02 | F | 5.2.0 | N5-030648 | OSA2 |
| NP-030548 | 29.198-13 | 007 | - | Rel-6 | Correction of standard datatypes supported by TpPolicy - Align with 29.198-02 | A | 6.0.0 | N5-030647 | OSA2 |
| NP-030548 | 29.198-14 | 017 | - | Rel-5 | Correction of description of TpAttributeType to adequately support possible types - Align with 29.198-02 | F | 5.3.0 | N5-030645 | OSA2 |
| NP-030548 | 29.198-14 | 018 | - | Rel-5 | Correction of definitin of TpPAMAttribute and addition of Service Properties to publish supported attribute types - Align with 29.198-02 | F | 5.3.0 | N5-030646 | OSA2 |

*CR-Form-v7*

# CHANGE REQUEST

⌘ **29.198-02 CR 042** ⌘ **rev - ⌘** Current version: **5.4.0** ⌘

*For* ***HELP*** *on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

***Proposed change affects:*** UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network ☒

| | | | |
|---|---|---|---|
| ***Title:*** ⌘ | Correction of datatypes supported by TpAttribute | | |
| ***Source:*** ⌘ | CN5 (Telcordia, jlbakker@research.telcordia.com) | | |
| ***Work item code:*** ⌘ | OSA2 | ***Date:*** ⌘ | 19/11/2003 |
| ***Category:*** ⌘ | **F** | ***Release:*** ⌘ | *REL-5* |

*Use one of the following categories:*
- ***F*** *(correction)*
- ***A*** *(corresponds to a correction in an earlier release)*
- ***B*** *(addition of feature),*
- ***C*** *(functional modification of feature)*
- ***D*** *(editorial modification)*

Detailed explanations of the above categories can be found in 3GPP TR 21.900.

*Use one of the following releases:*

| | |
|---|---|
| *2* | *(GSM Phase 2)* |
| *R96* | *(Release 1996)* |
| *R97* | *(Release 1997)* |
| *R98* | *(Release 1998)* |
| *R99* | *(Release 1999)* |
| *Rel-4* | *(Release 4)* |
| *Rel-5* | *(Release 5)* |
| *Rel-6* | *(Release 6)* |

| | |
|---|---|
| ***Reason for change:*** ⌘ | Attribute typing system is not sufficient to ensure portable applications; vendors can define custom types such that different SCFs can not be integrated. |
| ***Summary of change:*** ⌘ | TpAttribute is extended with the CORBA standard primitive types, CORBA complex types, and an XML datatype, allowing any IDL or XML-expressable and verifiable datatype to be passed as an attribute. To ensure portability, custom types are scoped using namespaces or modules. |
| ***Consequences if not approved:*** ⌘ | SCFs will define proprietary extensions to support these data types, making any applications that use them vendor specific. |

| | | | | |
|---|---|---|---|---|
| ***Clauses affected:*** ⌘ | | | 5.1.12, 5.1.13, 5.1.20 - 5.1.34 | |
| | **Y** | **N** | | |
| ***Other specs affected:*** ⌘ | **X** | | Other core specifications ⌘ | Rel-5/6 29.198-13, 29.198-14 |
| | | **X** | Test specifications | |
| | | **X** | O&M Specifications | |
| ***Other comments:*** ⌘ | | | Children CRs against 29.198-13, 29.198-14 in N5-030645 to N5-030648 | |

## How to create CRs using this form:

Comprehensive information and tips about how to create CRs can be found at http://www.3gpp.org/specs/CR.htm.

### 5.1.12 TpAttribute

This is a `Sequence of Data Elements` containing the attribute name~~, type,~~ and value.~~ The attribute Value is interpreted based on the value of the attribute Type.~~

| Sequence Element Name | Sequence Element Type | Notes |
|---|---|---|
| ~~AttributeName~~ | ~~TpString~~ | ~~The name of the attribute.~~ |
| ~~AttributeType~~ | ~~TpAttributeType~~ | ~~The type of the attribute. Valid values for Type must include at least TpString, TpInt32 and TpFloat.~~ |
| ~~AttributeValue~~ | ~~TpAny~~ | ~~The values for the attribute. This model allows multi-valued attributes. Cannot be an empty list.~~ |
| AttributeName | TpString | The name of the attribute. |
| AttributeValue | TpAttributeValue | The typed value(s) for the attribute. |

### 5.1.13 ~~TpAttributeType~~TpAttributeValue

This is a tagged choice of data elements to hold attribute values of different complexity.

|  | Tag Element Type |  |
|---|---|---|
|  | TpAttributeTagInfo |  |

| Tag Element Value | Choice Element Type | Choice Element Name |
|---|---|---|
| P_SIMPLE_TYPE | TpSimpleAttributeValue | SimpleValue |
| P_STRUCTURED_TYPE | TpStructuredAttributeValue | StructuredValue |
| P_XML_TYPE | TpXMLString | XMLValue |

~~This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the type of an attribute. Other Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The following values are defined.~~

| ~~Character String Value~~ | ~~Description~~ |
|---|---|
| ~~*NULL*~~ | ~~An empty (NULL) string indicates no attribute type~~ |
| ~~P_STRING~~ | ~~Attribute type is type TpString.~~ |
| ~~P_INT32~~ | ~~Attribute type is type TpInt32.~~ |
| ~~P_FLOAT~~ | ~~Attribute type is type TpFloat.~~ |

### 5.1.20 TpAttributeTagInfo

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

| Name | Value | Description |
|---|---|---|
| P_SIMPLE_TYPE | 0 | Simple type |
| P_STRUCTURED_TYPE | 1 | Structured type |
| P_XML_TYPE | 2 | XML type |

## 5.1.21 TpSimpleAttributeValue

This is a tagged choice of data elements to hold attribute values of different complexity.

| | Tag Element Type | |
|---|---|---|
| | TpSimpleAttributeTypeInfo | |

| TAG_ELEMENT_VALUE | Choice Element Type | Choice Element Name |
|---|---|---|
| P_BOOLEAN | TpBoolean | BooleanValue |
| P_OCTET | TpOctet | OctetValue |
| P_CHAR | TpChar | CharValue |
| P_WCHAR | TpWChar | WCharValue |
| P_STRING | TpString | StringValue |
| P_WSTRING | TpWString | WStringValue |
| P_INT16 | TpInt16 | Int16Value |
| P_UNSIGNED_INT16 | TpUnsignedInt16 | UnsignedInt16Value |
| P_INT32 | TpInt32 | Int32Value |
| P_UNSIGNED_INT32 | TpUnsignedInt32 | UnsignedInt32Value |
| P_INT64 | TpInt64 | Int64Value |
| P_UNSIGNED_INT64 | TpUnsignedInt64 | UnsignedInt64Value |
| P_FLOAT | TpFloat | FloatValue |
| P_DOUBLE | TpDouble | DoubleValue |
| P_FIXED | TpFixed | FixedValue |

## 5.1.22 TpSimpleAttributeTypeInfo

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

| Name | Value | Description |
|---|---|---|
| P_BOOLEAN | 0 | Attribute type is type TpBoolean. |
| P_OCTET | 1 | Attribute type is type TpOctet. |
| P_CHAR | 2 | Attribute type is type TpChar. |
| P_WCHAR | 3 | Attribute type is type TpWChar. |
| P_STRING | 4 | Attribute type is type TpString. |
| P_WSTRING | 5 | Attribute type is type TpWString. |
| P_INT16 | 6 | Attribute type is type TpInt16. |
| P_UNSIGNED_INT16 | 7 | Attribute type is type TpUnsignedInt16. |
| P_INT32 | 8 | Attribute type is type TpInt32. |
| P_UNSIGNED_INT32 | 9 | Attribute type is type TpUnsignedInt32. |
| P_INT64 | 10 | Attribute type is type TpInt64. |
| P_UNSIGNED_INT64 | 11 | Attribute type is type TpUnsignedInt64. |
| P_FLOAT | 12 | Attribute type is type TpFloat. |
| P_DOUBLE | 13 | Attribute type is type TpDouble. |
| P_FIXED | 14 | Attribute type is type TpFixed. |

### 5.1.23 TpStructuredAttributeType

This data type is identical to a TpString, and is defined as a string of characters that uniquely identifies the type of a structured data type. Network operator specific capabilities may also be used, but should be preceded by the string "SP_". The pattern of values is defined, where module names and class names map to a fully specified class name.

| Character String Value | Description |
|---|---|
| `P_module1/module2/module3/className` | An object of the specified, fully qualified class. |

### 5.1.24 TpStructuredAttributeValue

This is a `Sequence of Data Elements` containing the structured attribute type tag and the value to be interpreted using that type.

| Sequence Element Name | Sequence Element Type | Notes |
|---|---|---|
| `Type` | `TpStructuredAttributeType` | The type for the value. |
| `Value` | `TpAny` | The structured values for the attribute. |

### 5.1.25 TpChar

This type is an 8-bit quantity that may undergo conversion when transmitted by the communication system.

### 5.1.26 TpWChar

This type is a quantity that may undergo conversion when transmitted by the communication system. The size of this type is implementation-dependent.

### 5.1.27 TpWString

Defines a TpWChar string, comprising length and data. The length shall be at least a 16-bit integer.

### 5.1.28 TpInt16

Defines a signed 16-bit integer.

### 5.1.29 TpUnsignedInt16

Defines an unsigned 16-bit integer.

### 5.1.30 TpUnsignedInt32

Defines an unsigned 32-bit integer.

### 5.1.31 TpUnsignedInt64

Defines an unsigned 64-bit integer.

### 5.1.32 TpDouble

Defines a double precision real number.

### 5.1.33 TpFixed

This data type defines a fixed-point decimal number of up to 31 significant digits. The scale factor is a non-negative integer less than or equal to the total number of digits.

### 5.1.34 TpXMLString

This data type is TpString containing well-formed XML and may contain a reference to/include a DTD or Schema.

# Annex E (informative): Change history

| Change history | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Date** | **TSG #** | **TSG Doc.** | **CR** | **Rev** | **Subject/Comment** | **Old** | **New** |
| Mar 2001 | CN_11 | NP-010134 | 047 | -- | CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158) | 3.2.0 | 4.0.0 |
| Jun 2001 | CN_12 | NP-010330 | 001 | -- | Corrections to OSA API Rel4 (Exception handling mechanism without ambiguity - Replace TpGeneralException and TpResultInfo with detailed exception classes which can be thrown for each method (N5-010261) | 4.0.0 | 4.1.0 |
| Jun 2001 | CN_12 | NP-010333 | 002 | -- | Introduction of TpOctet (In order to make sure that some data is sent over the "distributed wire" untouched a new data type is needed) (N5-010304) | 4.0.0 | 4.1.0 |
| Sep 2001 | CN_13 | NP-010465 | 003 | -- | Changing references to JAIN | 4.1.0 | 4.2.0 |
| Sep 2001 | CN_13 | NP-010465 | 004 | -- | Clarification of common exceptions | 4.1.0 | 4.2.0 |
| Sep 2001 | CN_13 | NP-010465 | 005 | -- | Invalid parameter value exception for SLA violation | 4.1.0 | 4.2.0 |
| Sep 2001 | CN_13 | NP-010465 | 006 | -- | Storing eventCriteria | 4.1.0 | 4.2.0 |
| Dec 2001 | CN_14 | NP-010595 | 007 | -- | Replace Out Parameters with Return Types | 4.2.0 | 4.3.0 |
| Dec 2001 | CN_14 | NP-010595 | 008 | -- | Correction to Common Data (CD) | 4.2.0 | 4.3.0 |
| Dec 2001 | CN_14 | NP-010595 | 009 | -- | Correction to values of TpAddressPlan | 4.2.0 | 4.3.0 |
| Mar 2002 | CN_15 | NP-020104 | 010 | -- | Ambiguous definition of TpAssignmentID | 4.3.0 | 4.4.0 |
| Mar 2002 | CN_15 | NP-020104 | 011 | -- | Data type alignment in the common data types | 4.3.0 | 4.4.0 |
| Jun 2002 | CN_16 | NP-020185 | 011 | -- | Allowing the use of tel URL in TpAddressPlan | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 012 | -- | Adding TpInt64 in order to aling with the new Rel-5 TS 29.198-14 | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 013 | -- | Addition of undefined Data types: TpStringList and TpStringSet | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020181 | 014 | -- | Addition of support for Java API technology realisation | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020182 | 015 | -- | Addition of support for WSDL realisation | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 016 | -- | Deletion of P_SET_LENGTH_EXCEEDED | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 017 | -- | Removal of MIDL | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 018 | -- | Revise the scope of TpSessionID and TpAssignmentID | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 019 | -- | Deprecate P_ADDRESS_PLAN_MSMAIL | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 020 | -- | Addition of support for an Exception Hierarchy | 4.4.0 | 5.0.0 |
| Jun 2002 | CN_16 | NP-020185 | 021 | -- | Addition of type TpVersion in common data | 4.4.0 | 5.0.0 |
| Sep 2002 | CN_17 | NP-020395 | 022 | -- | Add text to clarify relationship between 3GPP and ETSI/Parlay OSA specifications | 5.0.0 | 5.1.0 |
| Oct 2002 | -- | -- | -- | -- | Added the two missing attachments<br>(osa.idl contained in archive 2919802IDL.ZIP)<br>(osa.wsdl contained in archive 2919802WSDL.ZIP) | 5.1.0 | 5.1.1 |
| Mar 2003 | CN_19 | NP-030018 | 025 | -- | Clarification on uniqueness of assignmentID | 5.1.1 | 5.2.0 |
| Mar 2003 | CN_19 | NP-030018 | 027 | -- | Correction to P_INVALID_STATE value | 5.1.1 | 5.2.0 |
| Mar 2003 | CN_19 | NP-030018 | 029 | -- | Addition of Support of National Numbering Plans | 5.1.1 | 5.2.0 |
| Mar 2003 | CN_19 | NP-030027 | 030 | -- | Addition of Numbered List of Data Elements definition | 5.1.1 | 5.2.0 |
| Mar 2003 | CN_19 | NP-030027 | 031 | -- | Correction of Exception Hierarchy to align with Java Realisation | 5.1.1 | 5.2.0 |
| Mar 2003 | CN_19 | NP-030027 | 032 | -- | Promotion of TpDataSessionQosClass dat type definition to the Common Data Types | 5.1.1 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030236 | 034 | -- | Correction of SIP Address wildcard rules | 5.2.0 | 5.3.0 |
| Jun 2003 | CN_20 | NP-030240 | 035 | -- | Add the type TpURN to Common Data Types | 5.2.0 | 5.3.0 |
| Sep 2003 | CN_21 | NP-030352 | 036 | -- | Correction to Java Realisation Annex | 5.3.0 | 5.4.0 |
| | | | | | | | |

CR-Form-v7

# CHANGE REQUEST

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ⌘ | **29.198-13** | **CR** | **007** | ⌘ **rev** | **-** | ⌘ | Current version: | **6.0.0** | ⌘ |

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

| | | | | |
|---|---|---|---|---|
| ***Proposed change affects:*** | UICC apps⌘ [ ] | ME [ ] | Radio Access Network [ ] | Core Network [X] |

| | | | | |
|---|---|---|---|---|
| ***Title:*** | ⌘ | Correction of standard datatypes supported by TpPolicy - Align with 29.198-02 | | |
| ***Source:*** | ⌘ | CN5 (Telcordia, jlbakker@research.telcordia.com; Lucent, squtub@lucent.com) | | |
| ***Work item code:*** | ⌘ | OSA2 | ***Date:*** ⌘ | 19/11/2003 |
| ***Category:*** | ⌘ | **A** | ***Release:*** ⌘ | *REL-6* |
| | | *Use one of the following categories:*<br>***F*** *(correction)*<br>***A*** *(corresponds to a correction in an earlier release)*<br>***B*** *(addition of feature),*<br>***C*** *(functional modification of feature)*<br>***D*** *(editorial modification)*<br>Detailed explanations of the above categories can be found in 3GPP TR 21.900. | | *Use one of the following releases:*<br>*2 (GSM Phase 2)*<br>*R96 (Release 1996)*<br>*R97 (Release 1997)*<br>*R98 (Release 1998)*<br>*R99 (Release 1999)*<br>*Rel-4 (Release 4)*<br>*Rel-5 (Release 5)*<br>*Rel-6 (Release 6)* |

| | | |
|---|---|---|
| ***Reason for change:*** | ⌘ | TpPolicyAtomicType is a copy of TpAttributeType, adding P_BOOLEAN. A companion CR adds P_BOOLEAN to TpAttributeType. In order to prevent increasing the number of types in OSA common types have been defined. This CR proposes to use common type TpAttributeType rather than a custom copy for reasons of clarity to application developers, flexibility and ease of maintenance. Additionally, the type TpPolicyAtomicType does not allow customization through the SP_ rule. Hence, the current definition of TpPolicyAtomicType was found to restrictive and to implementation specific. |
| ***Summary of change:*** | ⌘ | TpAttributeType is extended with the CORBA standard primitive types, CORBA complex types, and an XML datatype, allowing any IDL or XML-expressable and verifiable datatype to be passed, including Boolean, Digit and Date. TpPolicyAtomicType only allows 4 types. |
| ***Consequences if not approved:*** | ⌘ | Limited applicability of the Policy Management API; Policy Management API cannot manage, e.g., currency amount based policies such that such policies are portable. Policy typing system not rigourously defined. |

| | | | | | | |
|---|---|---|---|---|---|---|
| ***Clauses affected:*** | ⌘ | 5.2-3, 5.5, 8.1.12.1, 8.1.14.1, 10, 11.2.3, 11.2.6-7, 11.3.1-2, 11.3.6, 11.4-5 | | | | |
| | | **Y** | **N** | | | |
| ***Other specs affected:*** | ⌘ | | **X** | Other core specifications | ⌘ | |
| | | | **X** | Test specifications | | |
| | | | **X** | O&M Specifications | | |
| ***Other comments:*** | ⌘ | Rel-6 Mirror CR of N5-030648 | | | | |

**How to create CRs using this form:**
Comprehensive information and tips about how to create CRs can be found at http://www.3gpp.org/specs/CR.htm.

## 5.2 Introduce condition and action into rule

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

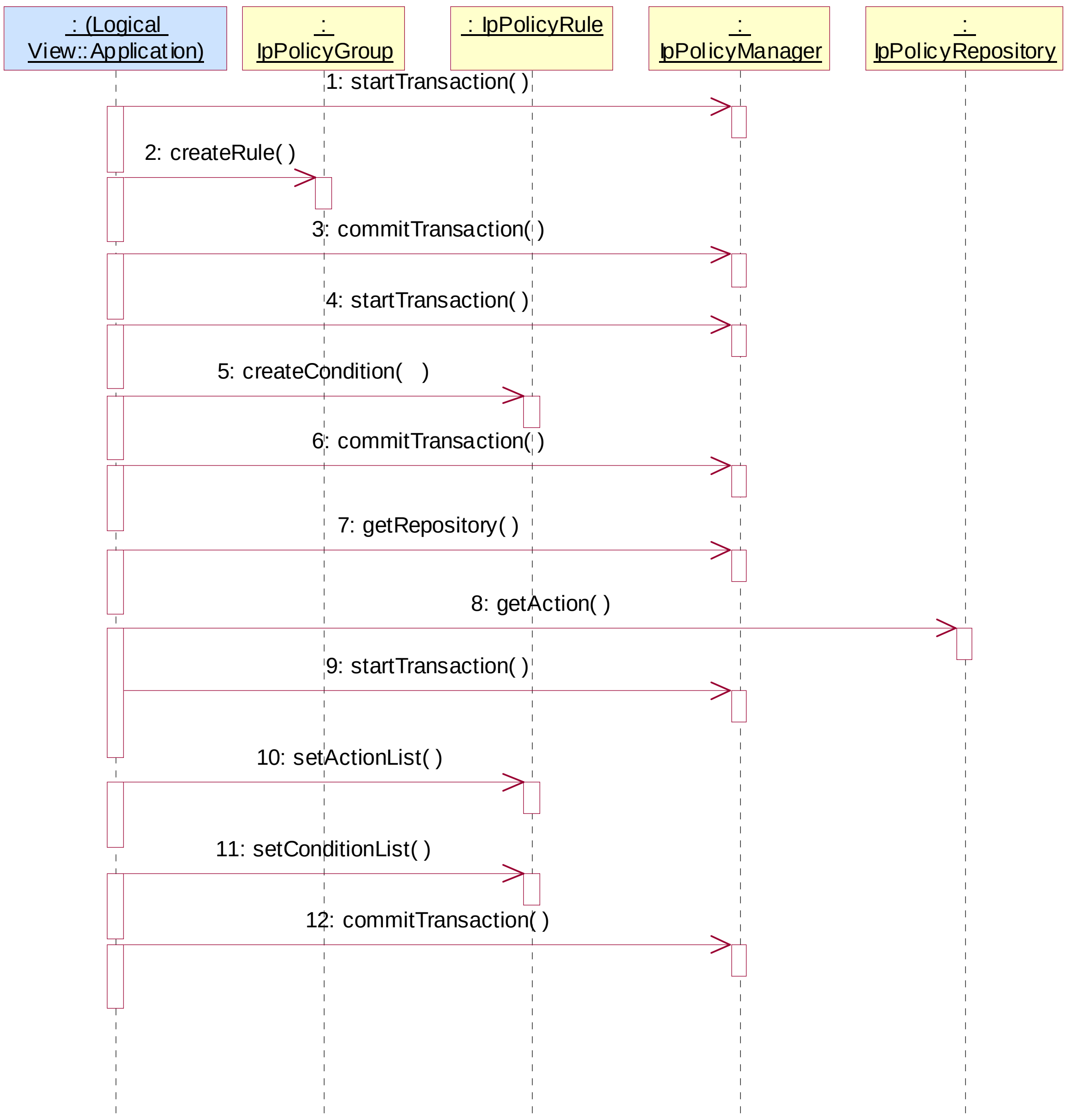
This sequence includes:

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

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

- establishing a transaction bracket.

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



1: Opens the transaction bracket.

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

3: Closes the transaction bracket.

4: Opens the transaction bracket.

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

Parameters passed are the condition name and the condition type.

Returns a reference to this condition object.

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

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

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

~~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~~ IpPolicyRule object created with message 2.

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

Activation of actions is done similarly.

12: Closes the transaction bracket.

## 5.3 Create event

This sequence shows how policy events are used.

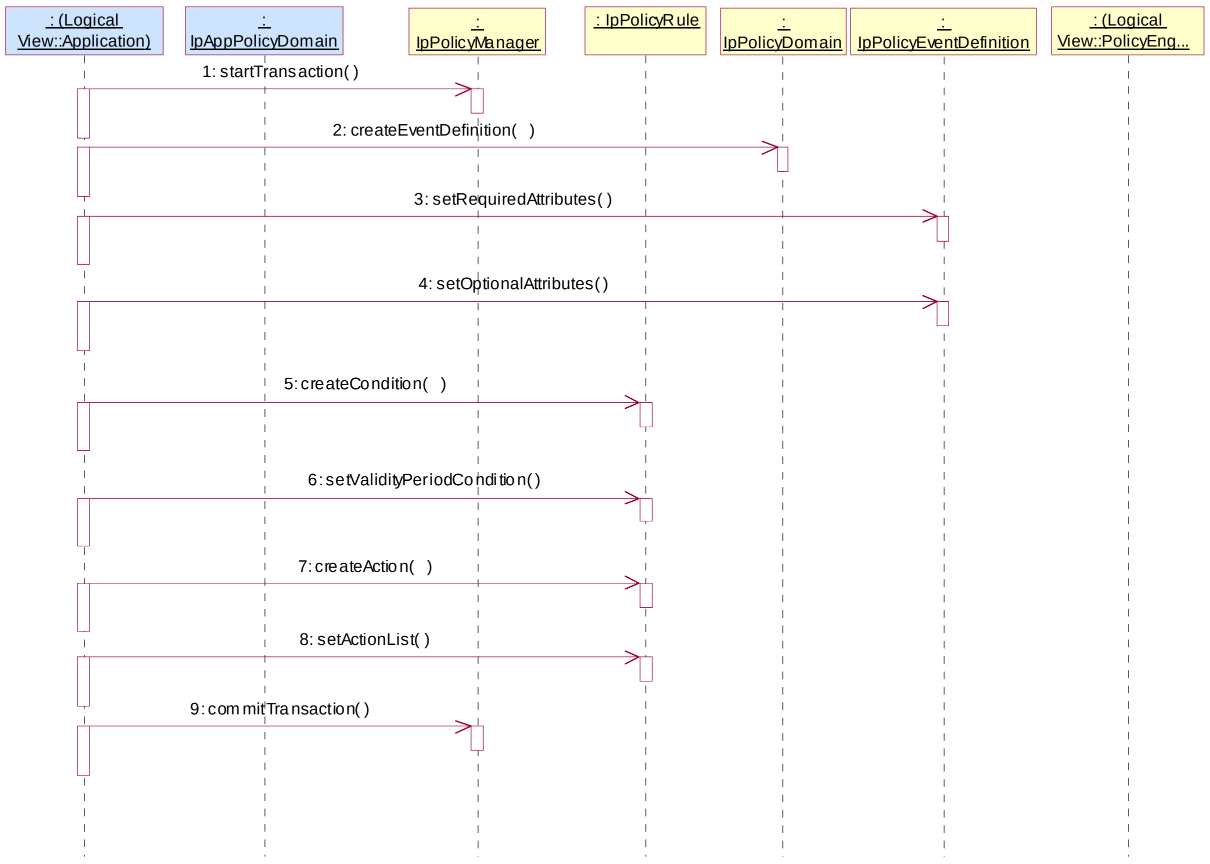
For clarification we list the different policy related objects used:

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

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

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

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


: (Logical View::Application)
: IpAppPolicyDomain
: IpPolicyManager
: IpPolicyRule
: IpPolicyDomain
: IpPolicyEventDefinition
: (Logical View::PolicyEng...
1: startTransaction( )
2: createEventDefinition( )
3: setRequiredAttributes( )
4: setOptionalAttributes( )
5: createCondition( )
6: setValidityPeriodCondition( )
7: createAction( )
8: setActionList( )
9: commitTransaction( )

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

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

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

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

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

Returns a reference to the new instance of IpPolicyTimePeriodCondition object.

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

TpAttribute.AttributeName = "TimePeriod"

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

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

~~5: 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().~~

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

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

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

Returns the reference to the newly created action object.

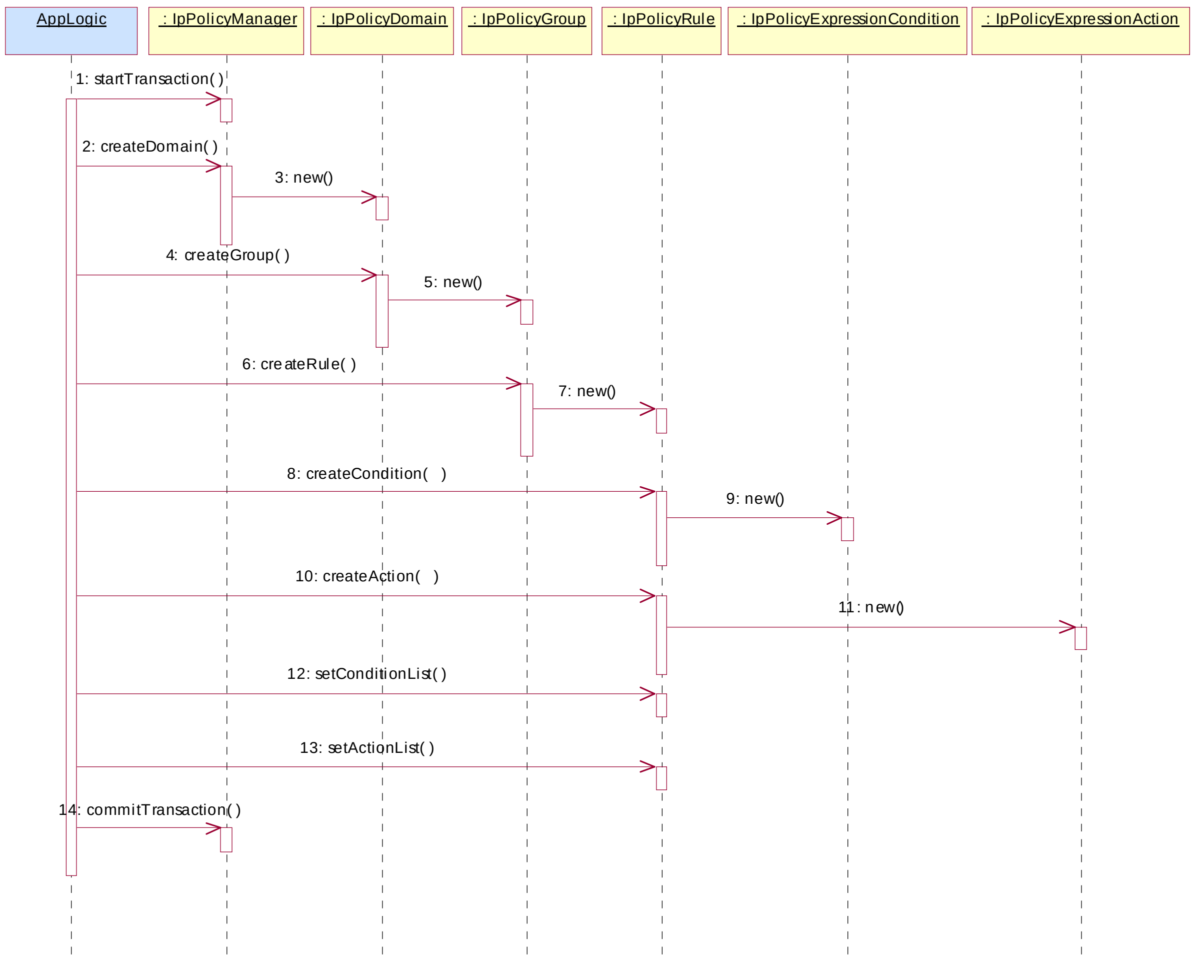
8: This method activates the action (here the action event) for this rule. After creation this action is not yet active.

The name of the action object is passed.

9: This closes the transaction bracket.

## 5.5 ASP offering services to prepaid subscribers

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



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

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

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

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

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

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

- Enabled == TRUE; the policy rule is currently enabled.
- RuleUsage == NULL; no free-format usage recommendation is provided.
- Priority == 0; default value, as there is only one rule.
- Mandatory == TRUE; mandatory rule, evaluation of the expression must be attempted

- PolicyRoles == PrePaidBalanceCheck. Each rule must be assigned a policy role(s).
- ConditionListType == P_PM_DNF; disjunctive normal form (DNF)
- SequencedActions == 3; do not care, as there is only one rule.

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

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

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

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

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

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

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

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

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

- conditionList.Condition == <reference to the IpPolicyCondition interface returned by 9>
- conditionList.GroupNumber == 1; indicates how the conditions need to be grouped in DNF or CNF in case more groups of rules exist.
- conditionList.Negated == FALSE.

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

- actionList.Action == <reference to the IpPolicyAction interface returned by step 10>
- actionList.SequenceNumber == 1;

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

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

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

```
+-----------------------------------------------------------+
|PolicyDomain "eCommerceDomain"                             |
|                                                           |
|    +---------------------------------------------------+  |
|    |PolicyGroup "PrePaidGroup"                          |    |
|    |                                                    |    |
|    | +----------------------------------------------+   |    |
|    | |PolicyRule "SufficientCreditRule"             |   |    |
|    | |                                              |   |    |
|    | | +--------------------+  +--------------------+ |   |    |
|    | | |PolicyCondition     |  |PolicyAction        | |   |    |
|    | | | "SufficientCredit"|  | "PurchaseAllowed" | |   |    |
|    | | +--------------------+  +--------------------+ |   |    |
|    | +----------------------------------------------+   |    |
|    +----------------------------------------------------+    |
+--------------------------------------------------------------+
```

### 8.1.12.1 Attributes

**`CommonName : TpString`**

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

**`PolicyKeywords : TpStringSet`**

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

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

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

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

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

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

**`Caption : TpString`**

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

**`Description : TpString`**

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

**`Expression : TpString`**

The expression to be evaluated as the condition.  In case this SCF supports both eBNF and XML, then the TpAttributeTagInfo of the TpAttribute that populated this expression is used to distinguish between XML and eBNF string contents.  A TpAttributeTagInfo value of P_XML_TYPE indicates XML as contents of the Expression attribute and a TpAttributeTagInfo value of `P_SIMPLE_TYPE` indicates eBNF as contents of Expression attribute.  The eBNF~~The eBNF describing the expression is defined in clause 10.3.~~ definition can be found in Section 11.3.

#### 8.1.14.1 Attributes

**`CommonName : TpString`**

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

**`PolicyKeywords : TpStringSet`**

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

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

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

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

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

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

**`Caption : TpString`**

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

**`Description : TpString`**

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

**`Expression : TpString`**

The expression that should evaluated. In case this SCF supports both eBNF and XML, then the TpAttributeTagInfo of the TpAttribute that populated this expression is used to distinguish between XML and eBNF string contents. A TpAttributeTagInfo value of P_XML_TYPE indicates XML as contents of the Expression attribute and a TpAttributeTagInfo value of `P_SIMPLE_TYPE` indicates `eBNF` as contents of Expression attribute. The eBNF definition can be found in Section 11.3. ~~The BNF describing the expression is defined in clause 10.3.~~

# 10 PM Service Properties

The following table lists properties relevant to all the PM SCFs

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

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

```
P_SUPPORTED_ATTRIBUTE_TAGS = {
P_SIMPLE_TYPE
}
P_SUPPORTED_SIMPLE_ATTRIBUTE_TYPES = {
P_STRING,
P_FLOAT,
P_INT32,
P_BOOLEAN
}
P_SUPPORTED_VARIABLE_TAGS = {
P_SIMPLE_TYPE
P_PM_TYPE_RECORD,
P_PM_TYPE_LIST
}
P_SUPPORTED_SIMPLE_VARIABLE_TYPES = {
P_STRING,
P_FLOAT,
P_INT32,
P_BOOLEAN
}
```

### ~~11.2.3TpPolicyAtomicType~~

~~TpPolicyAtomicType defines a set of data elements of type string, that can contain the following values:~~

| ~~Character String Value~~ | ~~Description~~ |
|---|---|
| ~~*NULL*~~ | ~~An empty (NULL) string indicates no variable type~~ |
| ~~P_STRING~~ | ~~Variable type is type TpString.~~ |
| ~~P_INT32~~ | ~~Variable type is type TpInt32.~~ |
| ~~P_FLOAT~~ | ~~Variable type is type TpFloat.~~ |
| ~~P_BOOLEAN~~ | ~~Variable type is type TpBoolean.~~ |

### ~~10.2.6~~11.2.6 TpPolicyTypeInfo

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

| Name | Value | Description |
|---|---|---|
| P_SIMPLE_TYPE | 0 | Simple type |
| ~~P_PM_TYPE_ATOMIC~~ | ~~0~~ | ~~Atomic type~~ |
| P_PM_TYPE_RECORD | 1 | Record type |
| P_PM_TYPE_LIST | 2 | List type |
| P_STRUCTURED_TYPE | 3 | Structured type |
| P_XML_TYPE | 4 | XML type |

### ~~10.2.7~~11.2.7 TpPolicyType

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

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

| Tag Element Value | Choice Element Type | Choice Element Name |
|---|---|---|
| P_SIMPLE_TYPE~~P_PM_TYPE_ATOMIC~~ | TpSimpleAttributeType~~TpPolicyAtomicType~~ | SimpleType~~AtomicType~~ |
| P_PM_TYPE_RECORD | TpPolicyRecordType | RecordType |
| P_PM_TYPE_LIST | TpPolicyListType | ListType |
| P_STRUCTURED_TYPE | TpStructuredAttributeType | StructuredType |
| P_XML_TYPE | TpXMLString | XMLString |

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

The choice elements represent the following:

| | |
|---|---|
| SimpleType: | Defines an atomic type. |
| ~~AtomicType:~~ | ~~Defines an atomic type. Note that we do not want to use TpAttributeType here for the sake of differentiating between interface attributes and variable types.~~ |
| RecordType: | Defines a record type with named fields. |
| ListType: | Defines a homogeneous list type. Heterogeneous lists are not supported. |
| StructuredType | Defines an object of the specified, fully qualified class |
| XMLString | Defines a data type that contains well-formed XML. |

### ~~10.3.4~~11.3.1 Basic Definition

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

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

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

Note 2: The variable name syntax must conform to the eBNF specified by the identifier non-ter~~i~~minal above.

## ~~10.3.5~~11.3.2 Definitions of Constant (Literals)

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

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

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

~~Note: int_const must be an integer in the range $-2^{31}$ through $2^{31}$-1. float_const must be a float as defined in Section 3.10.1.2 of the CORBA 2.4.2 Architecture and Specification document dated Feb 2001.~~

```
number::=
      int_const
    | float_const
    ;

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

## ~~10.3.9~~11.3.6 Allowable Condition and Action Expressions

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

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

condition::= predicate;

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

Examples of action expressions include:

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

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

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

## ~~10.4~~11.4 Example Scenarios

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

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

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

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

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

1) Provision variables:

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

        // start transaction
        manager.startTransaction();

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

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

        // define the type of x
        // note that we can use the int_type defined as part of this
        // process, for the type of y as well
        TpPolicyType int_type = TpPolicyType(TpSimpleAttributeTypeInfo(P_INT32)~~TpPolicyAtomicType(P_INT32)~~);
        TpPolicyType float_type =
TpPolicyType(TpSimpleAttributeTypeInfo(P_FLOAT)~~TpPolicyAtomicType(P_ FLOAT)~~);
        Vector<TpString> field_names = [“a”, “b”];
        Vector<TpPolicyType> field_types = [int_type, float_type];
        TpPolicyType x_type = TpPolicyType(TpRecordType(field_names,field_types));

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


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

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

2) Create signature:

```
        IpPolicySignatureRef sig = domain.createSignature(“test_sig”);

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

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

3) Provision the rules:

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

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

4) Sending a decision request:

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

- Case 1:

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

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

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

Note that if the value of x was specified as:

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

The rule condition would not be true, which implies that the rule action would not be executed. However, the signature “sig_test” specified that y was an output variable and hence its value was to be sent back to the client. However (as mentioned earlier in our assumptions about variable semantics), y started out as being uninitialized, and hence an exception would be returned back to the client.

- Case 2:

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

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

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

## 11.5 Example XML Scenarios

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

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

```
    end
```

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

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

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

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

Condition (to be passed in an ActionAttribute.AttributeValue)

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

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

5) Provision variables:

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

          // start transaction
          manager.startTransaction();

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

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

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

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

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

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

6) Create signature:

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

          // set input and output variables
          TpStringSet input_vars = ["SIPAddress"];
```

```
        TpStringSet output_vars = ["setCallLegProperty"];
        sig.setInputVariables(input_vars);
        sig.setOutputVariables(output_vars);

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

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

# Annex B (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 |
| Jun 2003 | CN_20 | NP-030250 | 003 | -- | New Policy Evaluation SCF introduced | 5.1.0 | 6.0.0 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

**joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) N5-030648**
**Meeting #25, Bangkok, THAILAND, 27 - 31 October 2003**

*CR-Form-v7*

# CHANGE REQUEST

| | | | | | | |
|---|---|---|---|---|---|---|
| ⌘ **29.198-13** | **CR** | **006** | ⌘ **rev** | **-** | ⌘ Current version: | **5.2.0** ⌘ |

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

***Proposed change affects:*** UICC apps⌘ [ ] ME [ ] Radio Access Network [ ] Core Network [X]

| | | | | |
|---|---|---|---|---|
| ***Title:*** | ⌘ | Correction of standard datatypes supported by TpPolicy - Align with 29.198-02 | | |
| ***Source:*** | ⌘ | CN5 (Telcordia, jlbakker@research.telcordia.com; Lucent, squtub@lucent.com) | | |
| ***Work item code:*** | ⌘ | OSA2 | ***Date:*** ⌘ | 19/11/2003 |
| ***Category:*** | ⌘ | **F** | ***Release:*** ⌘ | *REL-5* |

*Use one of the following categories:*
- ***F*** *(correction)*
- ***A*** *(corresponds to a correction in an earlier release)*
- ***B*** *(addition of feature),*
- ***C*** *(functional modification of feature)*
- ***D*** *(editorial modification)*

Detailed explanations of the above categories can be found in 3GPP TR 21.900.

*Use one of the following releases:*

| | |
|---|---|
| *2* | *(GSM Phase 2)* |
| *R96* | *(Release 1996)* |
| *R97* | *(Release 1997)* |
| *R98* | *(Release 1998)* |
| *R99* | *(Release 1999)* |
| *Rel-4* | *(Release 4)* |
| *Rel-5* | *(Release 5)* |
| *Rel-6* | *(Release 6)* |

| | | |
|---|---|---|
| ***Reason for change:*** | ⌘ | In order to prevent increasing the number of types in OSA common types have been defined. A companion CR corrects the list types specified by TpAttributeType. This CR proposes to use common type TpAttributeType rather than a custom copy for reasons of clarity to application developers, flexibility and ease of maintenance. |
| ***Summary of change:*** | ⌘ | TpAttributeType is extended with the CORBA standard primitive types, CORBA complex types, and an XML datatype, allowing any IDL or XML-expressable and verifiable datatype to be passed, including Boolean, Digit and Date. The original TpAttributeType only allows 3 types. |
| ***Consequences if not approved:*** | ⌘ | Limited applicability of the Policy Management API; Policy Management API cannot manage, e.g., currency amount based policies such that such policies are portable. Policy typing system not rigourously defined. |

| | | | | | | |
|---|---|---|---|---|---|---|
| ***Clauses affected:*** | ⌘ | 5.2-3, 5.5, 8.12.1, 8.14.1, 10 | | | | |
| | | **Y** | **N** | | | |
| ***Other specs affected:*** | ⌘ | **X** | | Other core specifications | ⌘ | Rel-6 29.198-13 |
| | | | **X** | Test specifications | | |
| | | | **X** | O&M Specifications | | |
| ***Other comments:*** | ⌘ | Child CR to Rel-5 29.198-02 CR in N5-030643.<br><br>Rel-6 Mirror CR in N5-030647. | | | | |

**How to create CRs using this form:**
Comprehensive information and tips about how to create CRs can be found at http://www.3gpp.org/specs/CR.htm.

## 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.

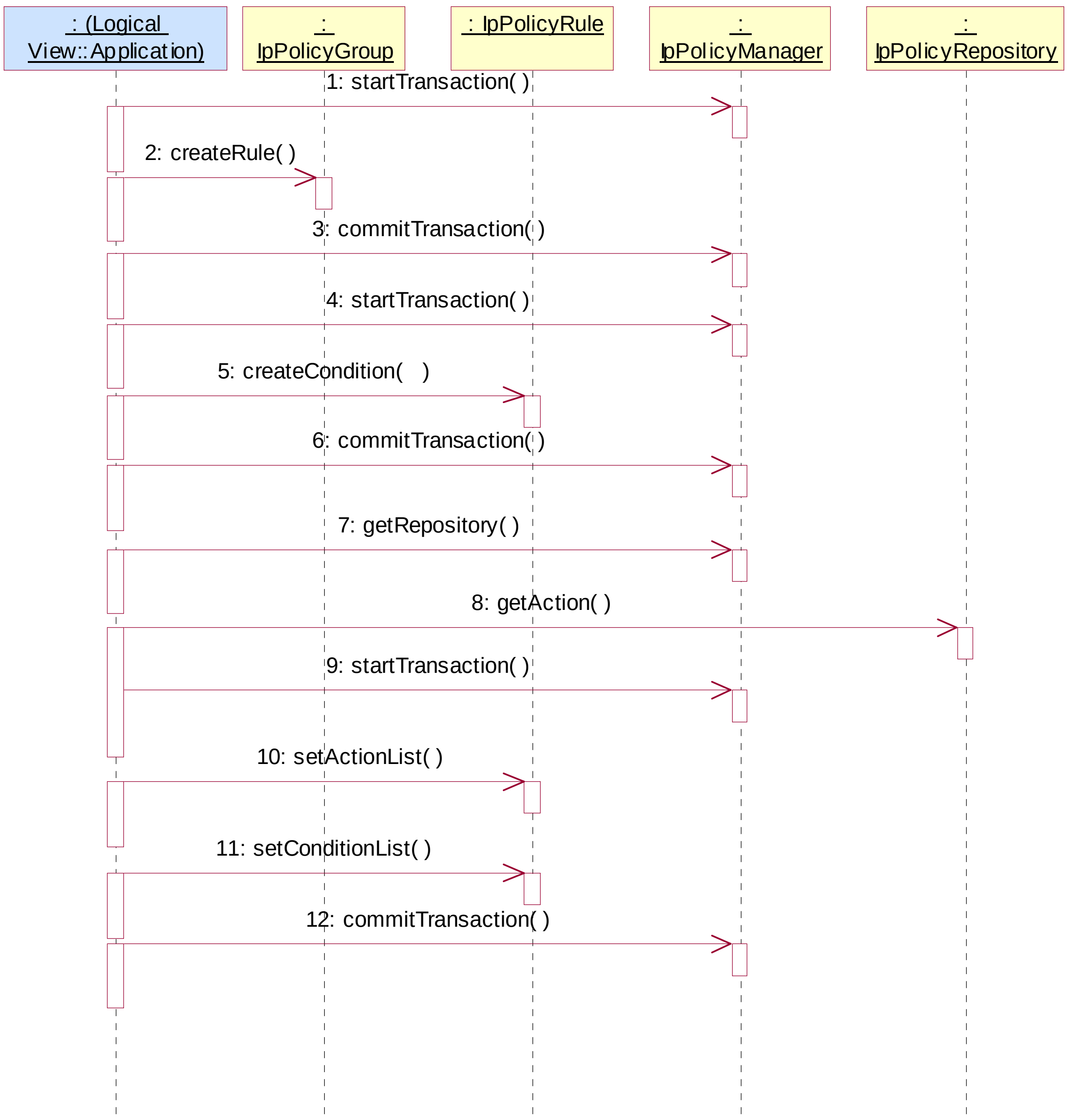
This sequence includes:

- 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.

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

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

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

6: Closes the transaction bracket.

~~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~~ IpPolicyRule object created with message 2.

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

Activation of actions is done similarly.

12: Closes the transaction bracket.

## 5.3 Create & receive an event

This sequence shows how policy events are used.

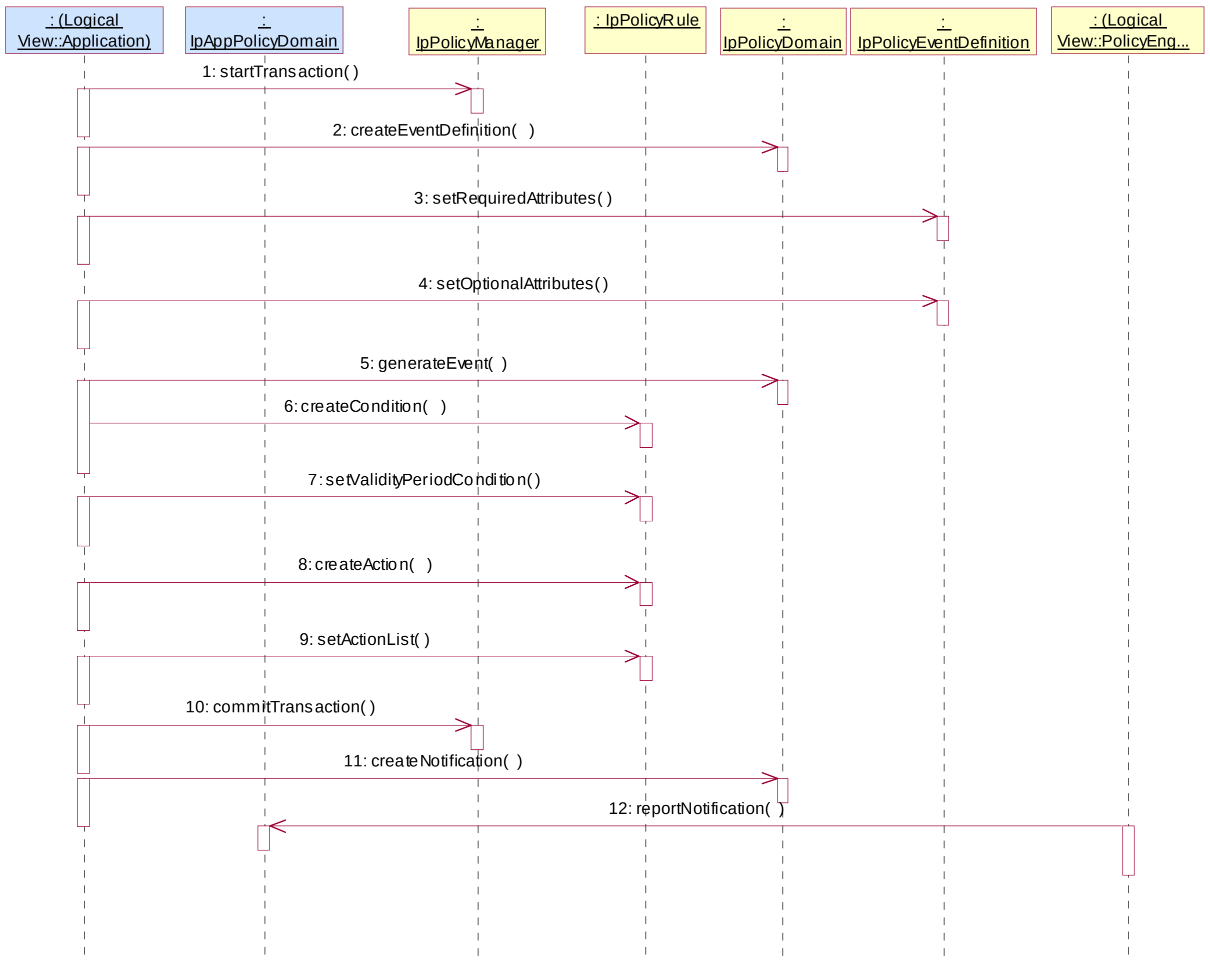
For clarification we list the different policy related objects used:

- 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.


: (Logical View::Application)
: IpAppPolicyDomain
: IpPolicyManager
: IpPolicyRule
: IpPolicyDomain
: IpPolicyEventDefinition
: (Logical View::PolicyEng...
1: startTransaction( )
2: createEventDefinition( )
3: setRequiredAttributes( )
4: setOptionalAttributes( )
5: generateEvent( )
6: createCondition( )
7: setValidityPeriodCondition( )
8: createAction( )
9: setActionList( )
10: commitTransaction( )
11: createNotification( )
12: reportNotification( )

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 an instance of PolicyTimePeriodCondition defining the validity period of this rule.

Returns a reference to the new instance of IpPolicyTimePeriodCondition object.

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

TpAttribute.AttributeName = "TimePeriod"

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

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

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

~~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.

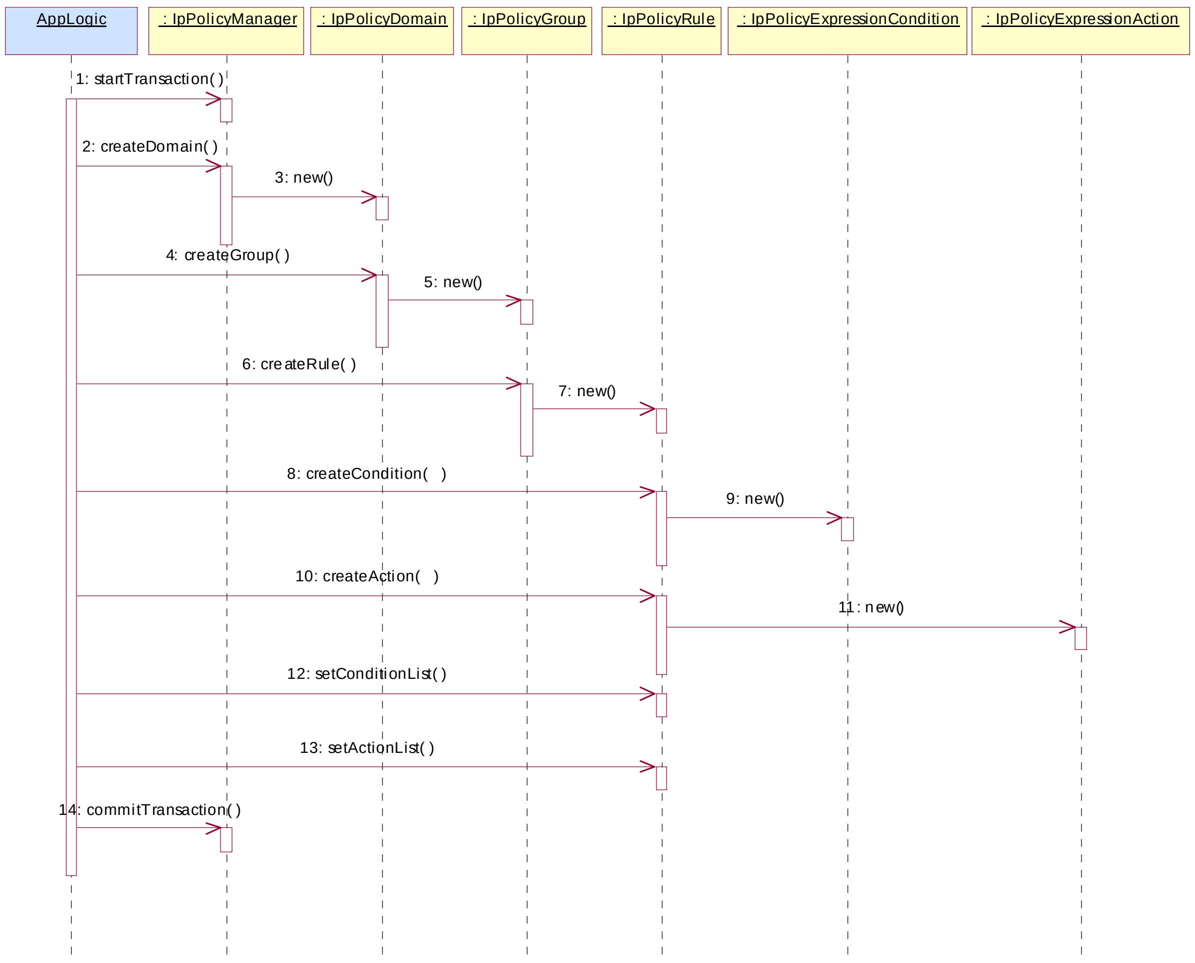
Returns a sessionID.

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.

## 5.5 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.SimpleValue.StringValue = "PrePaidCredit > CurrentCharge"

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

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

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

- ActionAttribute.AttributeName = "PurchaseAllowedExpression"

~~- ActionAttribute.AttributeType = "P_STRING"~~

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

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

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

- conditionList.Condition == <reference to the IpPolicyCondition interface returned by 9>
- conditionList.GroupNumber == 1; indicates how the conditions need to be grouped in DNF or CNF in case more groups of rules exist.
- conditionList.Negated == FALSE.

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

- actionList.Action == <reference to the IpPolicyAction interface returned by step 10>
- actionList.SequenceNumber == 1;

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

```
+-------------------------------------------------------------+
|PolicyDomain "eCommerceDomain"                               |
|                                                             |
|    +---------------------------------------------------+    |
|    |PolicyGroup "PrePaidGroup"                         |    |
|    |                                                   |    |
|    | +----------------------------------------------+  |    |
|    | |PolicyRule "SufficientCreditRule"             |  |    |
|    | |                                              |  |    |
|    | | +-------------------+  +-------------------+ |  |    |
|    | | |PolicyCondition    |  |PolicyAction       | |  |    |
|    | | | "SufficientCredit"|  | "PurchaseAllowed" | |  |    |
|    | | +-------------------+  +-------------------+ |  |    |
|    | +----------------------------------------------+  |    |
|    +---------------------------------------------------+    |
+-------------------------------------------------------------+
```

## 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.

In case this SCF supports both BNF and XML, then the TpAttributeTagInfo of the TpAttribute that populated this expression is used to distinguish between XML and BNF string contents.  A TpAttributeTagInfo value of P_XML_TYPE indicates XML as contents of the Expression attribute and a TpAttributeTagInfo value of P_SIMPLE_TYPE indicates BNF as contents of Expression attribute.

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~~P_INT32, ~~TpFloat~~ P_FLOAT or ~~TpString~~P_STRING} 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 P_INT32~~TpInt32~~ or P_FLOAT~~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~~concatenation, 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 P_FLOAT~~TpFloat~~ (or P_INT32~~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.

### 8.14.1 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.

In case this SCF supports both BNF and XML, then the TpAttributeTagInfo of the TpAttribute that populated this expression is used to distinguish between XML and BNF string contents. A TpAttributeTagInfo value of P_XML_TYPE indicates XML as contents of the Expression attribute and a TpAttributeTagInfo value of P_SIMPLE_TYPE indicates BNF as contents of Expression attribute.

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 P_INT32, P_FLOAT or P_STRING ~~, 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~~quotes) '='. 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 P_INT32 or P_FLOAT~~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~~quotes) '*' (concatenation), '-' (string difference) and '/' (string overlap).

3. Example showing an assignment expression formed using Variables of type P_FLOAT (or P_INT32)~~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.

# 10 PM Service Properties

The following table lists properties relevant to all the PM SCFs

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

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

```
P_SUPPORTED_ATTRIBUTE_TAGS = {
P_SIMPLE_TYPE
}
P_SUPPORTED_SIMPLE_ATTRIBUTE_TYPES = {
P_STRING,
P_FLOAT,
P_INT32,
}
```

# 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 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

**joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5)** **N5-030645**
**Meeting #25, Bangkok, THAILAND, 27 - 31 October 2003**

CR-Form-v7

# CHANGE REQUEST

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ⌘ | **29.198-14** | **CR** | **017** | ⌘ **rev** | **-** | ⌘ | Current version: | **5.3.0** ⌘ |

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

***Proposed change affects:*** UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network [X]

| | | | | |
|---|---|---|---|---|
| ***Title:*** | ⌘ | Correction of description of TpAttributeType to adequately support possible types - Align with 29.198-02 | | |
| ***Source:*** | ⌘ | CN5 (IBM scottjb@us.ibm.com, Telcordia, jlbakker@research.telcordia.com) | | |
| ***Work item code:*** | ⌘ | OSA2 | ***Date:*** ⌘ | 19/11/2003 |
| ***Category:*** | ⌘ | **F** | ***Release:*** ⌘ | *REL-5* |

*Use one of the following categories:*
**F** *(correction)*
**A** *(corresponds to a correction in an earlier release)*
**B** *(addition of feature),*
**C** *(functional modification of feature)*
**D** *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

*Use one of the following releases:*

| | |
|---|---|
| *2* | *(GSM Phase 2)* |
| *R96* | *(Release 1996)* |
| *R97* | *(Release 1997)* |
| *R98* | *(Release 1998)* |
| *R99* | *(Release 1999)* |
| *Rel-4* | *(Release 4)* |
| *Rel-5* | *(Release 5)* |
| *Rel-6* | *(Release 6)* |

| | | |
|---|---|---|
| ***Reason for change:*** | ⌘ | The usage of TpAttributeType in PAM is not consistent with the definition of it in 29.198-02. This inconstency causes portability problems.<br><br>The PAM spec. uses but does not define P_ADDRESS and P_PAM_CAPABILITY. Additionally, the usage of the P_ prefix is not consistent with its definition in 29.198-02, section 5.1.13. |
| ***Summary of change:*** | ⌘ | The PAM spec. uses but does not define P_ADDRESS and P_PAM_CAPABILITY. |
| ***Consequences if not approved:*** | ⌘ | The application programmer can not generically write the appropriate application code to support a TpAny that is some kind of object or primivite datatype that is not in the current TpAttribute table. This leaves the implementations to deviate from the standard specification. |

| | | | | | |
|---|---|---|---|---|---|
| ***Clauses affected:*** | ⌘ | 11.10 | | | |
| | | **Y** | **N** | | |
| ***Other specs affected:*** | ⌘ | | **X** | Other core specifications | ⌘ |
| | | | **X** | Test specifications | |
| | | | **X** | O&M Specifications | |
| ***Other comments:*** | ⌘ | Child CR to CR-29.198-02 Rel-5 in N5-030643 | | | |

**How to create CRs using this form:**
Comprehensive information and tips about how to create CRs can be found at http://www.3gpp.org/specs/CR.htm.

**Change in Clause 11.10**

## 11.10 Pre-defined Entity Types and Attributes

This version of the specification pre-defines one identity type called "Presentity". The following constant can be used to refer to this Identity Type. All identities in the PAM service are associated with this identity type. For example, the identityType parameter in IpIdentityPresence and IpEventHandler methods take this as the value. This is also used in the event registration data structure (e.g., TpPAMAVCEventData) in the IdentityType field.

| Character String Value | Description |
|---|---|
| P_PAM_PRESENTITY_TYPE | The pre-defined identity type called Presentity. |

Every identity type in PAM can be defined with a set of attributes that are associated with all identities of that type. The following dynamic attributes are pre-defined as attributes of type TpPAMAttribute for the "Presentity" identity type and shall be supported as attributes of all identities in implementations of this service. These attributes are defined using TpPAMAttributeDef fields as follows:

| AttributeName | AttributeType | IsStatic | IsRevertOn Expiration | DefaultValue | Description |
|---|---|---|---|---|---|
| P_SUBSCRIBER_STATUS | P_STRING | False | False | None | Specifies the status of the subscriber |
| P_NETWORK_STATUS | P_STRING | False | False | None | Specifies the status of the network |
| P_COMMUNICATION_MEANS | P_org/csapi/pam/TpPAMCapability~~P_PAM_CAPABILITY~~ | False | False | None | Specifies the means of communication. The type is TpPAMCapability |
| P_CONTACT_ADDRESS | P_org/csapi/TpAddress~~P_ADDRESS~~ | False | False | None | Address for communication |
| P_SUBSCRIBER_PROVIDED_LOCATION | P_STRING | False | False | None | Location nformation provided by subscriber. Is optional. |
| P_NETWORK_PROVIDED_LOCATION | P_STRING | False | False | None | Location information provided by subscriber. Is optional. |
| P_PRIORITY | P_INT32 | False | False | None | Priority for communication |
| P_OTHER_INFO | P_STRING | False | False | None | Additional information |

**End of Change in Clause 11.10**
**End of 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-020196 | -- | -- | Draft v200 submitted to TSG CN#16 for Approval | 2.0.0 | 5.0.0 |
| Sep 2002 | CN_17 | NP-020440 | 001 | -- | Add text to clarify requirements on support of methods | 5.0.0 | 5.1.0 |
| Sep 2002 | CN_17 | NP-020440 | 002 | -- | Remove declaration of unused datatype TpPAMTime | 5.0.0 | 5.1.0 |
| Sep 2002 | CN_17 | NP-020395 | 003 | -- | Add text to clarify relationship between 3GPP and ETSI/Parlay OSA specifications | 5.0.0 | 5.1.0 |
| Jun 2003 | CN_20 | NP-030245 | 004 | -- | Make TpPAMCapability extensible by changing its type to TpString | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030240 | 005 | -- | Change the type of TpPAMFQName to TpURN | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 006 | -- | Clarifiy use of askerData parameter to getAuthToken method in each PAM SCF | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 007 | -- | Add authToken parameter to computeAvailability method | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 008 | -- | Replace use of IpInterfaceRef in PAM with actual application interfaces | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 009 | -- | Add expiration time for PAM event registrations | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 010 | -- | Send subscription notification cancellation to watchers | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030241 | 011 | -- | Change PAM Presence and Availability SCF name to PAM Access | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 012 | -- | Move Access Control Mechanism to Manager Interface | 5.1.0 | 5.2.0 |
| Sep 2003 | CN_21 | NP-030352 | 013 | -- | Correction to Java Realisation Annex | 5.2.0 | 5.3.0 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

**joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5)** **N5-030646**
**Meeting #25, Bangkok, THAILAND, 27 - 31 October 2003**

*CR-Form-v7*

# CHANGE REQUEST

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ⌘ | **29.198-14** | **CR** | **018** | ⌘ **rev** | **-** | ⌘ Current version: | **5.3.0** ⌘ |

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

***Proposed change affects:*** UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network [X]

| | | |
|---|---|---|
| ***Title:*** | ⌘ | Correction of definitin of TpPAMAttribute and addition of Service Properties to publish supported attribute types - Align with 29.198-02 |
| ***Source:*** | ⌘ | CN5 (Telcordia, jlbakker@research.telcordia.com) |
| ***Work item code:*** | ⌘ | OSA2 |
| ***Date:*** | ⌘ | 19/11/2003 |
| ***Category:*** | ⌘ | **F** |
| ***Release:*** | ⌘ | *REL-5* |

*Use one of the following categories:*
- ***F*** *(correction)*
- ***A*** *(corresponds to a correction in an earlier release)*
- ***B*** *(addition of feature),*
- ***C*** *(functional modification of feature)*
- ***D*** *(editorial modification)*

Detailed explanations of the above categories can be found in 3GPP TR 21.900.

*Use one of the following releases:*

| | |
|---|---|
| *2* | *(GSM Phase 2)* |
| *R96* | *(Release 1996)* |
| *R97* | *(Release 1997)* |
| *R98* | *(Release 1998)* |
| *R99* | *(Release 1999)* |
| *Rel-4* | *(Release 4)* |
| *Rel-5* | *(Release 5)* |
| *Rel-6* | *(Release 6)* |

| | | |
|---|---|---|
| ***Reason for change:*** | ⌘ | The usage of TpAttributeType in PAM is not consistent with the definition of it in 29.198-02. Attribute typing system is not sufficient to ensure portable applications; vendors can define custom types such that different SCFs can not be integrated. |
| ***Summary of change:*** | ⌘ | TpPAMAttribute is alligned with TpAttribute. Additionally, addition of Service Properties. The Service Properties list the supported attribute types. This allows for PAM applications to discover a matching PAM SCF instance through the Framework. |
| ***Consequences if not approved:*** | ⌘ | PAM SCFs will define proprietary extensions to support additional data types, making any applications that use them vendor specific. |

| | | |
|---|---|---|
| ***Clauses affected:*** | ⌘ | 10, 11.2.1 |

| | | Y | N | | |
|---|---|---|---|---|---|
| ***Other specs affected:*** | ⌘ | | X | Other core specifications | ⌘ |
| | | | X | Test specifications | |
| | | | X | O&M Specifications | |

| | | |
|---|---|---|
| ***Other comments:*** | ⌘ | Child CR to CR-29.198-02 Rel-5 in N5-030643 |

**How to create CRs using this form:**
Comprehensive information and tips about how to create CRs can be found at http://www.3gpp.org/specs/CR.htm.

**Change in Clause 10**

## 10 PAM Service Properties

The following table lists properties relevant to all the PAM SCFs

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

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

```
P_SUPPORTED_ATTRIBUTE_TAGS = {
P_SIMPLE_TYPE
}
P_SUPPORTED_SIMPLE_ATTRIBUTE_TYPES = {
P_STRING,
P_FLOAT,
P_INT32
}
```

**End of Change in Clause 10**

**Change in Clause 11.2.1**

### 11.2.1 TpPAMAttribute

This is a `Sequence of Data Elements` containing the attribute name~~, type~~, expiration time and value. This is derived from the common attribute type TpAttribute to add the expiration value for dynamic attributes.

| Sequence Element Name | Sequence Element Type | Notes |
|---|---|---|
| AttributeName | TpString | The name of the attribute. |
| AttributeValue | TpAttributeValue | The typed value(s) for the attribute. |
| ~~AttributeType~~ | ~~TpAttributeType~~ | ~~The type of the attribute. Valid values for Type must include at least TpString, TpInt32 and TpFloat.~~ |
| ~~AttributeValue~~ | ~~TpAny~~ | ~~The values for the attribute. This model allows multi-valued attributes. Cannot be an empty list.~~ |
| ExpiresIn | TpPAMTimeInterval | The interval in milliseconds in which the attribute values are valid. A time interval of PAM_MAX_LONGINT indicates static attribute values that never expire. A time interval of 0 or negative values indicate an expired value and the time for which it has expired. |

**End of Change in Clause 11.2.1**
**End of 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-020196 | -- | -- | Draft v200 submitted to TSG CN#16 for Approval | 2.0.0 | 5.0.0 |
| Sep 2002 | CN_17 | NP-020440 | 001 | -- | Add text to clarify requirements on support of methods | 5.0.0 | 5.1.0 |
| Sep 2002 | CN_17 | NP-020440 | 002 | -- | Remove declaration of unused datatype TpPAMTime | 5.0.0 | 5.1.0 |
| Sep 2002 | CN_17 | NP-020395 | 003 | -- | Add text to clarify relationship between 3GPP and ETSI/Parlay OSA specifications | 5.0.0 | 5.1.0 |
| Jun 2003 | CN_20 | NP-030245 | 004 | -- | Make TpPAMCapability extensible by changing its type to TpString | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030240 | 005 | -- | Change the type of TpPAMFQName to TpURN | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 006 | -- | Clarifiy use of askerData parameter to getAuthToken method in each PAM SCF | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 007 | -- | Add authToken parameter to computeAvailability method | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 008 | -- | Replace use of IpInterfaceRef in PAM with actual application interfaces | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 009 | -- | Add expiration time for PAM event registrations | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 010 | -- | Send subscription notification cancellation to watchers | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030241 | 011 | -- | Change PAM Presence and Availability SCF name to PAM Access | 5.1.0 | 5.2.0 |
| Jun 2003 | CN_20 | NP-030245 | 012 | -- | Move Access Control Mechanism to Manager Interface | 5.1.0 | 5.2.0 |
| Sep 2003 | CN_21 | NP-030352 | 013 | -- | Correction to Java Realisation Annex | 5.2.0 | 5.3.0 |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |