



# eXtensible Access Control Markup Language (XACML) Version 3.0

## Committee Specification Draft 06

19 April 2012

### Specification URIs

#### This version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-csd06-en.doc> (Authoritative)  
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-csd06-en.html>  
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-csd06-en.pdf>

#### Previous version:

<http://www.oasis-open.org/committees/download.php/43799/xacml-3.0-core-spec-csprd03-en.zip>

#### Latest version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.doc> (Authoritative)  
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html>  
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.pdf>

#### Technical Committee:

[OASIS eXtensible Access Control Markup Language \(XACML\) TC](#)

#### Chairs:

Bill Parducci ([bill@parducci.net](mailto:bill@parducci.net)), Individual  
Hal Lockhart ([hal.lockhart@oracle.com](mailto:hal.lockhart@oracle.com)), Oracle

#### Editor:

Erik Rissanen ([erik@axiomatics.com](mailto:erik@axiomatics.com)), Axiomatics AB

#### Additional artifacts:

This prose specification is one component of a Work Product which also includes:

- XML schema: <http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd>

#### Related work:

This specification replaces or supersedes:

- *eXtensible Access Control Markup Language (XACML) Version 2.0*. 01 February 2005. OASIS Standard.  
[http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-core-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf).

#### Declared XML namespaces:

- `urn:oasis:names:tc:xacml:3.0:core:schema:wd-17`

#### Abstract:

This specification defines version 3.0 of the eXtensible Access Control Markup Language.

#### Status:

This document was last revised or approved by the OASIS eXtensible Access Control Markup Language (XACML) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the

“Send A Comment” button on the Technical Committee’s web page at <http://www.oasis-open.org/committees/xacml/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/xacml/ipr.php>).

**Citation format:**

When referencing this specification the following citation format should be used:

**[XACML-V3.0]**

*eXtensible Access Control Markup Language (XACML) Version 3.0*. 19 April 2012. OASIS Committee Specification Draft 06.

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-csd06-en.html>.

---

## Notices

Copyright © OASIS Open 2012. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

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

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

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

---

# Table of Contents

1	Introduction.....	9
1.1	Glossary (non-normative) .....	9
1.1.1	Preferred terms.....	9
1.1.2	Related terms .....	11
1.2	Terminology .....	11
1.3	Schema organization and namespaces .....	12
1.4	Normative References .....	12
1.5	Non-Normative References .....	13
2	Background (non-normative).....	14
2.1	Requirements .....	14
2.2	Rule and policy combining.....	15
2.3	Combining algorithms .....	15
2.4	Multiple subjects .....	16
2.5	Policies based on subject and resource attributes .....	16
2.6	Multi-valued attributes.....	16
2.7	Policies based on resource contents.....	16
2.8	Operators .....	17
2.9	Policy distribution .....	17
2.10	Policy indexing.....	17
2.11	Abstraction layer .....	18
2.12	Actions performed in conjunction with enforcement.....	18
2.13	Supplemental information about a decision.....	18
3	Models (non-normative) .....	19
3.1	Data-flow model.....	19
3.2	XACML context.....	20
3.3	Policy language model.....	21
3.3.1	Rule .....	21
3.3.2	Policy .....	22
3.3.3	Policy set .....	24
4	Examples (non-normative) .....	25
4.1	Example one.....	25
4.1.1	Example policy .....	25
4.1.2	Example request context.....	26
4.1.3	Example response context.....	28
4.2	Example two .....	28
4.2.1	Example medical record instance .....	28
4.2.2	Example request context.....	29
4.2.3	Example plain-language rules .....	31
4.2.4	Example XACML rule instances.....	31
5	Syntax (normative, with the exception of the schema fragments) .....	43
5.1	Element <PolicySet> .....	43
5.2	Element <Description> .....	45
5.3	Element <PolicyIssuer> .....	45

5.4 Element <PolicySetDefaults> .....	45
5.5 Element <XPathVersion> .....	46
5.6 Element <Target> .....	46
5.7 Element <AnyOf> .....	46
5.8 Element <AllOf> .....	47
5.9 Element <Match> .....	47
5.10 Element <PolicySetIdReference> .....	48
5.11 Element <PolicyIdReference> .....	48
5.12 Simple type VersionType .....	48
5.13 Simple type VersionMatchType .....	49
5.14 Element <Policy> .....	49
5.15 Element <PolicyDefaults> .....	51
5.16 Element <CombinerParameters> .....	51
5.17 Element <CombinerParameter> .....	52
5.18 Element <RuleCombinerParameters> .....	52
5.19 Element <PolicyCombinerParameters> .....	53
5.20 Element <PolicySetCombinerParameters> .....	53
5.21 Element <Rule> .....	54
5.22 Simple type EffectType .....	55
5.23 Element <VariableDefinition> .....	55
5.24 Element <VariableReference> .....	55
5.25 Element <Expression> .....	56
5.26 Element <Condition> .....	56
5.27 Element <Apply> .....	56
5.28 Element <Function> .....	57
5.29 Element <AttributeDesignator> .....	57
5.30 Element <AttributeSelector> .....	59
5.31 Element <AttributeValue> .....	60
5.32 Element <Obligations> .....	60
5.33 Element <AssociatedAdvice> .....	60
5.34 Element <Obligation> .....	61
5.35 Element <Advice> .....	61
5.36 Element <AttributeAssignment> .....	61
5.37 Element <ObligationExpressions> .....	62
5.38 Element <AdviceExpressions> .....	62
5.39 Element <ObligationExpression> .....	63
5.40 Element <AdviceExpression> .....	63
5.41 Element <AttributeAssignmentExpression> .....	64
5.42 Element <Request> .....	65
5.43 Element <RequestDefaults> .....	66
5.44 Element <Attributes> .....	66
5.45 Element <Content> .....	66
5.46 Element <Attribute> .....	67
5.47 Element <Response> .....	67
5.48 Element <Result> .....	68

5.49	Element <PolicyIdentifierList>	69
5.50	Element <MultiRequests>	69
5.51	Element <RequestReference>	69
5.52	Element <AttributesReference>	70
5.53	Element <Decision>	70
5.54	Element <Status>	70
5.55	Element <StatusCode>	71
5.56	Element <StatusMessage>	71
5.57	Element <StatusDetail>	71
5.58	Element <MissingAttributeDetail>	72
6	XPath 2.0 definitions	74
7	Functional requirements	76
7.1	Unicode issues	76
7.1.1	Normalization	76
7.1.2	Version of Unicode	76
7.2	Policy enforcement point	76
7.2.1	Base PEP	76
7.2.2	Deny-biased PEP	76
7.2.3	Permit-biased PEP	77
7.3	Attribute evaluation	77
7.3.1	Structured attributes	77
7.3.2	Attribute bags	77
7.3.3	Multivalued attributes	78
7.3.4	Attribute Matching	78
7.3.5	Attribute Retrieval	78
7.3.6	Environment Attributes	79
7.3.7	AttributeSelector evaluation	79
7.4	Expression evaluation	80
7.5	Arithmetic evaluation	80
7.6	Match evaluation	80
7.7	Target evaluation	81
7.8	VariableReference Evaluation	82
7.9	Condition evaluation	82
7.10	Extended Indeterminate	83
7.11	Rule evaluation	83
7.12	Policy evaluation	83
7.13	Policy Set evaluation	84
7.14	Policy and Policy set value for Indeterminate Target	84
7.15	PolicySetIdReference and PolicyIdReference evaluation	85
7.16	Hierarchical resources	85
7.17	Authorization decision	85
7.18	Obligations and advice	85
7.19	Exception handling	86
7.19.1	Unsupported functionality	86
7.19.2	Syntax and type errors	86

7.19.3	Missing attributes .....	86
7.20	Identifier equality.....	86
8	XACML extensibility points (non-normative) .....	88
8.1	Extensible XML attribute types .....	88
8.2	Structured attributes .....	88
9	Security and privacy considerations (non-normative) .....	89
9.1	Threat model.....	89
9.1.1	Unauthorized disclosure.....	89
9.1.2	Message replay .....	89
9.1.3	Message insertion .....	89
9.1.4	Message deletion .....	90
9.1.5	Message modification.....	90
9.1.6	NotApplicable results.....	90
9.1.7	Negative rules.....	90
9.1.8	Denial of service .....	91
9.2	Safeguards.....	91
9.2.1	Authentication.....	91
9.2.2	Policy administration .....	91
9.2.3	Confidentiality .....	92
9.2.4	Policy integrity .....	92
9.2.5	Policy identifiers .....	92
9.2.6	Trust model.....	93
9.2.7	Privacy.....	93
9.3	Unicode security issues .....	94
9.4	Identifier equality.....	94
10	Conformance .....	95
10.1	Introduction .....	95
10.2	Conformance tables.....	95
10.2.1	Schema elements.....	95
10.2.2	Identifier Prefixes.....	96
10.2.3	Algorithms.....	96
10.2.4	Status Codes .....	97
10.2.5	Attributes .....	97
10.2.6	Identifiers .....	97
10.2.7	Data-types .....	98
10.2.8	Functions .....	98
10.2.9	Identifiers planned for future deprecation.....	103
Appendix A.	Data-types and functions (normative) .....	105
A.1	Introduction.....	105
A.2	Data-types .....	105
A.3	Functions .....	107
A.3.1	Equality predicates.....	107
A.3.2	Arithmetic functions.....	109
A.3.3	String conversion functions.....	110
A.3.4	Numeric data-type conversion functions.....	110

A.3.5 Logical functions .....	110
A.3.6 Numeric comparison functions.....	111
A.3.7 Date and time arithmetic functions.....	111
A.3.8 Non-numeric comparison functions .....	112
A.3.9 String functions .....	115
A.3.10 Bag functions .....	119
A.3.11 Set functions .....	120
A.3.12 Higher-order bag functions .....	120
A.3.13 Regular-expression-based functions .....	124
A.3.14 Special match functions .....	126
A.3.15 XPath-based functions.....	126
A.3.16 Other functions.....	127
A.3.17 Extension functions and primitive types.....	127
A.4 Functions, data types, attributes and algorithms planned for deprecation .....	128
Appendix B.    XACML identifiers (normative) .....	130
B.1 XACML namespaces.....	130
B.2 Attribute categories .....	130
B.3 Data-types .....	130
B.4 Subject attributes.....	131
B.5 Resource attributes .....	132
B.6 Action attributes.....	132
B.7 Environment attributes .....	132
B.8 Status codes.....	133
B.9 Combining algorithms.....	133
Appendix C.    Combining algorithms (normative) .....	135
C.1 Extended Indeterminate values.....	135
C.2 Deny-overrides .....	135
C.3 Ordered-deny-overrides .....	137
C.4 Permit-overrides .....	137
C.5 Ordered-permit-overrides.....	138
C.6 Deny-unless-permit.....	139
C.7 Permit-unless-deny .....	139
C.8 First-applicable .....	140
C.9 Only-one-applicable .....	142
C.10 Legacy Deny-overrides .....	143
C.11 Legacy Ordered-deny-overrides .....	144
C.12 Legacy Permit-overrides .....	145
C.13 Legacy Ordered-permit-overrides .....	147
Appendix D.    Acknowledgements .....	148
Appendix E.    Revision History .....	149

---

# 1 Introduction

## 1.1 Glossary (non-normative)

### 1.1.1 Preferred terms

#### Access

Performing an *action*

#### Access control

Controlling *access* in accordance with a *policy* or *policy set*

#### Action

An operation on a *resource*

#### Advice

A supplementary piece of information in a *policy* or *policy set* which is provided to the *PEP* with the *decision* of the *PDP*.

#### Applicable policy

The set of *policies* and *policy sets* that governs *access* for a specific *decision request*

#### Attribute

Characteristic of a *subject*, *resource*, *action* or *environment* that may be referenced in a *predicate* or *target* (see also – *named attribute*)

#### Authorization decision

The result of evaluating *applicable policy*, returned by the *PDP* to the *PEP*. A function that evaluates to "Permit", "Deny", "Indeterminate" or "NotApplicable", and (optionally) a set of *obligations and advice*

#### Bag

An unordered collection of values, in which there may be duplicate values

#### Condition

An expression of *predicates*. A function that evaluates to "True", "False" or "Indeterminate"

#### Conjunctive sequence

A sequence of *predicates* combined using the logical 'AND' operation

#### Context

The canonical representation of a *decision request* and an *authorization decision*

#### Context handler

The system entity that converts *decision requests* in the native request format to the XACML canonical form, coordinates with Policy Information Points to add attribute values to the request context, and converts *authorization decisions* in the XACML canonical form to the native response format

#### Decision

The result of evaluating a *rule*, *policy* or *policy set*

#### Decision request

The request by a *PEP* to a *PDP* to render an *authorization decision*

- 39 **Disjunctive sequence**
- 40 A sequence of *predicates* combined using the logical 'OR' operation
- 41 **Effect**
- 42 The intended consequence of a satisfied *rule* (either "Permit" or "Deny")
- 43 **Environment**
- 44 The set of *attributes* that are relevant to an *authorization decision* and are independent of a  
45 particular *subject, resource* or *action*
- 46 **Identifier equality**
- 47 The identifier equality operation which is defined in section 7.20.
- 48 **Issuer**
- 49 A set of *attributes* describing the source of a *policy*
- 50 **Named attribute**
- 51 A specific instance of an *attribute*, determined by the *attribute* name and type, the identity of the  
52 *attribute* holder (which may be of type: *subject, resource, action* or *environment*) and  
53 (optionally) the identity of the issuing authority
- 54 **Obligation**
- 55 An operation specified in a *rule, policy* or *policy set* that should be performed by the *PEP* in  
56 conjunction with the enforcement of an *authorization decision*
- 57 **Policy**
- 58 A set of *rules*, an identifier for the *rule-combining algorithm* and (optionally) a set of  
59 *obligations* or *advice*. May be a component of a *policy set*
- 60 **Policy administration point (PAP)**
- 61 The system entity that creates a *policy* or *policy set*
- 62 **Policy-combining algorithm**
- 63 The procedure for combining the *decision* and *obligations* from multiple *policies*
- 64 **Policy decision point (PDP)**
- 65 The system entity that evaluates *applicable policy* and renders an *authorization decision*.  
66 This term is defined in a joint effort by the IETF Policy Framework Working Group and the  
67 Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198].  
68 This term corresponds to "Access Decision Function" (ADF) in [ISO10181-3].
- 69 **Policy enforcement point (PEP)**
- 70 The system entity that performs *access control*, by making *decision requests* and enforcing  
71 *authorization decisions*. This term is defined in a joint effort by the IETF Policy Framework  
72 Working Group and the Distributed Management Task Force (DMTF)/Common Information Model  
73 (CIM) in [RFC3198]. This term corresponds to "Access Enforcement Function" (AEF) in  
74 [ISO10181-3].
- 75 **Policy information point (PIP)**
- 76 The system entity that acts as a source of *attribute* values
- 77 **Policy set**
- 78 A set of *policies*, other *policy sets*, a *policy-combining algorithm* and (optionally) a set of  
79 *obligations* or *advice*. May be a component of another *policy set*
- 80 **Predicate**
- 81 A statement about *attributes* whose truth can be evaluated
- 82 **Resource**

83 Data, service or system component

#### 84 Rule

85 A **target**, an **effect**, a **condition** and (optionally) a set of **obligations** or **advice**. A component of  
86 a **policy**

#### 87 Rule-combining algorithm

88 The procedure for combining **decisions** from multiple **rules**

#### 89 Subject

90 An actor whose **attributes** may be referenced by a **predicate**

#### 91 Target

92 The set of **decision requests**, identified by definitions for **resource**, **subject** and **action** that a  
93 **rule**, **policy**, or **policy set** is intended to evaluate

#### 94 Type Unification

95 The method by which two type expressions are "unified". The type expressions are matched  
96 along their structure. Where a type variable appears in one expression it is then "unified" to  
97 represent the corresponding structure element of the other expression, be it another variable or  
98 subexpression. All variable assignments must remain consistent in both structures. Unification  
99 fails if the two expressions cannot be aligned, either by having dissimilar structure, or by having  
100 instance conflicts, such as a variable needs to represent both "xs:string" and "xs:integer". For a  
101 full explanation of **type unification**, please see [Hancock].

### 102 1.1.2 Related terms

103 In the field of **access control** and authorization there are several closely related terms in common use.  
104 For purposes of precision and clarity, certain of these terms are not used in this specification.

105 For instance, the term **attribute** is used in place of the terms: group and role.

106 In place of the terms: privilege, permission, authorization, entitlement and right, we use the term **rule**.

107 The term object is also in common use, but we use the term **resource** in this specification.

108 Requestors and initiators are covered by the term **subject**.

### 109 1.2 Terminology

110 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD  
111 NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described  
112 in [RFC2119].

113 This specification contains schema conforming to W3C XML Schema and normative text to describe the  
114 syntax and semantics of XML-encoded **policy** statements.

115

116 Listings of XACML schema appear like this.

117

118 Example code listings appear like this.

119

120 Conventional XML namespace prefixes are used throughout the listings in this specification to stand for  
121 their respective namespaces as follows, whether or not a namespace declaration is present in the  
122 example:

- 123 • The prefix `xacml:` stands for the XACML 3.0 namespace.
- 124 • The prefix `ds:` stands for the W3C XML Signature namespace [DS].
- 125 • The prefix `xs:` stands for the W3C XML Schema namespace [XS].

- 126 • The prefix `xf:` stands for the XQuery 1.0 and XPath 2.0 Function and Operators specification  
127 namespace **[XF]**.
  - 128 • The prefix `xml:` stands for the XML namespace <http://www.w3.org/XML/1998/namespace>.
- 129 This specification uses the following typographical conventions in text: `<XACMLElement>`,  
130 `<ns:ForeignElement>`, Attribute, Datatype, OtherCode. Terms in **bold-face italic** are intended  
131 to have the meaning defined in the Glossary.

### 132 1.3 Schema organization and namespaces

133 The XACML syntax is defined in a schema associated with the following XML namespace:  
134 `urn:oasis:names:tc:xacml:3.0:core:schema:wd-17`

### 135 1.4 Normative References

- 136 **[CMF]** Martin J. Dürst et al, eds., *Character Model for the World Wide Web 1.0:*  
137 *Fundamentals*, W3C Recommendation 15 February 2005,  
138 <http://www.w3.org/TR/2005/REC-charmod-20050215/>
- 139 **[DS]** D. Eastlake et al., *XML-Signature Syntax and Processing*,  
140 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 141 **[exc-c14n]** J. Boyer et al, eds., *Exclusive XML Canonicalization, Version 1.0*, W3C  
142 Recommendation 18 July 2002, [http://www.w3.org/TR/2002/REC-xml-exc-c14n-](http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/)  
143 [20020718/](http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/)
- 144 **[Hancock]** Hancock, *Polymorphic Type Checking*, in Simon L. Peyton Jones,  
145 *Implementation of Functional Programming Languages*, Section 8,  
146 Prentice-Hall International, 1987.
- 147 **[Hier]** *XACML v3.0 Hierarchical Resource Profile Version 1.0*. 11 March 2010.  
148 Committee Specification Draft 03. [http://docs.oasis-open.org/xacml/3.0/xacml-](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-03-en.html)  
149 [3.0-hierarchical-v1-spec-cd-03-en.html](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-03-en.html)
- 150 **[IEEE754]** IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8,  
151 IEEE Product No. SH10116-TBR.
- 152 **[ISO10181-3]** ISO/IEC 10181-3:1996 Information technology – Open Systems Interconnection -  
153 - Security frameworks for open systems: Access control framework.
- 154 **[Kudo00]** Kudo M and Hada S, *XML document security based on provisional authorization*,  
155 Proceedings of the Seventh ACM Conference on Computer and Communications  
156 Security, Nov 2000, Athens, Greece, pp 87-96.
- 157 **[LDAP-1]** RFC2256, *A summary of the X500(96) User Schema for use with LDAPv3*,  
158 Section 5, M Wahl, December 1997, <http://www.ietf.org/rfc/rfc2256.txt>
- 159 **[LDAP-2]** RFC2798, *Definition of the inetOrgPerson*, M. Smith, April 2000  
160 <http://www.ietf.org/rfc/rfc2798.txt>
- 161 **[MathML]** *Mathematical Markup Language (MathML)*, Version 2.0, W3C Recommendation,  
162 21 October 2003. Available at: [http://www.w3.org/TR/2003/REC-MathML2-](http://www.w3.org/TR/2003/REC-MathML2-20031021/)  
163 [20031021/](http://www.w3.org/TR/2003/REC-MathML2-20031021/)
- 164 **[Multi]** OASIS Committee Draft 03, *XACML v3.0 Multiple Decision Profile Version 1.0*,  
165 11 March 2010, [http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cd-03-en.doc)  
166 [cd-03-en.doc](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cd-03-en.doc)
- 167 **[Perritt93]** Perritt, H. Knowbots, *Permissions Headers and Contract Law*, Conference on  
168 Technological Strategies for Protecting Intellectual Property in the Networked  
169 Multimedia Environment, April 1993. Available at:  
170 <http://www.ifla.org/documents/infopol/copyright/perh2.txt>
- 171 **[RBAC]** David Ferraiolo and Richard Kuhn, *Role-Based Access Controls*, 15th National  
172 Computer Security Conference, 1992.
- 173 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,  
174 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

175 **[RFC2396]** Berners-Lee T, Fielding R, Masinter L, *Uniform Resource Identifiers (URI):*  
176 *Generic Syntax*. Available at: <http://www.ietf.org/rfc/rfc2396.txt>

177 **[RFC2732]** Hinden R, Carpenter B, Masinter L, *Format for Literal IPv6 Addresses in URL's*.  
178 Available at: <http://www.ietf.org/rfc/rfc2732.txt>

179 **[RFC3198]** IETF RFC 3198: *Terminology for Policy-Based Management*, November 2001.  
180 <http://www.ietf.org/rfc/rfc3198.txt>

181 **[UAX15]** Mark Davis, Martin Dürst, *Unicode Standard Annex #15: Unicode Normalization*  
182 *Forms, Unicode 5.1*, available from <http://unicode.org/reports/tr15/>

183 **[UTR36]** Davis, Mark, Suignard, Michel, *Unicode Technocal Report #36: Unicode Security*  
184 *Considerations*. Available at <http://www.unicode.org/reports/tr36/>

185 **[XACMLAdmin]** OASIS Committee Draft 03, *XACML v3.0 Administration and Delegation Profile*  
186 *Version 1.0*. 11 March 2010. [http://docs.oasis-open.org/xacml/3.0/xacml-3.0-](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-administration-v1-spec-cd-03-en.doc)  
187 [administration-v1-spec-cd-03-en.doc](http://docs.oasis-open.org/xacml/3.0/xacml-3.0-administration-v1-spec-cd-03-en.doc)

188 **[XACMLv1.0]** OASIS Standard, *Extensible access control markup language (XACML) Version*  
189 *1.0*. 18 February 2003. [http://www.oasis-](http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf)  
190 [open.org/committees/download.php/2406/oasis-xacml-1.0.pdf](http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf)

191 **[XACMLv1.1]** OASIS Committee Specification, *Extensible access control markup language*  
192 *(XACML) Version 1.1*. 7 August 2003. [http://www.oasis-](http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf)  
193 [open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf](http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf)

194 **[XF]** *XQuery 1.0 and XPath 2.0 Functions and Operators*, W3C Recommendation 23  
195 January 2007. Available at: [http://www.w3.org/TR/2007/REC-xpath-functions-](http://www.w3.org/TR/2007/REC-xpath-functions-20070123/)  
196 [20070123/](http://www.w3.org/TR/2007/REC-xpath-functions-20070123/)

197 **[XML]** Bray, Tim, et.al. eds, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*,  
198 W3C Recommendation 26 November 2008, available at  
199 <http://www.w3.org/TR/2008/REC-xml-20081126/>

200 **[XMLid]** Marsh, Jonathan, et.al. eds, *xml:id Version 1.0*. W3C Recommendation 9  
201 September 2005. Available at: [http://www.w3.org/TR/2005/REC-xml-id-](http://www.w3.org/TR/2005/REC-xml-id-20050909/)  
202 [20050909/](http://www.w3.org/TR/2005/REC-xml-id-20050909/)

203 **[XS]** *XML Schema, parts 1 and 2*. Available at: <http://www.w3.org/TR/xmlschema-1/>  
204 and <http://www.w3.org/TR/xmlschema-2/>

205 **[XPath]** *XML Path Language (XPath), Version 1.0*, W3C Recommendation 16 November  
206 1999. Available at: <http://www.w3.org/TR/xpath>

207 **[XPathFunc]** *XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)*, W3C  
208 Recommendation 14 December 2010. Available at:  
209 <http://www.w3.org/TR/2010/REC-xpath-functions-20101214/>

210 **[XSLT]** *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation 16 November  
211 1999. Available at: <http://www.w3.org/TR/xslt>

## 212 1.5 Non-Normative References

213 **[CM]** *Character model for the World Wide Web 1.0: Normalization*, W3C Working  
214 Draft, 27 October 2005, [http://www.w3.org/TR/2005/WD-charmod-norm-](http://www.w3.org/TR/2005/WD-charmod-norm-20051027/)  
215 [20051027/](http://www.w3.org/TR/2005/WD-charmod-norm-20051027/), World Wide Web Consortium.

216 **[Hinton94]** Hinton, H, M, Lee, E, S, *The Compatibility of Policies*, Proceedings 2nd ACM  
217 Conference on Computer and Communications Security, Nov 1994, Fairfax,  
218 Virginia, USA.

219 **[Sloman94]** Sloman, M. *Policy Driven Management for Distributed Systems*. Journal of  
220 Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.

221

## 2 Background (non-normative)

222 The "economics of scale" have driven computing platform vendors to develop products with very  
223 generalized functionality, so that they can be used in the widest possible range of situations. "Out of the  
224 box", these products have the maximum possible privilege for accessing data and executing software, so  
225 that they can be used in as many application environments as possible, including those with the most  
226 permissive security policies. In the more common case of a relatively restrictive security policy, the  
227 platform's inherent privileges must be constrained by configuration.

228 The security policy of a large enterprise has many elements and many points of enforcement. Elements  
229 of policy may be managed by the Information Systems department, by Human Resources, by the Legal  
230 department and by the Finance department. And the policy may be enforced by the extranet, mail, WAN,  
231 and remote-access systems; platforms which inherently implement a permissive security policy. The  
232 current practice is to manage the configuration of each point of enforcement independently in order to  
233 implement the security policy as accurately as possible. Consequently, it is an expensive and unreliable  
234 proposition to modify the security policy. Moreover, it is virtually impossible to obtain a consolidated view  
235 of the safeguards in effect throughout the enterprise to enforce the policy. At the same time, there is  
236 increasing pressure on corporate and government executives from consumers, shareholders, and  
237 regulators to demonstrate "best practice" in the protection of the information assets of the enterprise and  
238 its customers.

239 For these reasons, there is a pressing need for a common language for expressing security policy. If  
240 implemented throughout an enterprise, a common policy language allows the enterprise to manage the  
241 enforcement of all the elements of its security policy in all the components of its information systems.  
242 Managing security policy may include some or all of the following steps: writing, reviewing, testing,  
243 approving, issuing, combining, analyzing, modifying, withdrawing, retrieving, and enforcing policy.

244 XML is a natural choice as the basis for the common security-policy language, due to the ease with which  
245 its syntax and semantics can be extended to accommodate the unique requirements of this application,  
246 and the widespread support that it enjoys from all the main platform and tool vendors.

### 2.1 Requirements

247  
248 The basic requirements of a policy language for expressing information system security policy are:

- 249 • To provide a method for combining individual **rules** and **policies** into a single **policy set** that applies  
250 to a particular **decision request**.
- 251 • To provide a method for flexible definition of the procedure by which **rules** and **policies** are  
252 combined.
- 253 • To provide a method for dealing with multiple **subjects** acting in different capacities.
- 254 • To provide a method for basing an **authorization decision** on **attributes** of the **subject** and  
255 **resource**.
- 256 • To provide a method for dealing with multi-valued **attributes**.
- 257 • To provide a method for basing an **authorization decision** on the contents of an information  
258 **resource**.
- 259 • To provide a set of logical and mathematical operators on **attributes** of the **subject**, **resource** and  
260 **environment**.
- 261 • To provide a method for handling a distributed set of **policy** components, while abstracting the  
262 method for locating, retrieving and authenticating the **policy** components.
- 263 • To provide a method for rapidly identifying the **policy** that applies to a given **action**, based upon the  
264 values of **attributes** of the **subjects**, **resource** and **action**.
- 265 • To provide an abstraction-layer that insulates the **policy**-writer from the details of the application  
266 environment.

- 267 • To provide a method for specifying a set of **actions** that must be performed in conjunction with **policy**  
268 enforcement.

269 The motivation behind XACML is to express these well-established ideas in the field of **access control**  
270 policy using an extension language of XML. The XACML solutions for each of these requirements are  
271 discussed in the following sections.

## 272 2.2 Rule and policy combining

273 The complete **policy** applicable to a particular **decision request** may be composed of a number of  
274 individual **rules** or **policies**. For instance, in a personal privacy application, the owner of the personal  
275 information may define certain aspects of disclosure policy, whereas the enterprise that is the custodian  
276 of the information may define certain other aspects. In order to render an **authorization decision**, it must  
277 be possible to combine the two separate **policies** to form the single **policy** applicable to the request.

278 XACML defines three top-level **policy** elements: <Rule>, <Policy> and <PolicySet>. The <Rule>  
279 element contains a Boolean expression that can be evaluated in isolation, but that is not intended to be  
280 accessed in isolation by a **PDP**. So, it is not intended to form the basis of an **authorization decision** by  
281 itself. It is intended to exist in isolation only within an XACML **PAP**, where it may form the basic unit of  
282 management.

283 The <Policy> element contains a set of <Rule> elements and a specified procedure for combining the  
284 results of their evaluation. It is the basic unit of **policy** used by the **PDP**, and so it is intended to form the  
285 basis of an **authorization decision**.

286 The <PolicySet> element contains a set of <Policy> or other <PolicySet> elements and a  
287 specified procedure for combining the results of their evaluation. It is the standard means for combining  
288 separate **policies** into a single combined **policy**.

289 Hinton et al [Hinton94] discuss the question of the compatibility of separate **policies** applicable to the  
290 same **decision request**.

## 291 2.3 Combining algorithms

292 XACML defines a number of combining algorithms that can be identified by a RuleCombiningAlgId or  
293 PolicyCombiningAlgId attribute of the <Policy> or <PolicySet> elements, respectively. The  
294 **rule-combining algorithm** defines a procedure for arriving at an **authorization decision** given the  
295 individual results of evaluation of a set of **rules**. Similarly, the **policy-combining algorithm** defines a  
296 procedure for arriving at an **authorization decision** given the individual results of evaluation of a set of  
297 **policies**. Some examples of standard combining algorithms are (see Appendix C for a full list of standard  
298 combining algorithms):

- 299 • Deny-overrides (Ordered and Unordered),  
300 • Permit-overrides (Ordered and Unordered),  
301 • First-applicable and  
302 • Only-one-applicable.

303 In the case of the Deny-overrides algorithm, if a single <Rule> or <Policy> element is encountered that  
304 evaluates to "Deny", then, regardless of the evaluation result of the other <Rule> or <Policy> elements  
305 in the **applicable policy**, the combined result is "Deny".

306 Likewise, in the case of the Permit-overrides algorithm, if a single "Permit" result is encountered, then the  
307 combined result is "Permit".

308 In the case of the "First-applicable" combining algorithm, the combined result is the same as the result of  
309 evaluating the first <Rule>, <Policy> or <PolicySet> element in the list of **rules** whose **target** and  
310 **condition** is applicable to the **decision request**.

311 The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The result of this  
312 combining algorithm ensures that one and only one **policy** or **policy set** is applicable by virtue of their  
313 **targets**. If no **policy** or **policy set** applies, then the result is "NotApplicable", but if more than one **policy**  
314 or **policy set** is applicable, then the result is "Indeterminate". When exactly one **policy** or **policy set** is

315 applicable, the result of the combining algorithm is the result of evaluating the single **applicable policy** or  
316 **policy set**.

317 **Policies** and **policy sets** may take parameters that modify the behavior of the combining algorithms.  
318 However, none of the standard combining algorithms is affected by parameters.

319 Users of this specification may, if necessary, define their own combining algorithms.

## 320 2.4 Multiple subjects

321 **Access control policies** often place requirements on the **actions** of more than one **subject**. For  
322 instance, the **policy** governing the execution of a high-value financial transaction may require the  
323 approval of more than one individual, acting in different capacities. Therefore, XACML recognizes that  
324 there may be more than one **subject** relevant to a **decision request**. Different **attribute** categories are  
325 used to differentiate between **subjects** acting in different capacities. Some standard values for these  
326 **attribute** categories are specified, and users may define additional ones.

## 327 2.5 Policies based on subject and resource attributes

328 Another common requirement is to base an **authorization decision** on some characteristic of the  
329 **subject** other than its identity. Perhaps, the most common application of this idea is the **subject's** role  
330 **[RBAC]**. XACML provides facilities to support this approach. **Attributes** of **subjects** contained in the  
331 request **context** may be identified by the <AttributeDesignator> element. This element contains a  
332 URN that identifies the **attribute**. Alternatively, the <AttributeSelector> element may contain an  
333 XPath expression over the <Content> element of the **subject** to identify a particular **subject attribute**  
334 value by its location in the **context** (see Section 2.11 for an explanation of **context**).

335 XACML provides a standard way to reference the **attributes** defined in the LDAP series of specifications  
336 **[LDAP-1]**, **[LDAP-2]**. This is intended to encourage implementers to use standard **attribute** identifiers for  
337 some common **subject attributes**.

338 Another common requirement is to base an **authorization decision** on some characteristic of the  
339 **resource** other than its identity. XACML provides facilities to support this approach. **Attributes** of the  
340 **resource** may be identified by the <AttributeDesignator> element. This element contains a URN  
341 that identifies the **attribute**. Alternatively, the <AttributeSelector> element may contain an XPath  
342 expression over the <Content> element of the **resource** to identify a particular **resource attribute** value  
343 by its location in the **context**.

## 344 2.6 Multi-valued attributes

345 The most common techniques for communicating **attributes** (LDAP, XPath, SAML, etc.) support multiple  
346 values per **attribute**. Therefore, when an XACML **PDP** retrieves the value of a **named attribute**, the  
347 result may contain multiple values. A collection of such values is called a **bag**. A **bag** differs from a set in  
348 that it may contain duplicate values, whereas a set may not. Sometimes this situation represents an  
349 error. Sometimes the XACML **rule** is satisfied if any one of the **attribute** values meets the criteria  
350 expressed in the **rule**.

351 XACML provides a set of functions that allow a **policy** writer to be absolutely clear about how the **PDP**  
352 should handle the case of multiple **attribute** values. These are the “higher-order” functions (see Section  
353 A.3).

## 354 2.7 Policies based on resource contents

355 In many applications, it is required to base an **authorization decision** on data contained in the  
356 information **resource** to which **access** is requested. For instance, a common component of privacy  
357 **policy** is that a person should be allowed to read records for which he or she is the **subject**. The  
358 corresponding **policy** must contain a reference to the **subject** identified in the information **resource** itself.

359 XACML provides facilities for doing this when the information **resource** can be represented as an XML  
360 document. The <AttributeSelector> element may contain an XPath expression over the

361 <Content> element of the **resource** to identify data in the information **resource** to be used in the **policy**  
362 evaluation.  
363 In cases where the information **resource** is not an XML document, specified **attributes** of the **resource**  
364 can be referenced, as described in Section 2.5.

## 365 2.8 Operators

366 Information security **policies** operate upon **attributes** of **subjects**, the **resource**, the **action** and the  
367 **environment** in order to arrive at an **authorization decision**. In the process of arriving at the  
368 **authorization decision**, **attributes** of many different types may have to be compared or computed. For  
369 instance, in a financial application, a person's available credit may have to be calculated by adding their  
370 credit limit to their account balance. The result may then have to be compared with the transaction value.  
371 This sort of situation gives rise to the need for arithmetic operations on **attributes** of the **subject** (account  
372 balance and credit limit) and the **resource** (transaction value).

373 Even more commonly, a **policy** may identify the set of roles that are permitted to perform a particular  
374 **action**. The corresponding operation involves checking whether there is a non-empty intersection  
375 between the set of roles occupied by the **subject** and the set of roles identified in the **policy**; hence the  
376 need for set operations.

377 XACML includes a number of built-in functions and a method of adding non-standard functions. These  
378 functions may be nested to build arbitrarily complex expressions. This is achieved with the <Apply>  
379 element. The <Apply> element has an XML attribute called `FunctionId` that identifies the function to  
380 be applied to the contents of the element. Each standard function is defined for specific argument data-  
381 type combinations, and its return data-type is also specified. Therefore, data-type consistency of the  
382 **policy** can be checked at the time the **policy** is written or parsed. And, the types of the data values  
383 presented in the request **context** can be checked against the values expected by the **policy** to ensure a  
384 predictable outcome.

385 In addition to operators on numerical and set arguments, operators are defined for date, time and  
386 duration arguments.

387 Relationship operators (equality and comparison) are also defined for a number of data-types, including  
388 the RFC822 and X.500 name-forms, strings, URIs, etc.

389 Also noteworthy are the operators over Boolean data-types, which permit the logical combination of  
390 **predicates** in a **rule**. For example, a **rule** may contain the statement that **access** may be permitted  
391 during business hours AND from a terminal on business premises.

392 The XACML method of representing functions borrows from MathML [**MathML**] and from the XQuery 1.0  
393 and XPath 2.0 Functions and Operators specification [**XF**].

## 394 2.9 Policy distribution

395 In a distributed system, individual **policy** statements may be written by several **policy** writers and  
396 enforced at several enforcement points. In addition to facilitating the collection and combination of  
397 independent **policy** components, this approach allows **policies** to be updated as required. XACML  
398 **policy** statements may be distributed in any one of a number of ways. But, XACML does not describe  
399 any normative way to do this. Regardless of the means of distribution, **PDPs** are expected to confirm, by  
400 examining the **policy's** <Target> element that the **policy** is applicable to the **decision request** that it is  
401 processing.

402 <Policy> elements may be attached to the information **resources** to which they apply, as described by  
403 Perritt [**Perritt93**]. Alternatively, <Policy> elements may be maintained in one or more locations from  
404 which they are retrieved for evaluation. In such cases, the **applicable policy** may be referenced by an  
405 identifier or locator closely associated with the information **resource**.

## 406 2.10 Policy indexing

407 For efficiency of evaluation and ease of management, the overall security **policy** in force across an  
408 enterprise may be expressed as multiple independent **policy** components. In this case, it is necessary to

409 identify and retrieve the **applicable policy** statement and verify that it is the correct one for the requested  
410 **action** before evaluating it. This is the purpose of the <Target> element in XACML.

411 Two approaches are supported:

- 412 1. **Policy** statements may be stored in a database. In this case, the **PDP** should form a database  
413 query to retrieve just those **policies** that are applicable to the set of **decision requests** to which  
414 it expects to respond. Additionally, the **PDP** should evaluate the <Target> element of the  
415 retrieved **policy** or **policy set** statements as defined by the XACML specification.
- 416 2. Alternatively, the **PDP** may be loaded with all available **policies** and evaluate their <Target>  
417 elements in the context of a particular **decision request**, in order to identify the **policies** and  
418 **policy sets** that are applicable to that request.

419 The use of constraints limiting the applicability of a policy was described by Sloman [Sloman94].

## 420 2.11 Abstraction layer

421 **PEPs** come in many forms. For instance, a **PEP** may be part of a remote-access gateway, part of a Web  
422 server or part of an email user-agent, etc. It is unrealistic to expect that all **PEPs** in an enterprise do  
423 currently, or will in the future, issue **decision requests** to a **PDP** in a common format. Nevertheless, a  
424 particular **policy** may have to be enforced by multiple **PEPs**. It would be inefficient to force a **policy**  
425 writer to write the same **policy** several different ways in order to accommodate the format requirements of  
426 each **PEP**. Similarly **attributes** may be contained in various envelope types (e.g. X.509 attribute  
427 certificates, SAML attribute assertions, etc.). Therefore, there is a need for a canonical form of the  
428 request and response handled by an XACML **PDP**. This canonical form is called the XACML **context**. Its  
429 syntax is defined in XML schema.

430 Naturally, XACML-conformant **PEPs** may issue requests and receive responses in the form of an XACML  
431 **context**. But, where this situation does not exist, an intermediate step is required to convert between the  
432 request/response format understood by the **PEP** and the XACML **context** format understood by the **PDP**.

433 The benefit of this approach is that **policies** may be written and analyzed independently of the specific  
434 environment in which they are to be enforced.

435 In the case where the native request/response format is specified in XML Schema (e.g. a SAML-  
436 conformant **PEP**), the transformation between the native format and the XACML **context** may be  
437 specified in the form of an Extensible Stylesheet Language Transformation [XSLT].

438 Similarly, in the case where the **resource** to which **access** is requested is an XML document, the  
439 **resource** itself may be included in, or referenced by, the request **context**. Then, through the use of  
440 XPath expressions [XPath] in the **policy**, values in the **resource** may be included in the **policy**  
441 evaluation.

## 442 2.12 Actions performed in conjunction with enforcement

443 In many applications, **policies** specify actions that MUST be performed, either instead of, or in addition  
444 to, actions that MAY be performed. This idea was described by Sloman [Sloman94]. XACML provides  
445 facilities to specify actions that MUST be performed in conjunction with **policy** evaluation in the  
446 <Obligations> element. This idea was described as a provisional action by Kudo [Kudo00]. There  
447 are no standard definitions for these actions in version 3.0 of XACML. Therefore, bilateral agreement  
448 between a **PAP** and the **PEP** that will enforce its **policies** is required for correct interpretation. **PEPs** that  
449 conform to v3.0 of XACML are required to deny **access** unless they understand and can discharge all of  
450 the <Obligations> elements associated with the **applicable policy**. <Obligations> elements are  
451 returned to the **PEP** for enforcement.

## 452 2.13 Supplemental information about a decision

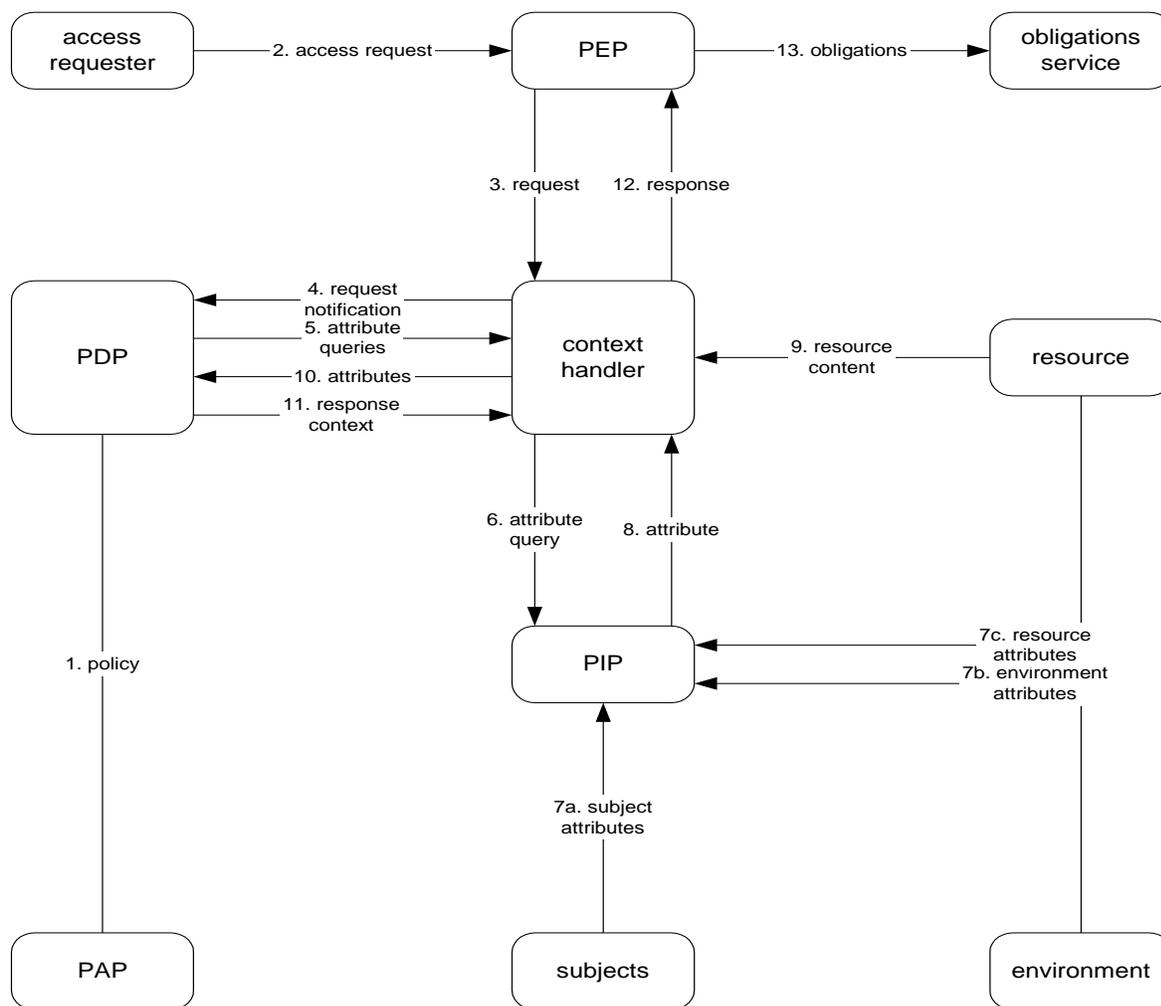
453 In some applications it is helpful to specify supplemental information about a decision. XACML provides  
454 facilities to specify supplemental information about a decision with the <Advice> element. Such **advice**  
455 may be safely ignored by the **PEP**.

### 3 Models (non-normative)

The data-flow model and language model of XACML are described in the following sub-sections.

#### 3.1 Data-flow model

The major actors in the XACML domain are shown in the data-flow diagram of Figure 1.



460  
461 *Figure 1 - Data-flow diagram*

462 Note: some of the data-flows shown in the diagram may be facilitated by a repository.  
463 For instance, the communications between the **context handler** and the **PIP** or the  
464 communications between the **PDP** and the **PAP** may be facilitated by a repository. The  
465 XACML specification is not intended to place restrictions on the location of any such  
466 repository, or indeed to prescribe a particular communication protocol for any of the data-  
467 flows.

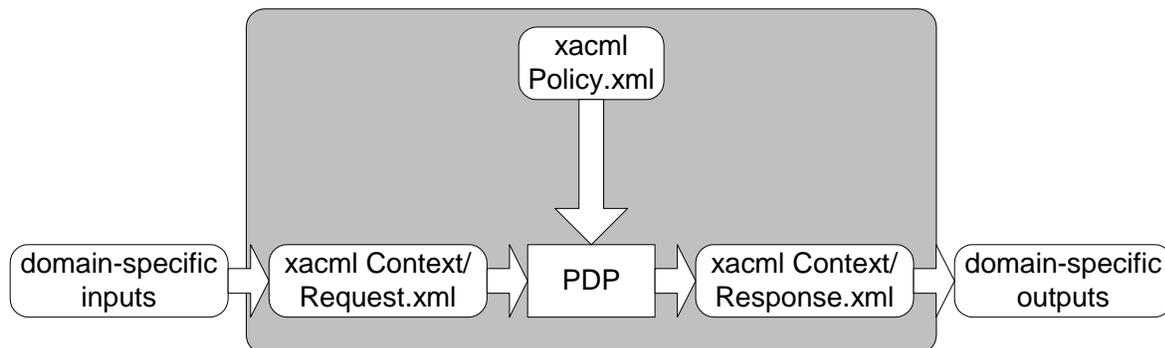
468 The model operates by the following steps.

- 469 1. **PAPs** write **policies** and **policy sets** and make them available to the **PDP**. These **policies** or  
470 **policy sets** represent the complete **policy** for a specified **target**.
- 471 2. The **access** requester sends a request for **access** to the **PEP**.

- 472 3. The **PEP** sends the request for **access** to the **context handler** in its native request format,  
473 optionally including **attributes** of the **subjects**, **resource**, **action**, **environment** and other  
474 categories.
- 475 4. The **context handler** constructs an XACML request **context**, optionally adds attributes, and  
476 sends it to the **PDP**.
- 477 5. The **PDP** requests any additional **subject**, **resource**, **action**, **environment** and other categories  
478 (not shown) **attributes** from the **context handler**.
- 479 6. The **context handler** requests the **attributes** from a **PIP**.
- 480 7. The **PIP** obtains the requested **attributes**.
- 481 8. The **PIP** returns the requested **attributes** to the **context handler**.
- 482 9. Optionally, the **context handler** includes the **resource** in the **context**.
- 483 10. The **context handler** sends the requested **attributes** and (optionally) the **resource** to the **PDP**.  
484 The **PDP** evaluates the **policy**.
- 485 11. The **PDP** returns the response **context** (including the **authorization decision**) to the **context**  
486 **handler**.
- 487 12. The **context handler** translates the response **context** to the native response format of the **PEP**.  
488 The **context handler** returns the response to the **PEP**.
- 489 13. The **PEP** fulfills the **obligations**.
- 490 14. (Not shown) If **access** is permitted, then the **PEP** permits **access** to the **resource**; otherwise, it  
491 denies **access**.

### 492 3.2 XACML context

493 XACML is intended to be suitable for a variety of application environments. The core language is  
494 insulated from the application environment by the XACML **context**, as shown in Figure 2, in which the  
495 scope of the XACML specification is indicated by the shaded area. The XACML **context** is defined in  
496 XML schema, describing a canonical representation for the inputs and outputs of the **PDP**. **Attributes**  
497 referenced by an instance of XACML **policy** may be in the form of XPath expressions over the  
498 <Content> elements of the **context**, or attribute designators that identify the **attribute** by its category,  
499 identifier, data-type and (optionally) its issuer. Implementations must convert between the **attribute**  
500 representations in the application environment (e.g., SAML, J2SE, CORBA, and so on) and the **attribute**  
501 representations in the XACML **context**. How this is achieved is outside the scope of the XACML  
502 specification. In some cases, such as SAML, this conversion may be accomplished in an automated way  
503 through the use of an XSLT transformation.



504  
505 *Figure 2 - XACML context*

506 Note: The **PDP** is not required to operate directly on the XACML representation of a **policy**. It may  
507 operate directly on an alternative representation.

508 Typical categories of **attributes** in the **context** are the **subject**, **resource**, **action** and **environment**, but  
509 users may define their own categories as needed. See appendix B.2 for suggested **attribute** categories.

510 See Section 7.3.5 for a more detailed discussion of the request **context**.

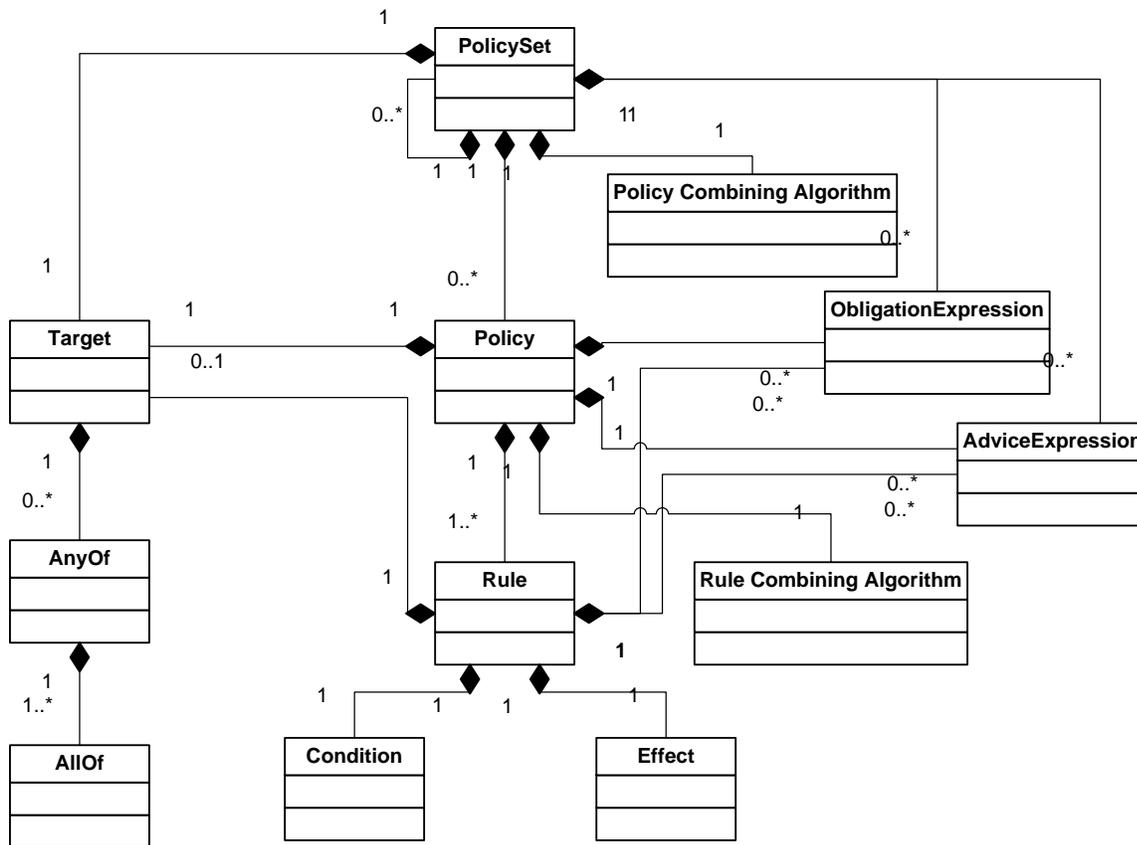
511 **3.3 Policy language model**

512 The **policy** language model is shown in Figure 3. The main components of the model are:

- 513 • **Rule**;
- 514 • **Policy**; and
- 515 • **Policy set**.

516 These are described in the following sub-sections.

517



518

519 *Figure 3 - Policy language model*

520 **3.3.1 Rule**

521 A **rule** is the most elementary unit of **policy**. It may exist in isolation only within one of the major actors of

522 the XACML domain. In order to exchange **rules** between major actors, they must be encapsulated in a

523 **policy**. A **rule** can be evaluated on the basis of its contents. The main components of a **rule** are:

- 524 • a **target**,
- 525 • an **effect**,
- 526 • a **condition**,
- 527 • **obligation** expressions, and
- 528 • **advice** expressions

529 These are discussed in the following sub-sections.

### 530 3.3.1.1 Rule target

531 The **target** defines the set of requests to which the **rule** is intended to apply in the form of a logical  
532 expression on **attributes** in the request. The <Condition> element may further refine the applicability  
533 established by the **target**. If the **rule** is intended to apply to all entities of a particular data-type, then the  
534 corresponding entity is omitted from the **target**. An XACML **PDP** verifies that the matches defined by the  
535 **target** are satisfied by the **attributes** in the request **context**.

536 The <Target> element may be absent from a <Rule>. In this case, the **target** of the <Rule> is the  
537 same as that of the parent <Policy> element.

538 Certain **subject** name-forms, **resource** name-forms and certain types of **resource** are internally  
539 structured. For instance, the X.500 directory name-form and RFC 822 name-form are structured **subject**  
540 name-forms, whereas an account number commonly has no discernible structure. UNIX file-system path-  
541 names and URIs are examples of structured **resource** name-forms. An XML document is an example of  
542 a structured **resource**.

543 Generally, the name of a node (other than a leaf node) in a structured name-form is also a legal instance  
544 of the name-form. So, for instance, the RFC822 name "med.example.com" is a legal RFC822 name  
545 identifying the set of mail addresses hosted by the med.example.com mail server. The XPath value  
546 md:record/md:patient/ is a legal XPath value identifying a node-set in an XML document.

547 The question arises: how should a name that identifies a set of **subjects** or **resources** be interpreted by  
548 the **PDP**, whether it appears in a **policy** or a request **context**? Are they intended to represent just the  
549 node explicitly identified by the name, or are they intended to represent the entire sub-tree subordinate to  
550 that node?

551 In the case of **subjects**, there is no real entity that corresponds to such a node. So, names of this type  
552 always refer to the set of **subjects** subordinate in the name structure to the identified node.  
553 Consequently, non-leaf **subject** names should not be used in equality functions, only in match functions,  
554 such as "urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match" not  
555 "urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal" (see Appendix 10.2.9).

### 556 3.3.1.2 Effect

557 The **effect** of the **rule** indicates the **rule**-writer's intended consequence of a "True" evaluation for the **rule**.  
558 Two values are allowed: "Permit" and "Deny".

### 559 3.3.1.3 Condition

560 **Condition** represents a Boolean expression that refines the applicability of the **rule** beyond the  
561 **predicates** implied by its **target**. Therefore, it may be absent.

### 562 3.3.1.4 Obligation expressions

563 **Obligation** expressions may be added by the writer of the **rule**.

564 When a **PDP** evaluates a **rule** containing **obligation** expressions, it evaluates the **obligation** expressions  
565 into **obligations** and returns certain of those **obligations** to the **PEP** in the response **context**. Section  
566 7.18 explains which **obligations** are to be returned.

### 567 3.3.1.5 Advice

568 **Advice** expressions may be added by the writer of the **rule**.

569 When a **PDP** evaluates a **rule** containing **advice** expressions, it evaluates the **advice** expressions into  
570 **advice** and returns certain of those **advice** to the **PEP** in the response **context**. Section 7.18 explains  
571 which **advice** are to be returned. In contrast to **obligations**, **advice** may be safely ignored by the **PEP**.

## 572 3.3.2 Policy

573 From the data-flow model one can see that **rules** are not exchanged amongst system entities. Therefore,  
574 a **PAP** combines **rules** in a **policy**. A **policy** comprises four main components:

- 575 • a **target**;
  - 576 • a **rule-combining algorithm**-identifier;
  - 577 • a set of **rules**;
  - 578 • **obligation** expressions and
  - 579 • **advice** expressions
- 580 **Rules** are described above. The remaining components are described in the following sub-sections.

### 581 3.3.2.1 Policy target

582 An XACML <PolicySet>, <Policy> or <Rule> element contains a <Target> element that specifies  
583 the set of requests to which it applies. The <Target> of a <PolicySet> or <Policy> may be declared  
584 by the writer of the <PolicySet> or <Policy>, or it may be calculated from the <Target> elements of  
585 the <PolicySet>, <Policy> and <Rule> elements that it contains.

586 A system entity that calculates a <Target> in this way is not defined by XACML, but there are two logical  
587 methods that might be used. In one method, the <Target> element of the outer <PolicySet> or  
588 <Policy> (the "outer component") is calculated as the union of all the <Target> elements of the  
589 referenced <PolicySet>, <Policy> or <Rule> elements (the "inner components"). In another  
590 method, the <Target> element of the outer component is calculated as the intersection of all the  
591 <Target> elements of the inner components. The results of evaluation in each case will be very  
592 different: in the first case, the <Target> element of the outer component makes it applicable to any  
593 **decision request** that matches the <Target> element of at least one inner component; in the second  
594 case, the <Target> element of the outer component makes it applicable only to **decision requests** that  
595 match the <Target> elements of every inner component. Note that computing the intersection of a set  
596 of <Target> elements is likely only practical if the **target** data-model is relatively simple.

597 In cases where the <Target> of a <Policy> is declared by the **policy** writer, any component <Rule>  
598 elements in the <Policy> that have the same <Target> element as the <Policy> element may omit  
599 the <Target> element. Such <Rule> elements inherit the <Target> of the <Policy> in which they  
600 are contained.

### 601 3.3.2.2 Rule-combining algorithm

602 The **rule-combining algorithm** specifies the procedure by which the results of evaluating the component  
603 **rules** are combined when evaluating the **policy**, i.e. the **decision** value placed in the response **context**  
604 by the **PDP** is the value of the **policy**, as defined by the **rule-combining algorithm**. A **policy** may have  
605 combining parameters that affect the operation of the **rule-combining algorithm**.

606 See Appendix Appendix C for definitions of the normative **rule-combining algorithms**.

### 607 3.3.2.3 Obligation expressions

608 **Obligation** expressions may be added by the writer of the **policy**.

609 When a **PDP** evaluates a **policy** containing **obligation** expressions, it evaluates the **obligation**  
610 expressions into **obligations** and returns certain of those **obligations** to the **PEP** in the response  
611 **context**. Section 7.18 explains which **obligations** are to be returned.

### 612 3.3.2.4 Advice

613 **Advice** expressions may be added by the writer of the **policy**.

614 When a **PDP** evaluates a **policy** containing **advice** expressions, it evaluates the **advice** expressions into  
615 **advice** and returns certain of those **advice** to the **PEP** in the response **context**. Section 7.18 explains  
616 which **advice** are to be returned. In contrast to **obligations**, **advice** may be safely ignored by the **PEP**.

### 617 3.3.3 Policy set

618 A **policy set** comprises four main components:

- 619 • a **target**;
- 620 • a **policy-combining algorithm**-identifier
- 621 • a set of **policies**;
- 622 • **obligation** expressions, and
- 623 • **advice** expressions

624 The **target** and **policy** components are described above. The other components are described in the  
625 following sub-sections.

#### 626 3.3.3.1 Policy-combining algorithm

627 The **policy-combining algorithm** specifies the procedure by which the results of evaluating the  
628 component **policies** are combined when evaluating the **policy set**, i.e. the `Decision` value placed in the  
629 response **context** by the **PDP** is the result of evaluating the **policy set**, as defined by the **policy-**  
630 **combining algorithm**. A **policy set** may have combining parameters that affect the operation of the  
631 **policy-combining algorithm**.

632 See Appendix Appendix C for definitions of the normative **policy-combining algorithms**.

#### 633 3.3.3.2 Obligation expressions

634 The writer of a **policy set** may add **obligation** expressions to the **policy set**, in addition to those  
635 contained in the component **rules**, **policies** and **policy sets**.

636 When a **PDP** evaluates a **policy set** containing **obligations** expressions, it evaluates the **obligation**  
637 expressions into **obligations** and returns certain of those **obligations** to the **PEP** in its response **context**.  
638 Section 7.18 explains which **obligations** are to be returned.

#### 639 3.3.3.3 Advice expressions

640 **Advice** expressions may be added by the writer of the **policy set**.

641 When a **PDP** evaluates a **policy set** containing **advice** expressions, it evaluates the **advice** expressions  
642 into **advice** and returns certain of those **advice** to the **PEP** in the response **context**. Section 7.18  
643 explains which **advice** are to be returned. In contrast to **obligations**, **advice** may be safely ignored by  
644 the **PEP**.

## 645 4 Examples (non-normative)

646 This section contains two examples of the use of XACML for illustrative purposes. The first example is a  
647 relatively simple one to illustrate the use of **target**, **context**, matching functions and **subject attributes**.  
648 The second example additionally illustrates the use of the **rule-combining algorithm**, **conditions** and  
649 **obligations**.

### 650 4.1 Example one

#### 651 4.1.1 Example policy

652 Assume that a corporation named Medi Corp (identified by its domain name: med.example.com) has an  
653 **access control policy** that states, in English:

654 *Any user with an e-mail name in the "med.example.com" namespace is allowed to perform any **action** on*  
655 *any resource.*

656 An XACML **policy** consists of header information, an optional text description of the **policy**, a **target**, one  
657 or more **rules** and an optional set of **obligation** expressions.

```
658 [a1] <?xml version="1.0" encoding="UTF-8"?>
659 [a2] <Policy
660 [a3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
661 [a4]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
662 [a5]   xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
663 [a6]   http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
664 [a7]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy1"
665 [a8]   Version="1.0"
666 [a9]   RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
667 [a10] <Description>
668 [a11]   Medi Corp access control policy
669 [a12] </Description>
670 [a13] <Target/>
671 [a14] <Rule
672 [a15]   RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule1"
673 [a16]   Effect="Permit">
674 [a17] <Description>
675 [a18]   Any subject with an e-mail name in the med.example.com domain
676 [a19]   can perform any action on any resource.
677 [a20] </Description>
678 [a21] <Target>
679 [a22]   <AnyOf>
680 [a23]     <AllOf>
681 [a24]       <Match
682 [a25]         MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
683 [a26]         <AttributeValue
684 [a27]           DataType="http://www.w3.org/2001/XMLSchema#string"
685 [a28]           >med.example.com</AttributeValue>
686 [a29]         <AttributeDesignator
687 [a30]           MustBePresent="false"
688 [a31]           Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
689 [a32] subject"
690 [a33]           AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
691 [a34]           DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
692 [a35]         </Match>
693 [a36]       </AllOf>
694 [a37]     </AnyOf>
695 [a38]   </Target>
696 [a39] </Rule>
697 </Policy>
```

698 [a1] is a standard XML document tag indicating which version of XML is being used and what the  
699 character encoding is.

700 [a2] introduces the XACML **Policy** itself.

701 [a3] - [a4] are XML namespace declarations.  
702 [a3] gives a URN for the XACML *polices* schema.  
703 [a7] assigns a name to this *policy* instance. The name of a *policy* has to be unique for a given *PDP* so  
704 that there is no ambiguity if one *policy* is referenced from another *policy*. The version attribute specifies  
705 the version of this policy is "1.0".  
706 [a9] specifies the algorithm that will be used to resolve the results of the various *rules* that may be in the  
707 *policy*. The deny-overrides *rule-combining algorithm* specified here says that, if any *rule* evaluates to  
708 "Deny", then the *policy* must return "Deny". If all *rules* evaluate to "Permit", then the *policy* must return  
709 "Permit". The *rule-combining algorithm*, which is fully described in Appendix Appendix C, also says  
710 what to do if an error were to occur when evaluating any *rule*, and what to do with *rules* that do not apply  
711 to a particular *decision request*.  
712 [a10] - [a12] provide a text description of the *policy*. This description is optional.  
713 [a13] describes the *decision requests* to which this *policy* applies. If the *attributes* in a *decision*  
714 *request* do not match the values specified in the *policy target*, then the remainder of the *policy* does not  
715 need to be evaluated. This *target* section is useful for creating an index to a set of *polices*. In this  
716 simple example, the *target* section says the *policy* is applicable to any *decision request*.  
717 [a14] introduces the one and only *rule* in this simple *policy*.  
718 [a15] specifies the identifier for this *rule*. Just as for a *policy*, each *rule* must have a unique identifier (at  
719 least unique for any *PDP* that will be using the *policy*).  
720 [a16] says what *effect* this *rule* has if the *rule* evaluates to "True". *Rules* can have an *effect* of either  
721 "Permit" or "Deny". In this case, if the *rule* is satisfied, it will evaluate to "Permit", meaning that, as far as  
722 this one *rule* is concerned, the requested *access* should be permitted. If a *rule* evaluates to "False",  
723 then it returns a result of "NotApplicable". If an error occurs when evaluating the *rule*, then the *rule*  
724 returns a result of "Indeterminate". As mentioned above, the *rule-combining algorithm* for the *policy*  
725 specifies how various *rule* values are combined into a single *policy* value.  
726 [a17] - [a20] provide a text description of this *rule*. This description is optional.  
727 [a21] introduces the *target* of the *rule*. As described above for the *target* of a *policy*, the *target* of a *rule*  
728 describes the *decision requests* to which this *rule* applies. If the *attributes* in a *decision request* do  
729 not match the values specified in the *rule target*, then the remainder of the *rule* does not need to be  
730 evaluated, and a value of "NotApplicable" is returned to the *rule* evaluation.  
731 The *rule target* is similar to the *target* of the *policy* itself, but with one important difference. [a22] - [a36]  
732 spells out a specific value that the *subject* in the *decision request* must match. The <Match> element  
733 specifies a matching function in the MatchId attribute, a literal value of "med.example.com" and a pointer  
734 to a specific *subject attribute* in the request *context* by means of the <AttributeDesignator>  
735 element with an *attribute* category which specifies the *access subject*. The matching function will be  
736 used to compare the literal value with the value of the *subject attribute*. Only if the match returns "True"  
737 will this *rule* apply to a particular *decision request*. If the match returns "False", then this *rule* will return  
738 a value of "NotApplicable".  
739 [a38] closes the *rule*. In this *rule*, all the work is done in the <Target> element. In more complex *rules*,  
740 the <Target> may have been followed by a <Condition> element (which could also be a set of  
741 *conditions* to be ANDed or ORed together).  
742 [a39] closes the *policy*. As mentioned above, this *policy* has only one *rule*, but more complex *polices*  
743 may have any number of *rules*.

## 744 4.1.2 Example request context

745 Let's examine a hypothetical *decision request* that might be submitted to a *PDP* that executes the  
746 *policy* above. In English, the *access* request that generates the *decision request* may be stated as  
747 follows:

748 *Bart Simpson, with e-mail name "bs@simpsons.com", wants to read his medical record at Medi Corp.*

749 In XACML, the information in the *decision request* is formatted into a request *context* statement that  
750 looks as follows:

```

751 [b1] <?xml version="1.0" encoding="UTF-8"?>
752 [b2] <Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
753 [b3] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
754 [b4] xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
755 http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
756 [b5] ReturnPolicyIdList="false">
757 [b6] <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
758 subject">
759 [b7] <Attribute IncludeInResult="false"
760 [b8] AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
761 [b9] <AttributeValue
762 [b10] DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
763 [b11] >bs@simpsons.com</AttributeValue>
764 [b12] </Attribute>
765 [b13] </Attributes>
766 [b14] <Attributes
767 [b15] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
768 [b16] <Attribute IncludeInResult="false"
769 [b17] AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
770 [b18] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
771 [b19] >file://example/med/record/patient/BartSimpson</AttributeValue>
772 [b20] </Attribute>
773 [b21] </Attributes>
774 [b22] <Attributes
775 [b23] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
776 [b24] <Attribute IncludeInResult="false"
777 [b25] AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
778 [b26] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
779 [b27] >read</AttributeValue>
780 [b28] </Attribute>
781 [b29] </Attributes>
782 [b30] </Request>

```

783 [b1] - [b2] contain the header information for the request **context**, and are used the same way as the  
784 header for the **policy** explained above.

785 The first <Attributes> element contains **attributes** of the entity making the **access** request. There  
786 can be multiple **subjects** in the form of additional <Attributes> elements with different categories, and  
787 each **subject** can have multiple **attributes**. In this case, in [b6] - [b13], there is only one **subject**, and the  
788 **subject** has only one **attribute**: the **subject's** identity, expressed as an e-mail name, is  
789 "bs@simpsons.com".

790 The second <Attributes> element contains **attributes** of the **resource** to which the **subject** (or  
791 **subjects**) has requested **access**. Lines [b14] - [b21] contain the one **attribute** of the **resource** to which  
792 Bart Simpson has requested **access**: the **resource** identified by its file URI, which is  
793 "file://medico/record/patient/BartSimpson".

794 The third <Attributes> element contains **attributes** of the **action** that the **subject** (or **subjects**)  
795 wishes to take on the **resource**. [b22] - [b29] describe the identity of the **action** Bart Simpson wishes to  
796 take, which is "read".

797 [b30] closes the request **context**. A more complex request **context** may have contained some **attributes**  
798 not associated with the **subject**, the **resource** or the **action**. Environment would be an example of such  
799 an attribute category. These would have been placed in additional <Attributes> elements. Examples  
800 of such **attributes** are **attributes** describing the **environment** or some application specific category of  
801 **attributes**.

802 The **PDP** processing this request **context** locates the **policy** in its **policy** repository. It compares the  
803 **attributes** in the request **context** with the **policy target**. Since the **policy target** is empty, the **policy**  
804 matches this **context**.

805 The **PDP** now compares the **attributes** in the request **context** with the **target** of the one **rule** in this  
806 **policy**. The requested **resource** matches the <Target> element and the requested **action** matches the  
807 <Target> element, but the requesting **subject-id attribute** does not match "med.example.com".

### 808 4.1.3 Example response context

809 As a result of evaluating the *policy*, there is no *rule* in this *policy* that returns a "Permit" result for this  
810 request. The *rule-combining algorithm* for the *policy* specifies that, in this case, a result of  
811 "NotApplicable" should be returned. The response *context* looks as follows:

```
812 [c1] <?xml version="1.0" encoding="UTF-8"?>  
813 [c2] <Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"  
814     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
815     xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17  
816     http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd">  
817 [c3]   <Result>  
818 [c4]     <Decision>NotApplicable</Decision>  
819 [c5]   </Result>  
820 [c6] </Response>
```

821 [c1] - [c2] contain the same sort of header information for the response as was described above for a  
822 *policy*.

823 The <Result> element in lines [c3] - [c5] contains the result of evaluating the *decision request* against  
824 the *policy*. In this case, the result is "NotApplicable". A *policy* can return "Permit", "Deny",  
825 "NotApplicable" or "Indeterminate". Therefore, the *PEP* is required to deny *access*.

826 [c6] closes the response *context*.

## 827 4.2 Example two

828 This section contains an example XML document, an example request *context* and example XACML  
829 *rules*. The XML document is a medical record. Four separate *rules* are defined. These illustrate a *rule-*  
830 *combining algorithm*, *conditions* and *obligation* expressions.

### 831 4.2.1 Example medical record instance

832 The following is an instance of a medical record to which the example XACML *rules* can be applied. The  
833 <record> schema is defined in the registered namespace administered by Medi Corp.

```
834 [d1] <?xml version="1.0" encoding="UTF-8"?>  
835 [d2] <record xmlns="urn:example:med:schemas:record"  
836     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
837 [d4]   <patient>  
838 [d5]     <patientName>  
839 [d6]       <first>Bartholomew</first>  
840 [d7]       <last>Simpson</last>  
841 [d8]     </patientName>  
842 [d9]     <patientContact>  
843 [d10]       <street>27 Shelbyville Road</street>  
844 [d11]       <city>Springfield</city>  
845 [d12]       <state>MA</state>  
846 [d13]       <zip>12345</zip>  
847 [d14]       <phone>555.123.4567</phone>  
848 [d15]       <fax/>  
849 [d16]       <email/>  
850 [d17]     </patientContact>  
851 [d18]     <patientDoB>1992-03-21</patientDoB>  
852 [d19]     <patientGender>male</patientGender>  
853 [d20]     <patient-number>555555</patient-number>  
854 [d21]   </patient>  
855 [d22]   <parentGuardian>  
856 [d23]     <parentGuardianId>HS001</parentGuardianId>  
857 [d24]     <parentGuardianName>  
858 [d25]       <first>Homer</first>  
859 [d26]       <last>Simpson</last>  
860 [d27]     </parentGuardianName>  
861 [d28]     <parentGuardianContact>  
862 [d29]       <street>27 Shelbyville Road</street>  
863 [d30]       <city>Springfield</city>  
864 [d31]       <state>MA</state>  
865 [d32]       <zip>12345</zip>  
866 [d33]       <phone>555.123.4567</phone>  
867 [d34]       <fax/>
```

```

868     [d35]         <email>homers@aol.com</email>
869     [d36]         </parentGuardianContact>
870     [d37]         </parentGuardian>
871     [d38]         <primaryCarePhysician>
872     [d39]         <physicianName>
873     [d40]         <first>Julius</first>
874     [d41]         <last>Hibbert</last>
875     [d42]         </physicianName>
876     [d43]         <physicianContact>
877     [d44]         <street>1 First St</street>
878     [d45]         <city>Springfield</city>
879     [d46]         <state>MA</state>
880     [d47]         <zip>12345</zip>
881     [d48]         <phone>555.123.9012</phone>
882     [d49]         <fax>555.123.9013</fax>
883     [d50]         <email/>
884     [d51]         </physicianContact>
885     [d52]         <registrationID>ABC123</registrationID>
886     [d53]         </primaryCarePhysician>
887     [d54]         <insurer>
888     [d55]         <name>Blue Cross</name>
889     [d56]         <street>1234 Main St</street>
890     [d57]         <city>Springfield</city>
891     [d58]         <state>MA</state>
892     [d59]         <zip>12345</zip>
893     [d60]         <phone>555.123.5678</phone>
894     [d61]         <fax>555.123.5679</fax>
895     [d62]         <email/>
896     [d63]         </insurer>
897     [d64]         <medical>
898     [d65]         <treatment>
899     [d66]         <drug>
900     [d67]         <name>methylphenidate hydrochloride</name>
901     [d68]         <dailyDosage>30mgs</dailyDosage>
902     [d69]         <startDate>1999-01-12</startDate>
903     [d70]         </drug>
904     [d71]         <comment>
905     [d72]         patient exhibits side-effects of skin coloration and carpal degeneration
906     [d73]         </comment>
907     [d74]         </treatment>
908     [d75]         <result>
909     [d76]         <test>blood pressure</test>
910     [d77]         <value>120/80</value>
911     [d78]         <date>2001-06-09</date>
912     [d79]         <performedBy>Nurse Betty</performedBy>
913     [d80]         </result>
914     [d81]         </medical>
915     [d82]         </record>

```

## 916 4.2.2 Example request context

917 The following example illustrates a request *context* to which the example *rules* may be applicable. It  
918 represents a request by the physician Julius Hibbert to read the patient date of birth in the record of  
919 Bartholomew Simpson.

```

920     [e1]         <?xml version="1.0" encoding="UTF-8"?>
921     [e2]         <Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
922     [e3]         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
923     [e4]         xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
924     http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
925     [e5]         ReturnPolicyIdList="false">
926     [e6]         <Attributes
927     [e7]         Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
928     [e8]         <Attribute IncludeInResult="false"
929     [e9]         AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
930     [e10]        Issuer="med.example.com">
931     [e11]         <AttributeValue
932     [e12]        DataType="http://www.w3.org/2001/XMLSchema#string">CN=Julius
933     Hibbert</AttributeValue>
934     [e13]        </Attribute>
935     [e14]        <Attribute IncludeInResult="false"
936     [e15]        AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"

```

```

937 [e16] Issuer="med.example.com">
938 [e17] <AttributeValue
939 [e18]   DataType="http://www.w3.org/2001/XMLSchema#string"
940 [e19]   >physician</AttributeValue>
941 [e20] </Attribute>
942 [e21] <Attribute IncludeInResult="false"
943 [e22]   AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id"
944 [e23]   Issuer="med.example.com">
945 [e24]   <AttributeValue
946 [e25]     DataType="http://www.w3.org/2001/XMLSchema#string">jh1234</AttributeValue>
947 [e26]   </Attribute>
948 [e27] </Attributes>
949 [e28] <Attributes
950 [e29]   Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
951 [e30]   <Content>
952 [e31]     <md:record xmlns:md="urn:example:med:schemas:record"
953 [e32]       xsi:schemaLocation="urn:example:med:schemas:record
954 [e33]       http://www.med.example.com/schemas/record.xsd">
955 [e34]       <md:patient>
956 [e35]         <md:patientDoB>1992-03-21</md:patientDoB>
957 [e36]         <md:patient-number>555555</md:patient-number>
958 [e37]         <md:patientContact>
959 [e38]           <md:email>b.simpson@example.com</md:email>
960 [e39]         </md:patientContact>
961 [e40]       </md:patient>
962 [e41]     </md:record>
963 [e42]   </Content>
964 [e43]   <Attribute IncludeInResult="false"
965 [e44]     AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector" >
966 [e45]     <AttributeValue
967 [e46]       XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
968 [e47]       DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
969 [e48]       >md:record/md:patient/md:patientDoB</AttributeValue>
970 [e49]     </Attribute>
971 [e50]   <Attribute IncludeInResult="false"
972 [e51]     AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace" >
973 [e52]     <AttributeValue
974 [e53]       DataType="http://www.w3.org/2001/XMLSchema#anyURI"
975 [e54]       >urn:example:med:schemas:record</AttributeValue>
976 [e55]     </Attribute>
977 [e56]   </Attributes>
978 [e57] <Attributes
979 [e58]   Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
980 [e59]   <Attribute IncludeInResult="false"
981 [e60]     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" >
982 [e61]     <AttributeValue
983 [e62]       DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
984 [e63]     </Attribute>
985 [e64]   </Attributes>
986 [e65] <Attributes
987 [e66]   Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
988 [e67]   <Attribute IncludeInResult="false"
989 [e68]     AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date" >
990 [e69]     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#date"
991 [e70]       >2010-01-11</AttributeValue>
992 [e71]     </Attribute>
993 [e72]   </Attributes>
994 [e73] </Request>

```

995 [e2] - [e4] Standard namespace declarations.

996 [e6] - [e27] **Access subject attributes** are placed in the urn:oasis:names:tc:xacml:1.0:subject-  
997 category:access-subject **attribute** category of the <Request> element. Each **attribute** consists of the  
998 **attribute** meta-data and the **attribute** value. There is only one **subject** involved in this request. This  
999 value of the **attribute** category denotes the identity for which the request was issued.

1000 [e8] - [e13] **Subject** subject-id **attribute**.

1001 [e14] - [e20] **Subject** role **attribute**.

1002 [e21] - [e26] **Subject** physician-id **attribute**.

1003 [e28] - [e56] **Resource attributes** are placed in the urn:oasis:names:tc:xacml:3.0:attribute-  
 1004 category:resource **attribute** category of the <Request> element. Each **attribute** consists of **attribute**  
 1005 meta-data and an **attribute** value.

1006 [e30] - [e42] **Resource** content. The XML **resource** instance, **access** to all or part of which may be  
 1007 requested, is placed here.

1008 [e43] - [e49] The identifier of the **Resource** instance for which **access** is requested, which is an XPath  
 1009 expression into the <Content> element that selects the data to be accessed.

1010 [e57] - [e64] **Action attributes** are placed in the urn:oasis:names:tc:xacml:3.0:attribute-category:action  
 1011 **attribute** category of the <Request> element.

1012 [e59] - [e63] **Action** identifier.

### 1013 4.2.3 Example plain-language rules

1014 The following plain-language **rules** are to be enforced:

- 1015 Rule 1: A person, identified by his or her patient number, may read any record for which he or she is  
 1016 the designated patient.
- 1017 Rule 2: A person may read any record for which he or she is the designated parent or guardian, and  
 1018 for which the patient is under 16 years of age.
- 1019 Rule 3: A physician may write to any medical element for which he or she is the designated primary  
 1020 care physician, provided an email is sent to the patient.
- 1021 Rule 4: An administrator shall not be permitted to read or write to medical elements of a patient  
 1022 record.

1023 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

### 1024 4.2.4 Example XACML rule instances

#### 1025 4.2.4.1 Rule 1

1026 **Rule 1** illustrates a simple **rule** with a single <Condition> element. It also illustrates the use of the  
 1027 <VariableDefinition> element to define a function that may be used throughout the **policy**. The  
 1028 following XACML <Rule> instance expresses **Rule 1**:

```

1029 [f1] <?xml version="1.0" encoding="UTF-8"?>
1030 [f2] <Policy
1031 [f3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1032 [f4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1033 [f5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1034 [f6]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1035 [f7]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:1"
1036 [f8]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1037 algorithm:deny-overrides"
1038 [f9]   Version="1.0">
1039 [f10]   <PolicyDefaults>
1040 [f11]     <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1041 [f12]   </PolicyDefaults>
1042 [f13]   <Target/>
1043 [f14]   <VariableDefinition VariableId="17590034">
1044 [f15]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1045 [f16]       <Apply
1046 [f17]         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1047 [f18]           <AttributeDesignator
1048 [f19]             MustBePresent="false"
1049 [f20]             Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
1050 subject"
1051 [f21]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:patient-
1052 number"
1053 [f22]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1054 [f23]         </Apply>
1055 [f24]       <Apply
1056 [f25]         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
```

```

1057 [f26]         <AttributeSelector
1058 [f27]             MustBePresent="false"
1059 [f28]             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1060 [f29]             Path="md:record/md:patient/md:patient-number/text()"
1061 [f30]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1062 [f31]         </Apply>
1063 [f32]     </Apply>
1064 [f33] </VariableDefinition>
1065 [f34] <Rule
1066 [f35]     RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"
1067 [f36]     Effect="Permit">
1068 [f37]     <Description>
1069 [f38]         A person may read any medical record in the
1070 [f39]         http://www.med.example.com/schemas/record.xsd namespace
1071 [f40]         for which he or she is the designated patient
1072 [f41]     </Description>
1073 [f42]     <Target>
1074 [f43]         <AnyOf>
1075 [f44]             <AllOf>
1076 [f45]                 <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1077 [f46]                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1078 [f47]                         >urn:example:med:schemas:record</AttributeValue>
1079 [f48]                     <AttributeDesignator
1080 [f49]                         MustBePresent="false"
1081 [f50]                         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1082 [f51]                         AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1083 [f52]                         DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1084 [f53]                 </Match>
1085 [f54]                 <Match
1086 [f55]                     MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1087 [f56]                         <AttributeValue
1088 [f57]                             DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1089 [f58]                             XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1090 [f59]                             >md:record</AttributeValue>
1091 [f60]                         <AttributeDesignator
1092 [f61]                             MustBePresent="false"
1093 [f62]                             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1094 [f63]                             AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1095 [f64]                             DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1096 [f65]                         </Match>
1097 [f66]                     </AllOf>
1098 [f67]                 </AnyOf>
1099 [f68]             <AnyOf>
1100 [f69]                 <AllOf>
1101 [f70]                     <Match
1102 [f71]                         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1103 [f72]                             <AttributeValue
1104 [f73]                                 DataType="http://www.w3.org/2001/XMLSchema#string"
1105 [f74]                                 >read</AttributeValue>
1106 [f75]                             <AttributeDesignator
1107 [f76]                                 MustBePresent="false"
1108 [f77]                                 Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1109 [f78]                                 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1110 [f79]                                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1111 [f80]                             </Match>
1112 [f81]                     </AllOf>
1113 [f82]                 </AnyOf>
1114 [f83]             </Target>
1115 [f84]         <Condition>
1116 [f85]             <VariableReference VariableId="17590034"/>
1117 [f86]         </Condition>
1118 [f87]     </Rule>
1119 [f88] </Policy>

```

1120 [f3] - [f6] XML namespace declarations.

1121 [f11] XPath expressions in the **policy** are to be interpreted according to the 1.0 version of the XPath specification.

1123 [f14] - [f33] A <VariableDefinition> element. It defines a function that evaluates the truth of the statement: the patient-number **subject attribute** is equal to the patient-number in the **resource**.

1125 [f15] The `FunctionId` attribute names the function to be used for comparison. In this case, comparison  
1126 is done with the “urn:oasis:names:tc:xacml:1.0:function:string-equal” function; this function takes two  
1127 arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1128 [f17] The first argument of the variable definition is a function specified by the `FunctionId` attribute.  
1129 Since urn:oasis:names:tc:xacml:1.0:function:string-equal takes arguments of type  
1130 “http://www.w3.org/2001/XMLSchema#string” and `AttributeDesignator` selects a **bag** of type  
1131 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-and-  
1132 only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly one  
1133 value.

1134 [f18] The `AttributeDesignator` selects a **bag** of values for the patient-number **subject attribute** in  
1135 the request **context**.

1136 [f25] The second argument of the variable definition is a function specified by the `FunctionId` attribute.  
1137 Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes arguments of type  
1138 “http://www.w3.org/2001/XMLSchema#string” and the `AttributeSelector` selects a **bag** of type  
1139 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-and-  
1140 only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly one  
1141 value.

1142 [f26] The `<AttributeSelector>` element selects a **bag** of values from the **resource** content using a  
1143 free-form XPath expression. In this case, it selects the value of the patient-number in the **resource**.  
1144 Note that the namespace prefixes in the XPath expression are resolved with the standard XML  
1145 namespace declarations.

1146 [f35] **Rule** identifier.

1147 [f36] **Rule effect** declaration. When a **rule** evaluates to ‘True’ it emits the value of the `Effect` attribute.  
1148 This value is then combined with the `Effect` values of other **rules** according to the **rule-combining**  
1149 **algorithm**.

1150 [f37] - [f41] Free form description of the **rule**.

1151 [f42] - [f83] A **rule target** defines a set of **decision requests** that the **rule** is intended to evaluate.

1152 [f43] - [f67] The `<AnyOf>` element contains a **disjunctive sequence** of `<AllOf>` elements. In this  
1153 example, there is just one.

1154 [f44] - [f66] The `<AllOf>` element encloses the **conjunctive sequence** of `Match` elements. In this  
1155 example, there are two.

1156 [f45] - [f53] The first `<Match>` element compares its first and second child elements according to the  
1157 matching function. A match is positive if the value of the first argument matches any of the values  
1158 selected by the second argument. This match compares the **target** namespace of the requested  
1159 document with the value of “urn:example:med:schemas:record”.

1160 [f45] The `MatchId` attribute names the matching function.

1161 [f46] - [f47] Literal **attribute** value to match.

1162 [f48] - [f52] The `<AttributeDesignator>` element selects the **target** namespace from the **resource**  
1163 contained in the request **context**. The **attribute** name is specified by the `AttributeId`.

1164 [f54] - [f65] The second `<Match>` element. This match compares the results of two XPath expressions  
1165 applied to the `<Content>` element of the **resource** category. The second XPath expression is the  
1166 location path to the requested XML element and the first XPath expression is the literal value “md:record”.  
1167 The “xpath-node-match” function evaluates to “True” if the requested XML element is below the  
1168 “md:record” element.

1169 [f68] - [f82] The `<AnyOf>` element contains a **disjunctive sequence** of `<AllOf>` elements. In this case,  
1170 there is just one `<AllOf>` element.

1171 [f69] - [f81] The `<AllOf>` element contains a **conjunctive sequence** of `<Match>` elements. In this case,  
1172 there is just one `<Match>` element.

1173 [f70] - [f80] The <Match> element compares its first and second child elements according to the matching  
1174 function. The match is positive if the value of the first argument matches any of the values selected by  
1175 the second argument. In this case, the value of the action-id **action attribute** in the request **context** is  
1176 compared with the literal value "read".

1177 [f84] - [f86] The <Condition> element. A **condition** must evaluate to "True" for the **rule** to be  
1178 applicable. This **condition** contains a reference to a variable definition defined elsewhere in the **policy**.

#### 1179 4.2.4.2 Rule 2

1180 **Rule 2** illustrates the use of a mathematical function, i.e. the <Apply> element with functionId  
1181 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate the date of the patient's  
1182 sixteenth birthday. It also illustrates the use of **predicate** expressions, with the functionId  
1183 "urn:oasis:names:tc:xacml:1.0:function:and". This example has one function embedded in the  
1184 <Condition> element and another one referenced in a <VariableDefinition> element.

```
1185 [g1] <?xml version="1.0" encoding="UTF-8"?>
1186 [g2] <Policy
1187 [g3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1188 [g4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1189 [g5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1190 [g6]   xmlns:xf="http://www.w3.org/2005/xpath-functions"
1191 [g7]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1192 [g8]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"
1193 [g9]   Version="1.0"
1194 [g10]  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1195      algorithm:deny-overrides">
1196 [g11]  <PolicyDefaults>
1197 [g12]    <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1198 [g13]  </PolicyDefaults>
1199 [g14]  <Target/>
1200 [g15]  <VariableDefinition VariableId="17590035">
1201 [g16]    <Apply
1202 [g17]      FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal">
1203 [g18]        <Apply
1204 [g19]          FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
1205 [g20]            <AttributeDesignator
1206 [g21]              MustBePresent="false"
1207 [g22]              Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
1208 [g23]              AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
1209 [g24]              DataType="http://www.w3.org/2001/XMLSchema#date"/>
1210 [g25]            </Apply>
1211 [g26]          <Apply
1212 [g27]            FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
1213 [g28]              <Apply
1214 [g29]                FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
1215 [g30]                  <AttributeSelector
1216 [g31]                    MustBePresent="false"
1217 [g32]                    Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1218 [g33]                    Path="md:record/md:patient/md:patientDoB/text()"
1219 [g34]                    DataType="http://www.w3.org/2001/XMLSchema#date"/>
1220 [g35]                  </Apply>
1221 [g36]                <AttributeValue
1222 [g37]                  DataType="http://www.w3.org/2001/XMLSchema#yearMonthDuration"
1223 [g38]                  >P16Y</AttributeValue>
1224 [g39]                </Apply>
1225 [g40]              </Apply>
1226 [g41]            </VariableDefinition>
1227 [g42]    <Rule
1228 [g43]      RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
1229 [g44]      Effect="Permit">
1230 [g45]      <Description>
1231 [g46]        A person may read any medical record in the
1232 [g47]        http://www.med.example.com/records.xsd namespace
1233 [g48]        for which he or she is the designated parent or guardian,
1234 [g49]        and for which the patient is under 16 years of age
1235 [g50]      </Description>
1236 [g51]      <Target>
1237 [g52]        <AnyOf>
1238 [g53]          <AllOf>
```

```

1239 [g54] <Match
1240 [g55]     MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1241 [g56]     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1242 [g57]     >urn:example:med:schemas:record</AttributeValue>
1243 [g58]     <AttributeDesignator
1244 [g59]     MustBePresent="false"
1245 [g60]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1246 [g61]     AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1247 [g62]     DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1248 [g63] </Match>
1249 [g64] <Match
1250 [g65]     MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1251 [g66]     <AttributeValue
1252 [g67]     DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1253 [g68]     XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1254 [g69]     >md:record</AttributeValue>
1255 [g70]     <AttributeDesignator
1256 [g71]     MustBePresent="false"
1257 [g72]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1258 [g73]     AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1259 [g74]     DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1260 [g75] </Match>
1261 [g76] </AllOf>
1262 [g77] </AnyOf>
1263 [g78] <AnyOf>
1264 [g79]     <AllOf>
1265 [g80]     <Match
1266 [g81]     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1267 [g82]     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1268 [g83]     >read</AttributeValue>
1269 [g84]     <AttributeDesignator
1270 [g85]     MustBePresent="false"
1271 [g86]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1272 [g87]     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1273 [g88]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1274 [g89]     </Match>
1275 [g90]     </AllOf>
1276 [g91] </AnyOf>
1277 [g92] </Target>
1278 [g93] <Condition>
1279 [g94]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1280 [g95]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1281 [g96]     <Apply
1282 [g97]     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1283 [g98]     <AttributeDesignator
1284 [g99]     MustBePresent="false"
1285 [g100]     Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1286 [g101]     AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:parent-
1287 guardian-id"
1288 [g102]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1289 [g103]     </Apply>
1290 [g104]     <Apply
1291 [g105]     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1292 [g106]     <AttributeSelector
1293 [g107]     MustBePresent="false"
1294 [g108]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1295 [g109]     Path="md:record/md:parentGuardian/md:parentGuardianId/text()"
1296 [g110]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1297 [g111]     </Apply>
1298 [g112]     </Apply>
1299 [g113]     <VariableReference VariableId="17590035"/>
1300 [g114]     </Apply>
1301 [g115]     </Condition>
1302 [g116] </Rule>
1303 [g117] </Policy>

```

1304 [g15] - [g41] The <VariableDefinition> element contains part of the **condition** (i.e. is the patient  
1305 under 16 years of age?). The patient is under 16 years of age if the current date is less than the date  
1306 computed by adding 16 to the patient's date of birth.

1307 [g16] - [g40] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to compare the two date  
1308 arguments.

1309 [g18] - [g25] The first date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-one-and-only” to  
 1310 ensure that the **bag** of values selected by its argument contains exactly one value of type  
 1311 “http://www.w3.org/2001/XMLSchema#date”.

1312 [g20] The current date is evaluated by selecting the “urn:oasis:names:tc:xacml:1.0:environment:current-  
 1313 date” **environment attribute**.

1314 [g26] - [g39] The second date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-add-  
 1315 yearMonthDuration” to compute the date of the patient’s sixteenth birthday by adding 16 years to the  
 1316 patient’s date of birth. The first of its arguments is of type “http://www.w3.org/2001/XMLSchema#date”  
 1317 and the second is of type “http://www.w3.org/TR/2007/REC-xpath-functions-20070123/#dt-  
 1318 yearMonthDuration”.

1319 [g30] The <AttributeSelector> element selects the patient’s date of birth by taking the XPath  
 1320 expression over the **resource** content.

1321 [g36] - [g38] Year Month Duration of 16 years.

1322 [g51] - [g92] **Rule** declaration and **rule target**. See **Rule** 1 in Section 4.2.4.1 for the detailed explanation  
 1323 of these elements.

1324 [g93] - [g115] The <Condition> element. The **condition** must evaluate to “True” for the **rule** to be  
 1325 applicable. This **condition** evaluates the truth of the statement: the requestor is the designated parent or  
 1326 guardian and the patient is under 16 years of age. It contains one embedded <Apply> element and one  
 1327 referenced <VariableDefinition> element.

1328 [g94] The **condition** uses the “urn:oasis:names:tc:xacml:1.0:function:and” function. This is a Boolean  
 1329 function that takes one or more Boolean arguments (2 in this case) and performs the logical “AND”  
 1330 operation to compute the truth value of the expression.

1331 [g95] - [g112] The first part of the **condition** is evaluated (i.e. is the requestor the designated parent or  
 1332 guardian?). The function is “urn:oasis:names:tc:xacml:1.0:function:string-equal” and it takes two  
 1333 arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1334 [g96] designates the first argument. Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes  
 1335 arguments of type “http://www.w3.org/2001/XMLSchema#string”,  
 1336 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that the **subject attribute**  
 1337 “urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id” in the request **context** contains  
 1338 exactly one value.

1339 [g98] designates the first argument. The value of the **subject attribute**  
 1340 “urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id” is selected from the request **context**  
 1341 using the <AttributeDesignator> element.

1342 [g104] As above, the “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that  
 1343 the **bag** of values selected by its argument contains exactly one value of type  
 1344 “http://www.w3.org/2001/XMLSchema#string”.

1345 [g106] The second argument selects the value of the <md:parentGuardianId> element from the  
 1346 **resource** content using the <AttributeSelector> element. This element contains a free-form XPath  
 1347 expression, pointing into the <Content> element of the resource category. Note that all namespace  
 1348 prefixes in the XPath expression are resolved with standard namespace declarations. The  
 1349 AttributeSelector evaluates to the **bag** of values of type  
 1350 “http://www.w3.org/2001/XMLSchema#string”.

1351 [g113] references the <VariableDefinition> element, where the second part of the **condition** is  
 1352 defined.

### 1353 4.2.4.3 Rule 3

1354 **Rule 3** illustrates the use of an **obligation** expression.

```

1355 [h1] <?xml version="1.0" encoding="UTF-8"?>
1356 [h2] <Policy
1357 [h3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1358 [h4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1359 [h5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

```

```

1360 [h6] xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
1361 http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
1362 [h7] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1363 [h8] PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:3"
1364 [h9] Version="1.0"
1365 [h10] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1366 algorithm:deny-overrides">
1367 [h11] <Description>
1368 [h12] Policy for any medical record in the
1369 [h13] http://www.med.example.com/schemas/record.xsd namespace
1370 [h14] </Description>
1371 [h15] <PolicyDefaults>
1372 [h16] <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1373 [h17] </PolicyDefaults>
1374 [h18] <Target>
1375 [h19] <AnyOf>
1376 [h20] <AllOf>
1377 [h21] <Match
1378 [h22] MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1379 [h23] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1380 [h24] >urn:example:med:schemas:record</AttributeValue>
1381 [h25] <AttributeDesignator
1382 [h26] MustBePresent="false"
1383 [h27] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1384 [h28] AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1385 [h29] DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1386 [h30] </Match>
1387 [h31] </AllOf>
1388 [h32] </AnyOf>
1389 [h33] </Target>
1390 [h34] <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:3"
1391 [h35] Effect="Permit">
1392 [h36] <Description>
1393 [h37] A physician may write any medical element in a record
1394 [h38] for which he or she is the designated primary care
1395 [h39] physician, provided an email is sent to the patient
1396 [h40] </Description>
1397 [h41] <Target>
1398 [h42] <AnyOf>
1399 [h43] <AllOf>
1400 [h44] <Match
1401 [h45] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1402 [h46] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1403 [h47] >physician</AttributeValue>
1404 [h48] <AttributeDesignator
1405 [h49] MustBePresent="false"
1406 [h50] Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1407 [h51] AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
1408 [h52] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1409 [h53] </Match>
1410 [h54] </AllOf>
1411 [h55] </AnyOf>
1412 [h56] <AnyOf>
1413 [h57] <AllOf>
1414 [h58] <Match
1415 [h59] MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1416 [h60] <AttributeValue
1417 [h61] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1418 [h62] XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1419 [h63] >md:record/md:medical</AttributeValue>
1420 [h64] <AttributeDesignator
1421 [h65] MustBePresent="false"
1422 [h66] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1423 [h67] AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1424 [h68] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1425 [h69] </Match>
1426 [h70] </AllOf>
1427 [h71] </AnyOf>
1428 [h72] <AnyOf>
1429 [h73] <AllOf>
1430 [h74] <Match
1431 [h75] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1432 [h76] <AttributeValue

```

```

1433 [h77]         DataType="http://www.w3.org/2001/XMLSchema#string"
1434 [h78]         >write</AttributeValue>
1435 [h79]         <AttributeDesignator
1436 [h80]             MustBePresent="false"
1437 [h81]             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1438 [h82]             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1439 [h83]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1440 [h84]         </Match>
1441 [h85]     </AllOf>
1442 [h86] </AnyOf>
1443 [h87] </Target>
1444 [h88] <Condition>
1445 [h89]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1446 [h90]         <Apply
1447 [h91]             FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1448 [h92]             <AttributeDesignator
1449 [h93]                 MustBePresent="false"
1450 [h94]                 Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1451 [h95]                 AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id"
1452 [h96]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1453 [h97]             </Apply>
1454 [h98]         <Apply
1455 [h99]             FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1456 [h100]            <AttributeSelector
1457 [h101]                MustBePresent="false"
1458 [h102]                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1459 [h103]                Path="md:record/md:primaryCarePhysician/md:registrationID/text()"
1460 [h104]                DataType="http://www.w3.org/2001/XMLSchema#string"/>
1461 [h105]            </Apply>
1462 [h106]        </Apply>
1463 [h107]    </Condition>
1464 [h108] </Rule>
1465 [h109] <ObligationExpressions>
1466 [h110]     <ObligationExpression
1467 [h111]         ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
1468 [h112]         FulfillOn="Permit">
1469 [h113]         <AttributeAssignmentExpression
1470 [h114]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:mailto">
1471 [h115]             <AttributeSelector
1472 [h116]                 MustBePresent="true"
1473 [h117]                 Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1474 [h118]                 Path="md:record/md:patient/md:patientContact/md:email"
1475 [h119]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1476 [h120]             </AttributeAssignmentExpression>
1477 [h121]         <AttributeAssignmentExpression
1478 [h122]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
1479 [h123]             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1480 [h124]                 >Your medical record has been accessed by:</AttributeValue>
1481 [h125]             </AttributeAssignmentExpression>
1482 [h126]         <AttributeAssignmentExpression
1483 [h127]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
1484 [h128]             <AttributeDesignator
1485 [h129]                 MustBePresent="false"
1486 [h130]                 Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1487 [h131]                 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1488 [h132]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1489 [h133]             </AttributeAssignmentExpression>
1490 [h134]         </ObligationExpression>
1491 [h135]     </ObligationExpressions>
1492 [h136] </Policy>

```

1493 [h2] - [h10] The <Policy> element includes standard namespace declarations as well as **policy** specific  
1494 parameters, such as PolicyId and RuleCombiningAlgId.

1495 [h8] **Policy** identifier. This parameter allows the **policy** to be referenced by a **policy set**.

1496 [h10] The **Rule-combining algorithm** identifies the algorithm for combining the outcomes of **rule**  
1497 evaluation.

1498 [h11] - [h14] Free-form description of the **policy**.

1499 [h18] - [h33] **Policy target**. The **policy target** defines a set of applicable **decision requests**. The  
1500 structure of the <Target> element in the <Policy> is identical to the structure of the <Target>

1501 element in the <Rule>. In this case, the **policy target** is the set of all XML **resources** that conform to  
1502 the namespace “urn:example:med:schemas:record”.

1503 [h34] - [h108] The only <Rule> element included in this <Policy>. Two parameters are specified in the  
1504 **rule** header: RuleId and Effect.

1505 [h41] - [h87] The **rule target** further constrains the **policy target**.

1506 [h44] - [h53] The <Match> element targets the **rule** at **subjects** whose  
1507 “urn:oasis:names:tc:xacml:3.0:example:attribute:role” **subject attribute** is equal to “physician”.

1508 [h58] - [h69] The <Match> element targets the **rule** at **resources** that match the XPath expression  
1509 “md:record/md:medical”.

1510 [h74] - [h84] The <Match> element targets the **rule** at **actions** whose  
1511 “urn:oasis:names:tc:xacml:1.0:action:action-id” **action attribute** is equal to “write”.

1512 [h88] - [h107] The <Condition> element. For the **rule** to be applicable to the **decision request**, the  
1513 **condition** must evaluate to “True”. This **condition** compares the value of the  
1514 “urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id” **subject attribute** with the value of the  
1515 <registrationId> element in the medical record that is being accessed.

1516 [h109] - [h134] The <ObligationExpressions> element. **Obligations** are a set of operations that  
1517 must be performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be  
1518 associated with a “Permit” or “Deny” **authorization decision**. The element contains a single **obligation**  
1519 expression, which will be evaluated into an obligation when the policy is evaluated.

1520 [h110] - [h133] The <ObligationExpression> element consists of the ObligationId attribute, the  
1521 **authorization decision** value for which it must be fulfilled, and a set of **attribute** assignments.

1522 [h110] The ObligationId attribute identifies the **obligation**. In this case, the **PEP** is required to send  
1523 email.

1524 [h111] The FulfillOn attribute defines the **authorization decision** value for which the **obligation**  
1525 derived from the **obligation** expression must be fulfilled. In this case, the **obligation** must be fulfilled  
1526 when **access** is permitted.

1527 [h112] - [h119] The first parameter indicates where the **PEP** will find the email address in the **resource**.  
1528 The **PDP** will evaluate the <AttributeSelector> and return the result to the **PEP** inside the resulting  
1529 **obligation**.

1530 [h120] - [h123] The second parameter contains literal text for the email body.

1531 [h125] - [h132] The third parameter indicates where the **PEP** will find further text for the email body in the  
1532 **resource**. The **PDP** will evaluate the <AttributeDesignator> and return the result to the **PEP** inside  
1533 the resulting **obligation**.

#### 1534 4.2.4.4 Rule 4

1535 **Rule 4** illustrates the use of the “Deny” **Effect** value, and a <Rule> with no <Condition> element.

```

1536 [i1] <?xml version="1.0" encoding="UTF-8"?>
1537 [i2] <Policy
1538 [i3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1539 [i4]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1540 [i5]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1541 [i6]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:4"
1542 [i7]   Version="1.0"
1543 [i8]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1544 [i8]   algorithm:deny-overrides">
1545 [i9]   <PolicyDefaults>
1546 [i10]     <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1547 [i11]   </PolicyDefaults>
1548 [i12]   <Target/>
1549 [i13]   <Rule
1550 [i14]     RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:4"
1551 [i15]     Effect="Deny">
1552 [i16]     <Description>
1553 [i17]       An Administrator shall not be permitted to read or write

```

```

1554 [i18]     medical elements of a patient record in the
1555 [i19]     http://www.med.example.com/records.xsd namespace.
1556 [i20] </Description>
1557 [i21] <Target>
1558 [i22]   <AnyOf>
1559 [i23]     <AllOf>
1560 [i24]       <Match
1561 [i25]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1562 [i26]         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1563 [i27]         >administrator</AttributeValue>
1564 [i28]         <AttributeDesignator
1565 [i29]           MustBePresent="false"
1566 [i30]         Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1567 [i31]         AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
1568 [i32]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1569 [i33]       </Match>
1570 [i34]     </AllOf>
1571 [i35]   </AnyOf>
1572 [i36] <AnyOf>
1573 [i37]   <AllOf>
1574 [i38]     <Match
1575 [i39]       MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1576 [i40]       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1577 [i41]       >urn:example:med:schemas:record</AttributeValue>
1578 [i42]       <AttributeDesignator
1579 [i43]         MustBePresent="false"
1580 [i44]       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1581 [i45]       AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1582 [i46]       DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1583 [i47]     </Match>
1584 [i48]     <Match
1585 [i49]       MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1586 [i50]       <AttributeValue
1587 [i51]         DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1588 [i52]       XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1589 [i53]       >md:record/md:medical</AttributeValue>
1590 [i54]       <AttributeDesignator
1591 [i55]         MustBePresent="false"
1592 [i56]       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1593 [i57]       AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1594 [i58]       DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1595 [i59]     </Match>
1596 [i60]   </AllOf>
1597 [i61] </AnyOf>
1598 [i62] <AnyOf>
1599 [i63]   <AllOf>
1600 [i64]     <Match
1601 [i65]       MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1602 [i66]       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1603 [i67]       >read</AttributeValue>
1604 [i68]       <AttributeDesignator
1605 [i69]         MustBePresent="false"
1606 [i70]       Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1607 [i71]       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1608 [i72]       DataType="http://www.w3.org/2001/XMLSchema#string"/>
1609 [i73]     </Match>
1610 [i74]   </AllOf>
1611 [i75] <AllOf>
1612 [i76]   <Match
1613 [i77]     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1614 [i78]     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1615 [i79]     >write</AttributeValue>
1616 [i80]     <AttributeDesignator
1617 [i81]       MustBePresent="false"
1618 [i82]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1619 [i83]     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1620 [i84]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1621 [i85]   </Match>
1622 [i86] </AllOf>
1623 [i87] </AnyOf>
1624 [i88] </Target>
1625 [i89] </Rule>
1626 [i90] </Policy>

```

1627 [i13] - [i15] The <Rule> element declaration.

1628 [i15] **Rule Effect**. Every **rule** that evaluates to “True” emits the **rule effect** as its value. This **rule**  
 1629 **Effect** is “Deny” meaning that according to this **rule**, **access** must be denied when it evaluates to  
 1630 “True”.

1631 [i16] - [i20] Free form description of the **rule**.

1632 [i21] - [i88] **Rule target**. The **Rule target** defines the set of **decision requests** that are applicable to the  
 1633 **rule**.

1634 [i24] - [i33] The <Match> element targets the **rule** at **subjects** whose  
 1635 “urn:oasis:names:tc:xacml:3.0:example:attribute:role” **subject attribute** is equal to “administrator”.

1636 [i36] - [i61] The <AnyOf> element contains one <AllOf> element, which (in turn) contains two <Match>  
 1637 elements. The **target** matches if the **resource** identified by the request **context** matches both **resource**  
 1638 match criteria.

1639 [i38] - [i47] The first <Match> element targets the **rule** at **resources** whose  
 1640 “urn:oasis:names:tc:xacml:2.0:resource:target-namespace” **resource attribute** is equal to  
 1641 “urn:example:med:schemas:record”.

1642 [i48] - [i59] The second <Match> element targets the **rule** at XML elements that match the XPath  
 1643 expression “/md:record/md:medical”.

1644 [i62] - [i87] The <AnyOf> element contains two <AllOf> elements, each of which contains one <Match>  
 1645 element. The **target** matches if the **action** identified in the request **context** matches either of the **action**  
 1646 match criteria.

1647 [i64] - [i85] The <Match> elements **target** the **rule** at **actions** whose  
 1648 “urn:oasis:names:tc:xacml:1.0:action:action-id” **action attribute** is equal to “read” or “write”.

1649 This **rule** does not have a <Condition> element.

#### 1650 4.2.4.5 Example PolicySet

1651 This section uses the examples of the previous sections to illustrate the process of combining **policies**.  
 1652 The **policy** governing read **access** to medical elements of a record is formed from each of the four **rules**  
 1653 described in Section 4.2.3. In plain language, the combined **rule** is:

- 1654 • Either the requestor is the patient; or
- 1655 • the requestor is the parent or guardian and the patient is under 16; or
- 1656 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1657 • the requestor is not an administrator.

1658 The following **policy set** illustrates the combined **policies**. **Policy 3** is included by reference and **policy**  
 1659 **2** is explicitly included.

```

1660 [j1]    <?xml version="1.0" encoding="UTF-8"?>
1661 [j2]    <PolicySet
1662 [j3]      xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1663 [j4]      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1664 [j5]      PolicySetId="urn:oasis:names:tc:xacml:3.0:example:policysetid:1"
1665 [j6]      Version="1.0"
1666 [j7]      PolicyCombiningAlgId=
1667 [j8]      "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
1668 [j9]    <Description>
1669 [j10]      Example policy set.
1670 [j11]    </Description>
1671 [j12]    <Target>
1672 [j13]      <AnyOf>
1673 [j14]        <AllOf>
1674 [j15]          <Match
1675 [j16]            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1676 [j17]            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1677 [j18]              >urn:example:med:schema:records</AttributeValue>
1678 [j19]            <AttributeDesignator
1679 [j20]              MustBePresent="false"

```

```

1680 [j21]         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1681 [j22]         AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1682 [j23]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1683 [j24]         </Match>
1684 [j25]         </AllOf>
1685 [j26]         </AnyOf>
1686 [j27]         </Target>
1687 [j28]         <PolicyIdReference>
1688 [j29]           urn:oasis:names:tc:xacml:3.0:example:policyid:3
1689 [j30]         </PolicyIdReference>
1690 [j31]         <Policy
1691 [j32]           PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"
1692 [j33]           RuleCombiningAlgId=
1693 [j34]             "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
1694 [j35]           Version="1.0">
1695 [j36]           <Target/>
1696 [j37]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"
1697 [j38]             Effect="Permit">
1698 [j39]           </Rule>
1699 [j40]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
1700 [j41]             Effect="Permit">
1701 [j42]           </Rule>
1702 [j43]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:4"
1703 [j44]             Effect="Deny">
1704 [j45]           </Rule>
1705 [j46]         </Policy>
1706 [j47]         </PolicySet>

```

1707 [j2] - [j8] The <PolicySet> element declaration. Standard XML namespace declarations are included.

1708 [j5] The PolicySetId attribute is used for identifying this **policy set** for possible inclusion in another  
1709 **policy set**.

1710 [j7] - [j8] The **policy-combining algorithm** identifier. **Policies** and **policy sets** in this **policy set** are  
1711 combined according to the specified **policy-combining algorithm** when the **authorization decision** is  
1712 computed.

1713 [j9] - [j11] Free form description of the **policy set**.

1714 [j12] - [j27] The **policy set** <Target> element defines the set of **decision requests** that are applicable to  
1715 this <PolicySet> element.

1716 [j28] - [j30] PolicyIdReference includes a **policy** by id.

1717 [j31] - [j46] **Policy 2** is explicitly included in this **policy set**. The **rules** in **Policy 2** are omitted for clarity.

---

## 5 Syntax (normative, with the exception of the schema fragments)

### 5.1 Element <PolicySet>

The <PolicySet> element is a top-level element in the XACML *policy* schema. <PolicySet> is an aggregation of other *policy sets* and *policies*. *Policy sets* MAY be included in an enclosing <PolicySet> element either directly using the <PolicySet> element or indirectly using the <PolicySetIdReference> element. *Policies* MAY be included in an enclosing <PolicySet> element either directly using the <Policy> element or indirectly using the <PolicyIdReference> element.

A <PolicySet> element may be evaluated, in which case the evaluation procedure defined in Section 7.13 SHALL be used.

If a <PolicySet> element contains references to other *policy sets* or *policies* in the form of URLs, then these references MAY be resolvable.

*Policy sets* and *policies* included in a <PolicySet> element MUST be combined using the algorithm identified by the PolicyCombiningAlgId attribute. <PolicySet> is treated exactly like a <Policy> in all *policy-combining algorithms*.

A <PolicySet> element MAY contain a <PolicyIssuer> element. The interpretation of the <PolicyIssuer> element is explained in the separate administrative *policy* profile [XACMLAdmin].

The <Target> element defines the applicability of the <PolicySet> element to a set of *decision requests*. If the <Target> element within the <PolicySet> element matches the request *context*, then the <PolicySet> element MAY be used by the *PDP* in making its *authorization decision*. See Section 7.13.

The <ObligationExpressions> element contains a set of *obligation* expressions that MUST be evaluated into *obligations* by the *PDP* and the resulting *obligations* MUST be fulfilled by the *PEP* in conjunction with the *authorization decision*. If the *PEP* does not understand or cannot fulfill any of the *obligations*, then it MUST act according to the PEP bias. See Section 7.2 and 7.18.

The <AdviceExpressions> element contains a set of *advice* expressions that MUST be evaluated into *advice* by the *PDP*. The resulting *advice* MAY be safely ignored by the *PEP* in conjunction with the *authorization decision*. See Section 7.18.

```
<xs:element name="PolicySet" type="xacml:PolicySetType"/>
<xs:complexType name="PolicySetType">
  <xs:sequence>
    <xs:element ref="xacml:Description" minOccurs="0"/>
    <xs:element ref="xacml:PolicyIssuer" minOccurs="0"/>
    <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>
    <xs:element ref="xacml:Target"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xacml:PolicySet"/>
      <xs:element ref="xacml:Policy"/>
      <xs:element ref="xacml:PolicySetIdReference"/>
      <xs:element ref="xacml:PolicyIdReference"/>
      <xs:element ref="xacml:CombinerParameters"/>
      <xs:element ref="xacml:PolicyCombinerParameters"/>
      <xs:element ref="xacml:PolicySetCombinerParameters"/>
    </xs:choice>
    <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
    <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
  </xs:sequence>
```

```

1767 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>
1768 <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
1769 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>
1770 <xs:attribute name="MaxDelegationDepth" type="xs:integer" use="optional"/>
1771 </xs:complexType>

```

1772 The <PolicySet> element is of PolicySetType complex type.

1773 The <PolicySet> element contains the following attributes and elements:

1774 PolicySetId [Required]

1775 **Policy set** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to  
 1776 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or URI  
 1777 scheme. If the **policy set** identifier is in the form of a URL, then it MAY be resolvable.

1778 Version [Required]

1779 The version number of the PolicySet.

1780 PolicyCombiningAlgId [Required]

1781 The identifier of the **policy-combining algorithm** by which the <PolicySet>,  
 1782 <CombinerParameters>, <PolicyCombinerParameters> and  
 1783 <PolicySetCombinerParameters> components MUST be combined. Standard **policy-**  
 1784 **combining algorithms** are listed in Appendix Appendix C. Standard **policy-combining**  
 1785 **algorithm** identifiers are listed in Section B.9.

1786 MaxDelegationDepth [Optional]

1787 If present, limits the depth of delegation which is authorized by this **policy set**. See the delegation  
 1788 profile [XACMLAdmin].

1789 <Description> [Optional]

1790 A free-form description of the **policy set**.

1791 <PolicyIssuer> [Optional]

1792 **Attributes** of the **issuer** of the **policy set**.

1793 <PolicySetDefaults> [Optional]

1794 A set of default values applicable to the **policy set**. The scope of the <PolicySetDefaults>  
 1795 element SHALL be the enclosing **policy set**.

1796 <Target> [Required]

1797 The <Target> element defines the applicability of a **policy set** to a set of **decision requests**.

1798 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be computed  
 1799 from the <Target> elements of the referenced <Policy> elements, either as an intersection or  
 1800 as a union.

1801 <PolicySet> [Any Number]

1802 A **policy set** that is included in this **policy set**.

1803 <Policy> [Any Number]

1804 A **policy** that is included in this **policy set**.

1805 <PolicySetIdReference> [Any Number]

1806 A reference to a **policy set** that MUST be included in this **policy set**. If  
 1807 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1808 <PolicyIdReference> [Any Number]

1809 A reference to a **policy** that MUST be included in this **policy set**. If the  
 1810 <PolicyIdReference> is a URL, then it MAY be resolvable.

- 1811 <ObligationExpressions> [Optional]  
 1812       Contains the set of <ObligationExpression> elements. See Section 7.18 for a description of  
 1813       how the set of **obligations** to be returned by the **PDP** shall be determined.
- 1814 <AdviceExpressions> [Optional]  
 1815       Contains the set of <AdviceExpression> elements. See Section 7.18 for a description of how  
 1816       the set of **advice** to be returned by the **PDP** shall be determined.
- 1817 <CombinerParameters> [Optional]  
 1818       Contains a sequence of <CombinerParameter> elements. The parameters apply to the  
 1819       combining algorithm as such and it is up to the specific combining algorithm to interpret them and  
 1820       adjust its behavior accordingly.
- 1821 <PolicyCombinerParameters> [Optional]  
 1822       Contains a sequence of <CombinerParameter> elements that are associated with a particular  
 1823       <Policy> or <PolicyIdReference> element within the <PolicySet>. It is up to the specific  
 1824       combining algorithm to interpret them and adjust its behavior accordingly.
- 1825 <PolicySetCombinerParameters> [Optional]  
 1826       Contains a sequence of <CombinerParameter> elements that are associated with a particular  
 1827       <PolicySet> or <PolicySetIdReference> element within the <PolicySet>. It is up to the  
 1828       specific combining algorithm to interpret them and adjust its behavior accordingly.

## 1829 5.2 Element <Description>

- 1830 The <Description> element contains a free-form description of the <PolicySet>, <Policy>,  
 1831 <Rule> or <Apply> element. The <Description> element is of xs:string simple type.

```
1832 <xs:element name="Description" type="xs:string"/>
```

## 1833 5.3 Element <PolicyIssuer>

- 1834 The <PolicyIssuer> element contains **attributes** describing the issuer of the **policy** or **policy set**.  
 1835 The use of the **policy** issuer element is defined in a separate administration profile [**XACMLAdmin**]. A  
 1836 PDP which does not implement the administration profile MUST report an error or return an Indeterminate  
 1837 result if it encounters this element.

```
1838 <xs:element name="PolicyIssuer" type="xacml:PolicyIssuerType"/>
1839 <xs:complexType name="PolicyIssuerType">
1840   <xs:sequence>
1841     <xs:element ref="xacml:Content" minOccurs="0"/>
1842     <xs:element ref="xacml:Attribute" minOccurs="0" maxOccurs="unbounded"/>
1843   </xs:sequence>
1844 </xs:complexType>
```

- 1845 The <PolicyIssuer> element is of PolicyIssuerType complex type.

- 1846 The <PolicyIssuer> element contains the following elements:

- 1847 <Content> [Optional]  
 1848       Free form XML describing the issuer. See Section 5.45.
- 1849 <Attribute> [Zero to many]  
 1850       An **attribute** of the issuer. See Section 5.46.

## 1851 5.4 Element <PolicySetDefaults>

- 1852 The <PolicySetDefaults> element SHALL specify default values that apply to the <PolicySet>  
 1853 element.

```

1854 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
1855 <xs:complexType name="DefaultsType">
1856   <xs:sequence>
1857     <xs:choice>
1858       <xs:element ref="xacml:XPathVersion">
1859     </xs:choice>
1860   </xs:sequence>
1861 </xs:complexType>

```

1862 <PolicySetDefaults> element is of DefaultsType complex type.

1863 The <PolicySetDefaults> element contains the following elements:

1864 <XPathVersion> [Optional]

1865       Default XPath version.

## 1866 5.5 Element <XPathVersion>

1867 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by  
1868 <AttributeSelector> elements and XPath-based functions in the **policy set** or **policy**.

```

1869 <xs:element name="XPathVersion" type="xs:anyURI"/>

```

1870 The URI for the XPath 1.0 specification is "http://www.w3.org/TR/1999/REC-xpath-19991116".

1871 The URI for the XPath 2.0 specification is "http://www.w3.org/TR/2007/REC-xpath20-20070123".

1872 The <XPathVersion> element is REQUIRED if the XACML enclosing **policy set** or **policy** contains  
1873 <AttributeSelector> elements or XPath-based functions.

## 1874 5.6 Element <Target>

1875 The <Target> element identifies the set of **decision requests** that the parent element is intended to  
1876 evaluate. The <Target> element SHALL appear as a child of a <PolicySet> and <Policy> element  
1877 and MAY appear as a child of a <Rule> element.

1878 The <Target> element SHALL contain a **conjunctive sequence** of <AnyOf> elements. For the parent  
1879 of the <Target> element to be applicable to the **decision request**, there MUST be at least one positive  
1880 match between each <AnyOf> element of the <Target> element and the corresponding section of the  
1881 <Request> element.

```

1882 <xs:element name="Target" type="xacml:TargetType"/>
1883 <xs:complexType name="TargetType">
1884   <xs:sequence minOccurs="0" maxOccurs="unbounded">
1885     <xs:element ref="xacml:AnyOf"/>
1886   </xs:sequence>
1887 </xs:complexType>

```

1888 The <Target> element is of TargetType complex type.

1889 The <Target> element contains the following elements:

1890 <AnyOf> [Zero to Many]

1891       Matching specification for **attributes** in the **context**. If this element is missing, then the **target**  
1892       SHALL match all **contexts**.

## 1893 5.7 Element <AnyOf>

1894 The <AnyOf> element SHALL contain a **disjunctive sequence** of <AllOf> elements.

```

1895 <xs:element name="AnyOf" type="xacml:AnyOfType"/>
1896 <xs:complexType name="AnyOfType">
1897   <xs:sequence minOccurs="1" maxOccurs="unbounded">
1898     <xs:element ref="xacml:AllOf"/>

```

1899       </xs:sequence>  
1900       </xs:complexType>

1901       The <AnyOf> element is of AnyOfType complex type.

1902       The <AnyOf> element contains the following elements:

1903       <AllOf> [One to Many, Required]

1904           See Section 5.8.

## 1905       5.8 Element <AllOf>

1906       The <AllOf> element SHALL contain a **conjunctive sequence** of <Match> elements.

```
1907       <xs:element name="AllOf" type="xacml:AllOfType"/>  
1908       <xs:complexType name="AllOfType">  
1909         <xs:sequence minOccurs="1" maxOccurs="unbounded">  
1910         <xs:element ref="xacml:Match"/>  
1911       </xs:sequence>  
1912       </xs:complexType>
```

1913       The <AllOf> element is of AllOfType complex type.

1914       The <AllOf> element contains the following elements:

1915       <Match> [One to Many]

1916           A **conjunctive sequence** of individual matches of the **attributes** in the request **context** and the  
1917           embedded **attribute** values. See Section 5.9.

## 1918       5.9 Element <Match>

1919       The <Match> element SHALL identify a set of entities by matching **attribute** values in an

1920       <Attributes> element of the request **context** with the embedded **attribute** value.

```
1921       <xs:element name="Match" type="xacml:MatchType"/>  
1922       <xs:complexType name="MatchType">  
1923         <xs:sequence>  
1924         <xs:element ref="xacml:AttributeValue"/>  
1925         <xs:choice>  
1926         <xs:element ref="xacml:AttributeDesignator"/>  
1927         <xs:element ref="xacml:AttributeSelector"/>  
1928       </xs:choice>  
1929       </xs:sequence>  
1930       <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>  
1931       </xs:complexType>
```

1932       The <Match> element is of MatchType complex type.

1933       The <Match> element contains the following attributes and elements:

1934       MatchId [Required]

1935           Specifies a matching function. The value of this attribute MUST be of type xs:anyURI with legal  
1936           values documented in Section 7.6.

1937       <AttributeValue> [Required]

1938           Embedded **attribute** value.

1939       <AttributeDesignator> [Required choice]

1940           MAY be used to identify one or more **attribute** values in an <Attributes> element of the  
1941           request **context**.

1942       <AttributeSelector> [Required choice]

1943 MAY be used to identify one or more **attribute** values in a <Content> element of the request  
1944 **context**.

## 1945 5.10 Element <PolicySetIdReference>

1946 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element by id.  
1947 If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet> element.  
1948 However, the mechanism for resolving a **policy set** reference to the corresponding **policy set** is outside  
1949 the scope of this specification.

```
1950 <xs:element name="PolicySetIdReference" type="xacml:IdReferenceType"/>  
1951 <xs:complexType name="IdReferenceType">  
1952   <xs:simpleContent>  
1953     <xs:extension base="xs:anyURI">  
1954       <xs:attribute name="xacml:Version"  
1955         type="xacml:VersionMatchType" use="optional"/>  
1956       <xs:attribute name="xacml:EarliestVersion"  
1957         type="xacml:VersionMatchType" use="optional"/>  
1958       <xs:attribute name="xacml:LatestVersion"  
1959         type="xacml:VersionMatchType" use="optional"/>  
1960     </xs:extension>  
1961   </xs:simpleContent>  
1962 </xs:complexType>
```

1963 Element <PolicySetIdReference> is of `xacml:IdReferenceType` complex type.

1964 `IdReferenceType` extends the `xs:anyURI` type with the following attributes:

1965 **Version** [Optional]

1966 Specifies a matching expression for the version of the **policy set** referenced.

1967 **EarliestVersion** [Optional]

1968 Specifies a matching expression for the earliest acceptable version of the **policy set** referenced.

1969 **LatestVersion** [Optional]

1970 Specifies a matching expression for the latest acceptable version of the **policy set** referenced.

1971 The matching operation is defined in Section 5.13. Any combination of these attributes MAY be present  
1972 in a <PolicySetIdReference>. The referenced **policy set** MUST match all expressions. If none of  
1973 these attributes is present, then any version of the **policy set** is acceptable. In the case that more than  
1974 one matching version can be obtained, then the most recent one SHOULD be used.

## 1975 5.11 Element <PolicyIdReference>

1976 The <PolicyIdReference> element SHALL be used to reference a <Policy> element by id. If  
1977 <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy> element. However, the  
1978 mechanism for resolving a **policy** reference to the corresponding **policy** is outside the scope of this  
1979 specification.

```
1980 <xs:element name="PolicyIdReference" type="xacml:IdReferenceType"/>
```

1981 Element <PolicyIdReference> is of `xacml:IdReferenceType` complex type (see Section 5.10) .

## 1982 5.12 Simple type VersionType

1983 Elements of this type SHALL contain the version number of the **policy** or **policy set**.

```
1984 <xs:simpleType name="VersionType">  
1985   <xs:restriction base="xs:string">  
1986     <xs:pattern value="(\d+\.)*\d+"/>  
1987   </xs:restriction>  
1988 </xs:simpleType>
```

1989 The version number is expressed as a sequence of decimal numbers, each separated by a period (.).  
1990 'd+' represents a sequence of one or more decimal digits.

### 1991 5.13 Simple type VersionMatchType

1992 Elements of this type SHALL contain a restricted regular expression matching a version number (see  
1993 Section 5.12). The expression SHALL match versions of a referenced **policy** or **policy set** that are  
1994 acceptable for inclusion in the referencing **policy** or **policy set**.

```
1995 <xs:simpleType name="VersionMatchType">  
1996   <xs:restriction base="xs:string">  
1997     <xs:pattern value="((\d+|\*)\.)*(\d+|\*|\+)" />  
1998   </xs:restriction>  
1999 </xs:simpleType>
```

2000 A version match is '.'-separated, like a version string. A number represents a direct numeric match. A '\*'  
2001 means that any single number is valid. A '+' means that any number, and any subsequent numbers, are  
2002 valid. In this manner, the following four patterns would all match the version string '1.2.3': '1.2.3', '1.\*.3',  
2003 '1.2.\*' and '1.+'

### 2004 5.14 Element <Policy>

2005 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2006 A <Policy> element may be evaluated, in which case the evaluation procedure defined in Section 7.12  
2007 SHALL be used.

2008 The main components of this element are the <Target>, <Rule>, <CombinerParameters>,  
2009 <RuleCombinerParameters>, <ObligationExpressions> and <AdviceExpressions>  
2010 elements and the RuleCombiningAlgId attribute.

2011 A <Policy> element MAY contain a <PolicyIssuer> element. The interpretation of the  
2012 <PolicyIssuer> element is explained in the separate administrative **policy** profile [XACMLAdmin].

2013 The <Target> element defines the applicability of the <Policy> element to a set of **decision requests**.  
2014 If the <Target> element within the <Policy> element matches the request **context**, then the  
2015 <Policy> element MAY be used by the **PDP** in making its **authorization decision**. See Section 7.12.

2016 The <Policy> element includes a sequence of choices between <VariableDefinition> and  
2017 <Rule> elements.

2018 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the  
2019 RuleCombiningAlgId attribute.

2020 The <ObligationExpressions> element contains a set of **obligation** expressions that MUST be  
2021 evaluated into **obligations** by the **PDP** and the resulting **obligations** MUST be fulfilled by the **PEP** in  
2022 conjunction with the **authorization decision**. If the **PEP** does not understand, or cannot fulfill, any of the  
2023 **obligations**, then it MUST act according to the PEP bias. See Section 7.2 and 7.18.

2024 The <AdviceExpressions> element contains a set of **advice** expressions that MUST be evaluated into  
2025 **advice** by the **PDP**. The resulting **advice** MAY be safely ignored by the **PEP** in conjunction with the  
2026 **authorization decision**. See Section 7.18.

```
2027 <xs:element name="Policy" type="xacml:PolicyType"/>  
2028 <xs:complexType name="PolicyType">  
2029   <xs:sequence>  
2030     <xs:element ref="xacml:Description" minOccurs="0"/>  
2031     <xs:element ref="xacml:PolicyIssuer" minOccurs="0"/>  
2032     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>  
2033     <xs:element ref="xacml:Target"/>  
2034     <xs:choice maxOccurs="unbounded">  
2035       <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>  
2036       <xs:element ref="xacml:RuleCombinerParameters" minOccurs="0"/>  
2037       <xs:element ref="xacml:VariableDefinition"/>
```

```

2038         <xs:element ref="xacml:Rule"/>
2039     </xs:choice>
2040     <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
2041     <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
2042 </xs:sequence>
2043 <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2044 <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
2045 <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2046 <xs:attribute name="MaxDelegationDepth" type="xs:integer" use="optional"/>
2047 </xs:complexType>

```

2048 The <Policy> element is of PolicyType complex type.

2049 The <Policy> element contains the following attributes and elements:

2050 PolicyId [Required]

2051 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to the  
2052 **PDP** have the same identifier. This MAY be achieved by following a predefined URN or URI  
2053 scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2054 Version [Required]

2055 The version number of the **Policy**.

2056 RuleCombiningAlgId [Required]

2057 The identifier of the **rule-combining algorithm** by which the <Policy>,  
2058 <CombinerParameters> and <RuleCombinerParameters> components MUST be  
2059 combined. Standard **rule-combining algorithms** are listed in Appendix Appendix C. Standard  
2060 **rule-combining algorithm** identifiers are listed in Section B.9.

2061 MaxDelegationDepth [Optional]

2062 If present, limits the depth of delegation which is authorized by this **policy**. See the delegation  
2063 profile [XACMLAdmin].

2064 <Description> [Optional]

2065 A free-form description of the **policy**. See Section 5.2.

2066 <PolicyIssuer> [Optional]

2067 **Attributes** of the **issuer** of the **policy**.

2068 <PolicyDefaults> [Optional]

2069 Defines a set of default values applicable to the **policy**. The scope of the <PolicyDefaults>  
2070 element SHALL be the enclosing **policy**.

2071 <CombinerParameters> [Optional]

2072 A sequence of parameters to be used by the **rule-combining algorithm**. The parameters apply  
2073 to the combining algorithm as such and it is up to the specific combining algorithm to interpret  
2074 them and adjust its behavior accordingly.

2075 <RuleCombinerParameters> [Optional]

2076 A sequence of <RuleCombinerParameter> elements that are associated with a particular  
2077 <Rule> element within the <Policy>.. It is up to the specific combining algorithm to interpret  
2078 them and adjust its behavior accordingly.

2079 <Target> [Required]

2080 The <Target> element defines the applicability of a <Policy> to a set of **decision requests**.

2081 The <Target> element MAY be declared by the creator of the <Policy> element, or it MAY be  
2082 computed from the <Target> elements of the referenced <Rule> elements either as an  
2083 intersection or as a union.

2084 <VariableDefinition> [Any Number]  
 2085 Common function definitions that can be referenced from anywhere in a *rule* where an  
 2086 expression can be found.

2087 <Rule> [Any Number]  
 2088 A sequence of *rules* that MUST be combined according to the `RuleCombiningAlgId` attribute.  
 2089 *Rules* whose <Target> elements and conditions match the *decision request* MUST be  
 2090 considered. *Rules* whose <Target> elements or conditions do not match the *decision request*  
 2091 SHALL be ignored.

2092 <ObligationExpressions> [Optional]  
 2093 A *conjunctive sequence* of *obligation* expressions which MUST be evaluated into *obligations*  
 2094 by the PDP. The corresponding *obligations* MUST be fulfilled by the *PEP* in conjunction with  
 2095 the *authorization decision*. See Section 7.18 for a description of how the set of *obligations* to  
 2096 be returned by the *PDP* SHALL be determined. See section 7.2 about enforcement of  
 2097 *obligations*.

2098 <AdviceExpressions> [Optional]  
 2099 A *conjunctive sequence* of *advice* expressions which MUST evaluated into *advice* by the *PDP*.  
 2100 The corresponding *advice* provide supplementary information to the *PEP* in conjunction with the  
 2101 *authorization decision*. See Section 7.18 for a description of how the set of *advice* to be  
 2102 returned by the *PDP* SHALL be determined.

## 2103 5.15 Element <PolicyDefaults>

2104 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy> element.

```
2105 <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>
2106 <xs:complexType name="DefaultsType">
2107   <xs:sequence>
2108     <xs:choice>
2109       <xs:element ref="xacml:XPathVersion" />
2110     </xs:choice>
2111   </xs:sequence>
2112 </xs:complexType>
```

2113 <PolicyDefaults> element is of `DefaultsType` complex type.

2114 The <PolicyDefaults> element contains the following elements:

2115 <XPathVersion> [Optional]  
 2116 Default XPath version.

## 2117 5.16 Element <CombinerParameters>

2118 The <CombinerParameters> element conveys parameters for a *policy- or rule-combining algorithm*.

2119 If multiple <CombinerParameters> elements occur within the same *policy* or *policy set*, they SHALL  
 2120 be considered equal to one <CombinerParameters> element containing the concatenation of all the  
 2121 sequences of <CombinerParameters> contained in all the aforementioned <CombinerParameters>  
 2122 elements, such that the order of occurrence of the <CombinerParameters> elements is preserved in the  
 2123 concatenation of the <CombinerParameter> elements.

2124 Note that none of the combining algorithms specified in XACML 3.0 is parameterized.

```
2125 <xs:element name="CombinerParameters" type="xacml:CombinerParametersType"/>
2126 <xs:complexType name="CombinerParametersType">
2127   <xs:sequence>
2128     <xs:element ref="xacml:CombinerParameter" minOccurs="0"
2129       maxOccurs="unbounded"/>
2130   </xs:sequence>
```

2131 `</xs:complexType>`

2132 The `<CombinerParameters>` element is of `CombinerParametersType` complex type.

2133 The `<CombinerParameters>` element contains the following elements:

2134 `<CombinerParameter>` [Any Number]

2135 A single parameter. See Section 5.17.

2136 Support for the `<CombinerParameters>` element is optional.

## 2137 **5.17 Element `<CombinerParameter>`**

2138 The `<CombinerParameter>` element conveys a single parameter for a *policy*- or *rule-combining*  
2139 *algorithm*.

```
2140 <xs:element name="CombinerParameter" type="xacml:CombinerParameterType"/>
2141 <xs:complexType name="CombinerParameterType">
2142   <xs:sequence>
2143     <xs:element ref="xacml:AttributeValue"/>
2144   </xs:sequence>
2145   <xs:attribute name="ParameterName" type="xs:string" use="required"/>
2146 </xs:complexType>
```

2147 The `<CombinerParameter>` element is of `CombinerParameterType` complex type.

2148 The `<CombinerParameter>` element contains the following attributes:

2149 `ParameterName` [Required]

2150 The identifier of the parameter.

2151 `<AttributeValue>` [Required]

2152 The value of the parameter.

2153 Support for the `<CombinerParameter>` element is optional.

## 2154 **5.18 Element `<RuleCombinerParameters>`**

2155 The `<RuleCombinerParameters>` element conveys parameters associated with a particular *rule*  
2156 within a *policy* for a *rule-combining algorithm*.

2157 Each `<RuleCombinerParameters>` element MUST be associated with a *rule* contained within the  
2158 same *policy*. If multiple `<RuleCombinerParameters>` elements reference the same *rule*, they SHALL  
2159 be considered equal to one `<RuleCombinerParameters>` element containing the concatenation of all  
2160 the sequences of `<CombinerParameters>` contained in all the aforementioned  
2161 `<RuleCombinerParameters>` elements, such that the order of occurrence of the  
2162 `<RuleCominberParameters>` elements is preserved in the concatenation of the  
2163 `<CombinerParameter>` elements.

2164 Note that none of the *rule-combining algorithms* specified in XACML 3.0 is parameterized.

```
2165 <xs:element name="RuleCombinerParameters"
2166 type="xacml:RuleCombinerParametersType"/>
2167 <xs:complexType name="RuleCombinerParametersType">
2168   <xs:complexContent>
2169     <xs:extension base="xacml:CombinerParametersType">
2170       <xs:attribute name="RuleIdRef" type="xs:string"
2171         use="required"/>
2172     </xs:extension>
2173   </xs:complexContent>
2174 </xs:complexType>
```

2175 The `<RuleCombinerParameters>` element contains the following attribute:

2176 RuleIdRef [Required]  
2177       The identifier of the <Rule> contained in the *policy*.  
2178 Support for the <RuleCombinerParameters> element is optional, only if support for combiner  
2179 parameters is not implemented.

## 2180 **5.19 Element <PolicyCombinerParameters>**

2181 The <PolicyCombinerParameters> element conveys parameters associated with a particular *policy*  
2182 within a *policy set* for a *policy-combining algorithm*.

2183 Each <PolicyCombinerParameters> element MUST be associated with a *policy* contained within the  
2184 same *policy set*. If multiple <PolicyCombinerParameters> elements reference the same *policy*,  
2185 they SHALL be considered equal to one <PolicyCombinerParameters> element containing the  
2186 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned  
2187 <PolicyCombinerParameters> elements, such that the order of occurrence of the  
2188 <PolicyCombinerParameters> elements is preserved in the concatenation of the  
2189 <CombinerParameter> elements.

2190 Note that none of the *policy-combining algorithms* specified in XACML 3.0 is parameterized.

```
2191 <xs:element name="PolicyCombinerParameters"  
2192 type="xacml:PolicyCombinerParametersType"/>  
2193 <xs:complexType name="PolicyCombinerParametersType">  
2194   <xs:complexContent>  
2195     <xs:extension base="xacml:CombinerParametersType">  
2196       <xs:attribute name="PolicyIdRef" type="xs:anyURI"  
2197 use="required"/>  
2198     </xs:extension>  
2199   </xs:complexContent>  
2200 </xs:complexType>
```

2201 The <PolicyCombinerParameters> element is of PolicyCombinerParametersType complex  
2202 type.

2203 The <PolicyCombinerParameters> element contains the following attribute:

2204 PolicyIdRef [Required]

2205       The identifier of a <Policy> or the value of a <PolicyIdReference> contained in the *policy*  
2206       *set*.

2207 Support for the <PolicyCombinerParameters> element is optional, only if support for combiner  
2208 parameters is not implemented.

## 2209 **5.20 Element <PolicySetCombinerParameters>**

2210 The <PolicySetCombinerParameters> element conveys parameters associated with a particular  
2211 *policy set* within a *policy set* for a *policy-combining algorithm*.

2212 Each <PolicySetCombinerParameters> element MUST be associated with a *policy set* contained  
2213 within the same *policy set*. If multiple <PolicySetCombinerParameters> elements reference the  
2214 same *policy set*, they SHALL be considered equal to one <PolicySetCombinerParameters>  
2215 element containing the concatenation of all the sequences of <CombinerParameters> contained in all  
2216 the aforementioned <PolicySetCombinerParameters> elements, such that the order of occurrence  
2217 of the <PolicySetCombinerParameters> elements is preserved in the concatenation of the  
2218 <CombinerParameter> elements.

2219 Note that none of the *policy-combining algorithms* specified in XACML 3.0 is parameterized.

```
2220 <xs:element name="PolicySetCombinerParameters"  
2221 type="xacml:PolicySetCombinerParametersType"/>  
2222 <xs:complexType name="PolicySetCombinerParametersType">
```

```

2223     <xs:complexContent>
2224         <xs:extension base="xacml:CombinerParametersType">
2225             <xs:attribute name="PolicySetIdRef" type="xs:anyURI"
2226 use="required"/>
2227         </xs:extension>
2228     </xs:complexContent>
2229 </xs:complexType>

```

2230 The <PolicySetCombinerParameters> element is of PolicySetCombinerParametersType  
2231 complex type.

2232 The <PolicySetCombinerParameters> element contains the following attribute:

2233 PolicySetIdRef [Required]

2234 The identifier of a <PolicySet> or the value of a <PolicySetIdReference> contained in the  
2235 **policy set**.

2236 Support for the <PolicySetCombinerParameters> element is optional, only if support for combiner  
2237 parameters is not implemented.

## 2238 5.21 Element <Rule>

2239 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of this  
2240 element are the <Target>, <Condition>, <ObligationExpressions> and  
2241 <AdviceExpressions> elements and the Effect attribute.

2242 A <Rule> element may be evaluated, in which case the evaluation procedure defined in Section 7.10  
2243 SHALL be used.

```

2244 <xs:element name="Rule" type="xacml:RuleType"/>
2245 <xs:complexType name="RuleType">
2246     <xs:sequence>
2247         <xs:element ref="xacml:Description" minOccurs="0"/>
2248         <xs:element ref="xacml:Target" minOccurs="0"/>
2249         <xs:element ref="xacml:Condition" minOccurs="0"/>
2250         <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
2251         <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
2252     </xs:sequence>
2253     <xs:attribute name="RuleId" type="xs:string" use="required"/>
2254     <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>
2255 </xs:complexType>

```

2256 The <Rule> element is of RuleType complex type.

2257 The <Rule> element contains the following attributes and elements:

2258 RuleId [Required]

2259 A string identifying this **rule**.

2260 Effect [Required]

2261 **Rule effect**. The value of this attribute is either "Permit" or "Deny".

2262 <Description> [Optional]

2263 A free-form description of the **rule**.

2264 <Target> [Optional]

2265 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If this  
2266 element is omitted, then the **target** for the <Rule> SHALL be defined by the <Target> element  
2267 of the enclosing <Policy> element. See Section 7.7 for details.

2268 <Condition> [Optional]

2269 A **predicate** that MUST be satisfied for the **rule** to be assigned its Effect value.

2270 <ObligationExpressions> [Optional]  
2271 A **conjunctive sequence** of **obligation** expressions which MUST be evaluated into **obligations**  
2272 by the PDP. The corresponding **obligations** MUST be fulfilled by the **PEP** in conjunction with  
2273 the **authorization decision**. See Section 7.18 for a description of how the set of **obligations** to  
2274 be returned by the **PDP** SHALL be determined. See section 7.2 about enforcement of  
2275 **obligations**.

2276 <AdviceExpressions> [Optional]  
2277 A **conjunctive sequence** of **advice** expressions which MUST be evaluated into **advice** by the **PDP**.  
2278 The corresponding **advice** provide supplementary information to the **PEP** in conjunction with the  
2279 **authorization decision**. See Section 7.18 for a description of how the set of **advice** to be  
2280 returned by the **PDP** SHALL be determined.

## 2281 5.22 Simple type EffectType

2282 The EffectType simple type defines the values allowed for the Effect attribute of the <Rule> element  
2283 and for the FulfillOn attribute of the <ObligationExpression> and <AdviceExpression>  
2284 elements.

```
2285 <xs:simpleType name="EffectType">  
2286   <xs:restriction base="xs:string">  
2287     <xs:enumeration value="Permit"/>  
2288     <xs:enumeration value="Deny"/>  
2289   </xs:restriction>  
2290 </xs:simpleType>
```

## 2291 5.23 Element <VariableDefinition>

2292 The <VariableDefinition> element SHALL be used to define a value that can be referenced by a  
2293 <VariableReference> element. The name supplied for its VariableId attribute SHALL NOT occur  
2294 in the VariableId attribute of any other <VariableDefinition> element within the encompassing  
2295 **policy**. The <VariableDefinition> element MAY contain undefined <VariableReference>  
2296 elements, but if it does, a corresponding <VariableDefinition> element MUST be defined later in  
2297 the encompassing **policy**. <VariableDefinition> elements MAY be grouped together or MAY be  
2298 placed close to the reference in the encompassing **policy**. There MAY be zero or more references to  
2299 each <VariableDefinition> element.

```
2300 <xs:element name="VariableDefinition" type="xacml:VariableDefinitionType"/>  
2301 <xs:complexType name="VariableDefinitionType">  
2302   <xs:sequence>  
2303     <xs:element ref="xacml:Expression"/>  
2304   </xs:sequence>  
2305   <xs:attribute name="VariableId" type="xs:string" use="required"/>  
2306 </xs:complexType>
```

2307 The <VariableDefinition> element is of VariableDefinitionType complex type. The  
2308 <VariableDefinition> element has the following elements and attributes:

2309 <Expression> [Required]  
2310 Any element of ExpressionType complex type.  
2311 VariableId [Required]  
2312 The name of the variable definition.

## 2313 5.24 Element <VariableReference>

2314 The <VariableReference> element is used to reference a value defined within the same  
2315 encompassing <Policy> element. The <VariableReference> element SHALL refer to the

2316 <VariableDefinition> element by **identifier equality** on the value of their respective VariableId  
2317 attributes. One and only one <VariableDefinition> MUST exist within the same encompassing  
2318 <Policy> element to which the <VariableReference> refers. There MAY be zero or more  
2319 <VariableReference> elements that refer to the same <VariableDefinition> element.

```
2320 <xs:element name="VariableReference" type="xacml:VariableReferenceType"  
2321 substitutionGroup="xacml:Expression"/>  
2322 <xs:complexType name="VariableReferenceType">  
2323   <xs:complexContent>  
2324     <xs:extension base="xacml:ExpressionType">  
2325       <xs:attribute name="VariableId" type="xs:string"  
2326         use="required"/>  
2327     </xs:extension>  
2328   </xs:complexContent>  
2329 </xs:complexType>
```

2330 The <VariableReference> element is of the VariableReferenceType complex type, which is of  
2331 the ExpressionType complex type and is a member of the <Expression> element substitution group.  
2332 The <VariableReference> element MAY appear any place where an <Expression> element occurs  
2333 in the schema.

2334 The <VariableReference> element has the following attribute:

2335 VariableId [Required]

2336 The name used to refer to the value defined in a <VariableDefinition> element.

## 2337 5.25 Element <Expression>

2338 The <Expression> element is not used directly in a **policy**. The <Expression> element signifies that  
2339 an element that extends the ExpressionType and is a member of the <Expression> element  
2340 substitution group SHALL appear in its place.

```
2341 <xs:element name="Expression" type="xacml:ExpressionType" abstract="true"/>  
2342 <xs:complexType name="ExpressionType" abstract="true"/>
```

2343 The following elements are in the <Expression> element substitution group:

2344 <Apply>, <AttributeSelector>, <AttributeValue>, <Function>, <VariableReference> and  
2345 <AttributeDesignator>.

## 2346 5.26 Element <Condition>

2347 The <Condition> element is a Boolean function over **attributes** or functions of **attributes**.

```
2348 <xs:element name="Condition" type="xacml:ConditionType"/>  
2349 <xs:complexType name="ConditionType">  
2350   <xs:sequence>  
2351     <xs:element ref="xacml:Expression"/>  
2352   </xs:sequence>  
2353 </xs:complexType>
```

2354 The <Condition> contains one <Expression> element, with the restriction that the <Expression>  
2355 return data-type MUST be "http://www.w3.org/2001/XMLSchema#boolean". Evaluation of the  
2356 <Condition> element is described in Section 7.9.

## 2357 5.27 Element <Apply>

2358 The <Apply> element denotes application of a function to its arguments, thus encoding a function call.  
2359 The <Apply> element can be applied to any combination of the members of the <Expression>  
2360 element substitution group. See Section 5.25.

```

2361 <xs:element name="Apply" type="xacml:ApplyType"
2362 substitutionGroup="xacml:Expression"/>
2363 <xs:complexType name="ApplyType">
2364   <xs:complexContent>
2365     <xs:extension base="xacml:ExpressionType">
2366       <xs:sequence>
2367         <xs:element ref="xacml:Description" minOccurs="0"/>
2368         <xs:element ref="xacml:Expression" minOccurs="0"
2369           maxOccurs="unbounded"/>
2370       </xs:sequence>
2371       <xs:attribute name="FunctionId" type="xs:anyURI"
2372         use="required"/>
2373     </xs:extension>
2374   </xs:complexContent>
2375 </xs:complexType>

```

2376 The <Apply> element is of ApplyType complex type.

2377 The <Apply> element contains the following attributes and elements:

2378 FunctionId [Required]

2379       The identifier of the function to be applied to the arguments. XACML-defined functions are  
2380       described in Appendix A.3.

2381 <Description> [Optional]

2382       A free-form description of the <Apply> element.

2383 <Expression> [Optional]

2384       Arguments to the function, which may include other functions.

## 2385 5.28 Element <Function>

2386 The <Function> element SHALL be used to name a function as an argument to the function defined by  
2387 the parent <Apply> element.

```

2388 <xs:element name="Function" type="xacml:FunctionType"
2389 substitutionGroup="xacml:Expression"/>
2390 <xs:complexType name="FunctionType">
2391   <xs:complexContent>
2392     <xs:extension base="xacml:ExpressionType">
2393       <xs:attribute name="FunctionId" type="xs:anyURI"
2394         use="required"/>
2395     </xs:extension>
2396   </xs:complexContent>
2397 </xs:complexType>

```

2398 The <Function> element is of FunctionType complex type.

2399 The <Function> element contains the following attribute:

2400 FunctionId [Required]

2401       The identifier of the function.

## 2402 5.29 Element <AttributeDesignator>

2403 The <AttributeDesignator> element retrieves a **bag** of values for a **named attribute** from the  
2404 request **context**. A **named attribute** SHALL be considered present if there is at least one **attribute** that  
2405 matches the criteria set out below.

2406 The <AttributeDesignator> element SHALL return a **bag** containing all the **attribute** values that are  
2407 matched by the **named attribute**. In the event that no matching **attribute** is present in the **context**, the

2408 MustBePresent attribute governs whether this element returns an empty **bag** or “Indeterminate”. See  
2409 Section 7.3.5.

2410 The <AttributeDesignator> MAY appear in the <Match> element and MAY be passed to the  
2411 <Apply> element as an argument.

2412 The <AttributeDesignator> element is of the AttributeDesignatorType complex type.

```
2413 <xs:element name="AttributeDesignator" type="xacml:AttributeDesignatorType"  
2414 substitutionGroup="xacml:Expression"/>  
2415 <xs:complexType name="AttributeDesignatorType">  
2416 <xs:complexContent>  
2417 <xs:extension base="xacml:ExpressionType">  
2418 <xs:attribute name="Category" type="xs:anyURI"  
2419 use="required"/>  
2420 <xs:attribute name="AttributeId" type="xs:anyURI"  
2421 use="required"/>  
2422 <xs:attribute name="DataType" type="xs:anyURI"  
2423 use="required"/>  
2424 <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2425 <xs:attribute name="MustBePresent" type="xs:boolean"  
2426 use="required"/>  
2427 </xs:extension>  
2428 </xs:complexContent>  
2429 </xs:complexType>
```

2430 A **named attribute** SHALL match an **attribute** if the values of their respective Category,  
2431 AttributeId, DataType and Issuer attributes match. The attribute designator’s Category MUST  
2432 match, by **identifier equality**, the Category of the <Attributes> element in which the **attribute** is  
2433 present. The attribute designator’s AttributeId MUST match, by **identifier equality**, the  
2434 AttributeId of the attribute. The attribute designator’s DataType MUST match, by **identifier**  
2435 **equality**, the DataType of the same **attribute**.

2436 If the Issuer attribute is present in the attribute designator, then it MUST match, using the  
2437 “urn:oasis:names:tc:xacml:1.0:function:string-equal” function, the Issuer of the same **attribute**. If the  
2438 Issuer is not present in the attribute designator, then the matching of the **attribute** to the **named**  
2439 **attribute** SHALL be governed by AttributeId and DataType attributes alone.

2440 The <AttributeDesignatorType> contains the following attributes:

2441 Category [Required]

2442 This attribute SHALL specify the Category with which to match the **attribute**.

2443 AttributeId [Required]

2444 This attribute SHALL specify the AttributeId with which to match the **attribute**.

2445 DataType [Required]

2446 The **bag** returned by the <AttributeDesignator> element SHALL contain values of this data-  
2447 type.

2448 Issuer [Optional]

2449 This attribute, if supplied, SHALL specify the Issuer with which to match the **attribute**.

2450 MustBePresent [Required]

2451 This attribute governs whether the element returns “Indeterminate” or an empty **bag** in the event  
2452 the **named attribute** is absent from the request **context**. See Section 7.3.5. Also see Sections  
2453 7.19.2 and 7.19.3.

2454 **5.30 Element <AttributeSelector>**

2455 The <AttributeSelector> element produces a **bag** of unnamed and uncategorized **attribute** values.  
2456 The values shall be constructed from the node(s) selected by applying the XPath expression given by the  
2457 element's Path attribute to the XML content indicated by the element's Category attribute. Support for  
2458 the <AttributeSelector> element is OPTIONAL.

2459 See section 7.3.7 for details of <AttributeSelector> evaluation.

```
2460 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"  
2461 substitutionGroup="xacml:Expression"/>  
2462 <xs:complexType name="AttributeSelectorType">  
2463 <xs:complexContent>  
2464 <xs:extension base="xacml:ExpressionType">  
2465 <xs:attribute name="Category" type="xs:anyURI"  
2466 use="required"/>  
2467 <xs:attribute name="ContextSelectorId" type="xs:anyURI"  
2468 use="optional"/>  
2469 <xs:attribute name="Path" type="xs:string"  
2470 use="required"/>  
2471 <xs:attribute name="DataType" type="xs:anyURI"  
2472 use="required"/>  
2473 <xs:attribute name="MustBePresent" type="xs:boolean"  
2474 use="required"/>  
2475 </xs:extension>  
2476 </xs:complexContent>  
2477 </xs:complexType>
```

2478 The <AttributeSelector> element is of AttributeSelectorType complex type.

2479 The <AttributeSelector> element has the following attributes:

2480 Category [Required]

2481 This attribute SHALL specify the **attributes** category of the <Content> element containing the  
2482 XML from which nodes will be selected. It also indicates the **attributes** category containing the  
2483 applicable ContextSelectorId attribute, if the element includes a ContextSelectorId xml  
2484 attribute.

2485 ContextSelectorId [Optional]

2486 This attribute refers to the **attribute** (by its AttributeId) in the request **context** in the category  
2487 given by the Category attribute. The referenced **attribute** MUST have data type  
2488 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression, and must select a single node in the  
2489 <Content> element. The XPathCategory attribute of the referenced **attribute** MUST be equal  
2490 to the Category attribute of the **attribute selector**.

2491 Path [Required]

2492 This attribute SHALL contain an XPath expression to be evaluated against the specified XML  
2493 content. See Section 7.3.7 for details of the XPath evaluation during <AttributeSelector>  
2494 processing.

2495 DataType [Required]

2496 The attribute specifies the datatype of the values returned from the evaluation of this  
2497 <AttributeSelector> element.

2498 MustBePresent [Required]

2499 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the event  
2500 the XPath expression selects no node. See Section 7.3.5. Also see Sections 7.19.2 and 7.19.3.

### 2501 5.31 Element <AttributeValue>

2502 The <AttributeValue> element SHALL contain a literal **attribute** value.

```
2503 <xs:element name="AttributeValue" type="xacml:AttributeValueType"  
2504 substitutionGroup="xacml:Expression"/>  
2505 <xs:complexType name="AttributeValueType" mixed="true">  
2506 <xs:complexContent mixed="true">  
2507 <xs:extension base="xacml:ExpressionType">  
2508 <xs:sequence>  
2509 <xs:any namespace="##any" processContents="lax"  
2510 minOccurs="0" maxOccurs="unbounded"/>  
2511 </xs:sequence>  
2512 <xs:attribute name="DataType" type="xs:anyURI"  
2513 use="required"/>  
2514 <xs:anyAttribute namespace="##any" processContents="lax"/>  
2515 </xs:extension>  
2516 </xs:complexContent>  
2517 </xs:complexType>
```

2518 The <AttributeValue> element is of AttributeValueType complex type.

2519 The <AttributeValue> element has the following attributes:

2520 DataType [Required]

2521 The data-type of the **attribute** value.

### 2522 5.32 Element <Obligations>

2523 The <Obligations> element SHALL contain a set of <Obligation> elements.

```
2524 <xs:element name="Obligations" type="xacml:ObligationsType"/>  
2525 <xs:complexType name="ObligationsType">  
2526 <xs:sequence>  
2527 <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>  
2528 </xs:sequence>  
2529 </xs:complexType>
```

2530 The <Obligations> element is of ObligationsType complexType.

2531 The <Obligations> element contains the following element:

2532 <Obligation> [One to Many]

2533 A sequence of **obligations**. See Section 5.34.

### 2534 5.33 Element <AssociatedAdvice>

2535 The <AssociatedAdvice> element SHALL contain a set of <Advice> elements.

```
2536 <xs:element name="AssociatedAdvice" type="xacml:AssociatedAdviceType"/>  
2537 <xs:complexType name="AssociatedAdviceType">  
2538 <xs:sequence>  
2539 <xs:element ref="xacml:Advice" maxOccurs="unbounded"/>  
2540 </xs:sequence>  
2541 </xs:complexType>
```

2542 The <AssociatedAdvice> element is of AssociatedAdviceType complexType.

2543 The <AssociatedAdvice> element contains the following element:

2544 <Advice> [One to Many]

2545 A sequence of **advice**. See Section 5.35.

### 2546 5.34 Element <Obligation>

2547 The <Obligation> element SHALL contain an identifier for the **obligation** and a set of **attributes** that  
2548 form arguments of the action defined by the **obligation**.

```
2549 <xs:element name="Obligation" type="xacml:ObligationType"/>
2550 <xs:complexType name="ObligationType">
2551   <xs:sequence>
2552     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2553       maxOccurs="unbounded"/>
2554   </xs:sequence>
2555   <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2556 </xs:complexType>
```

2557 The <Obligation> element is of ObligationType complexType. See Section 7.18 for a description  
2558 of how the set of **obligations** to be returned by the **PDP** is determined.

2559 The <Obligation> element contains the following elements and attributes:

2560 ObligationId [Required]

2561 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the **PEP**.

2562 <AttributeAssignment> [Optional]

2563 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be  
2564 interpreted by the **PEP**.

### 2565 5.35 Element <Advice>

2566 The <Advice> element SHALL contain an identifier for the **advice** and a set of **attributes** that form  
2567 arguments of the supplemental information defined by the **advice**.

```
2568 <xs:element name="Advice" type="xacml:AdviceType"/>
2569 <xs:complexType name="AdviceType">
2570   <xs:sequence>
2571     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2572       maxOccurs="unbounded"/>
2573   </xs:sequence>
2574   <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>
2575 </xs:complexType>
```

2576 The <Advice> element is of AdviceType complexType. See Section 7.18 for a description of how the  
2577 set of **advice** to be returned by the **PDP** is determined.

2578 The <Advice> element contains the following elements and attributes:

2579 AdviceId [Required]

2580 **Advice** identifier. The value of the **advice** identifier MAY be interpreted by the **PEP**.

2581 <AttributeAssignment> [Optional]

2582 **Advice** arguments assignment. The values of the **advice** arguments MAY be interpreted by the  
2583 **PEP**.

### 2584 5.36 Element <AttributeAssignment>

2585 The <AttributeAssignment> element is used for including arguments in **obligation** and **advice**  
2586 expressions. It SHALL contain an AttributeId and the corresponding **attribute** value, by extending  
2587 the AttributeValueType type definition. The <AttributeAssignment> element MAY be used in  
2588 any way that is consistent with the schema syntax, which is a sequence of <xs:any> elements. The  
2589 value specified SHALL be understood by the **PEP**, but it is not further specified by XACML. See Section  
2590 7.18. Section 4.2.4.3 provides a number of examples of arguments included in **obligation**.expressions.

```
2591 <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>
```

```

2592 <xs:complexType name="AttributeAssignmentType" mixed="true">
2593   <xs:complexContent>
2594     <xs:extension base="xacml:AttributeValueType">
2595       <xs:attribute name="AttributeId" type="xs:anyURI"
2596         use="required"/>
2597       <xs:attribute name="Category" type="xs:anyURI"
2598         use="optional"/>
2599       <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2600     </xs:extension>
2601   </xs:complexContent>
2602 </xs:complexType>

```

2603 The <AttributeAssignment> element is of AttributeAssignmentType complex type.

2604 The <AttributeAssignment> element contains the following attributes:

2605 AttributeId [Required]

2606       The **attribute** Identifier.

2607 Category [Optional]

2608       An optional category of the **attribute**. If this attribute is missing, the **attribute** has no category.

2609       The **PEP** SHALL interpret the significance and meaning of any Category attribute. Non-  
2610       normative note: an expected use of the category is to disambiguate **attributes** which are relayed  
2611       from the request.

2612 Issuer [Optional]

2613       An optional issuer of the **attribute**. If this attribute is missing, the **attribute** has no issuer. The  
2614       **PEP** SHALL interpret the significance and meaning of any Issuer attribute. Non-normative note:  
2615       an expected use of the issuer is to disambiguate **attributes** which are relayed from the request.

## 2616 5.37 Element <ObligationExpressions>

2617 The <ObligationExpressions> element SHALL contain a set of <ObligationExpression>  
2618 elements.

```

2619 <xs:element name="ObligationExpressions"
2620   type="xacml:ObligationExpressionsType"/>
2621 <xs:complexType name="ObligationExpressionsType">
2622   <xs:sequence>
2623     <xs:element ref="xacml:ObligationExpression" maxOccurs="unbounded"/>
2624   </xs:sequence>
2625 </xs:complexType>

```

2626 The <ObligationExpressions> element is of ObligationExpressionsType complexType.

2627 The <ObligationExpressions> element contains the following element:

2628 <ObligationExpression> [One to Many]

2629       A sequence of **obligations** expressions. See Section 5.39.

## 2630 5.38 Element <AdviceExpressions>

2631 The <AdviceExpressions> element SHALL contain a set of <AdviceExpression> elements.

```

2632 <xs:element name="AdviceExpressions" type="xacml:AdviceExpressionsType"/>
2633 <xs:complexType name="AdviceExpressionsType">
2634   <xs:sequence>
2635     <xs:element ref="xacml:AdviceExpression" maxOccurs="unbounded"/>
2636   </xs:sequence>
2637 </xs:complexType>

```

2638 The <AdviceExpressions> element is of AdviceExpressionsType complexType.

2639 The <AdviceExpressions> element contains the following element:

2640 <AdviceExpression> [One to Many]

2641 A sequence of **advice** expressions. See Section 5.40.

### 2642 5.39 Element <ObligationExpression>

2643 The <ObligationExpression> element evaluates to an **obligation** and SHALL contain an identifier  
2644 for an **obligation** and a set of expressions that form arguments of the action defined by the **obligation**.  
2645 The FulfillOn attribute SHALL indicate the **effect** for which this **obligation** must be fulfilled by the  
2646 **PEP**.

```
2647 <xs:element name="ObligationExpression"  
2648     type="xacml:ObligationExpressionType"/>  
2649 <xs:complexType name="ObligationExpressionType">  
2650     <xs:sequence>  
2651         <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"  
2652             maxOccurs="unbounded"/>  
2653     </xs:sequence>  
2654     <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>  
2655     <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>  
2656 </xs:complexType>
```

2657 The <ObligationExpression> element is of ObligationExpressionType complexType. See  
2658 Section 7.18 for a description of how the set of **obligations** to be returned by the **PDP** is determined.

2659 The <ObligationExpression> element contains the following elements and attributes:

2660 ObligationId [Required]

2661 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the **PEP**.

2662 FulfillOn [Required]

2663 The **effect** for which this **obligation** must be fulfilled by the **PEP**.

2664 <AttributeAssignmentExpression> [Optional]

2665 **Obligation** arguments in the form of expressions. The expressions SHALL be evaluated by the  
2666 PDP to constant <AttributeValue> elements or **bags**, which shall be the attribute  
2667 assignments in the <Obligation> returned to the PEP. If an  
2668 <AttributeAssignmentExpression> evaluates to an atomic **attribute** value, then there  
2669 MUST be one resulting <AttributeAssignment> which MUST contain this single **attribute**  
2670 value. If the <AttributeAssignmentExpression> evaluates to a **bag**, then there MUST be a  
2671 resulting <AttributeAssignment> for each of the values in the **bag**. If the **bag** is empty, there  
2672 shall be no <AttributeAssignment> from this <AttributeAssignmentExpression>. The  
2673 values of the **obligation** arguments SHALL be interpreted by the **PEP**.

### 2674 5.40 Element <AdviceExpression>

2675 The <AdviceExpression> element evaluates to an **advice** and SHALL contain an identifier for an  
2676 **advice** and a set of expressions that form arguments of the supplemental information defined by the  
2677 **advice**. The AppliesTo attribute SHALL indicate the **effect** for which this **advice** must be provided to  
2678 the **PEP**.

```
2679 <xs:element name="AdviceExpression" type="xacml:AdviceExpressionType"/>  
2680 <xs:complexType name="AdviceExpressionType">  
2681     <xs:sequence>  
2682         <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"  
2683             maxOccurs="unbounded"/>  
2684     </xs:sequence>  
2685     <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>  
2686     <xs:attribute name="AppliesTo" type="xacml:EffectType" use="required"/>
```

2687

```
</xs:complexType>
```

2688 The <AdviceExpression> element is of AdviceExpressionType complexType. See Section 7.18  
2689 for a description of how the set of **advice** to be returned by the **PDP** is determined.

2690 The <AdviceExpression> element contains the following elements and attributes:

2691 AdviceId [Required]

2692         **Advice** identifier. The value of the **advice** identifier MAY be interpreted by the **PEP**.

2693 AppliesTo [Required]

2694         The **effect** for which this **advice** must be provided to the **PEP**.

2695 <AttributeAssignmentExpression> [Optional]

2696         **Advice** arguments in the form of expressions. The expressions SHALL be evaluated by the PDP  
2697 to constant <AttributeValue> elements or **bags**, which shall be the attribute assignments in  
2698 the <Advice> returned to the PEP. If an <AttributeAssignmentExpression> evaluates to  
2699 an atomic **attribute** value, then there MUST be one resulting <AttributeAssignment> which  
2700 MUST contain this single **attribute** value. If the <AttributeAssignmentExpression>  
2701 evaluates to a **bag**, then there MUST be a resulting <AttributeAssignment> for each of the  
2702 values in the **bag**. If the **bag** is empty, there shall be no <AttributeAssignment> from this  
2703 <AttributeAssignmentExpression>. The values of the **advice** arguments MAY be  
2704 interpreted by the **PEP**.

## 2705 5.41 Element <AttributeAssignmentExpression>

2706 The <AttributeAssignmentExpression> element is used for including arguments in **obligations**  
2707 and **advice**. It SHALL contain an AttributeId and an expression which SHALL be evaluated into the  
2708 corresponding **attribute** value. The value specified SHALL be understood by the **PEP**, but it is not further  
2709 specified by XACML. See Section 7.18. Section 4.2.4.3 provides a number of examples of arguments  
2710 included in **obligations**.

```
2711 <xs:element name="AttributeAssignmentExpression"  
2712   type="xacml:AttributeAssignmentExpressionType"/>  
2713 <xs:complexType name="AttributeAssignmentExpressionType">  
2714   <xs:sequence>  
2715     <xs:element ref="xacml:Expression"/>  
2716   </xs:sequence>  
2717   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2718   <xs:attribute name="Category" type="xs:anyURI" use="optional"/>  
2719   <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2720 </xs:complexType>
```

2721 The <AttributeAssignmentExpression> element is of AttributeAssignmentExpressionType  
2722 complex type.

2723 The <AttributeAssignmentExpression> element contains the following attributes:

2724 <Expression> [Required]

2725         The expression which evaluates to a constant **attribute** value or a bag of zero or more attribute  
2726 values. See section 5.25.

2727 AttributeId [Required]

2728         The **attribute** identifier. The value of the AttributeId attribute in the resulting  
2729 <AttributeAssignment> element MUST be equal to this value.

2730 Category [Optional]

2731         An optional category of the **attribute**. If this attribute is missing, the **attribute** has no category.  
2732 The value of the Category attribute in the resulting <AttributeAssignment> element MUST be  
2733 equal to this value.

2734 Issuer [Optional]  
2735 An optional issuer of the **attribute**. If this attribute is missing, the **attribute** has no issuer. The  
2736 value of the Issuer attribute in the resulting <AttributeAssignment> element MUST be equal to  
2737 this value.

## 2738 5.42 Element <Request>

2739 The <Request> element is an abstraction layer used by the **policy** language. For simplicity of  
2740 expression, this document describes **policy** evaluation in terms of operations on the **context**. However a  
2741 conforming **PDP** is not required to actually instantiate the **context** in the form of an XML document. But,  
2742 any system conforming to the XACML specification MUST produce exactly the same **authorization**  
2743 **decisions** as if all the inputs had been transformed into the form of an <Request> element.

2744 The <Request> element contains <Attributes> elements. There may be multiple <Attributes>  
2745 elements with the same Category attribute if the **PDP** implements the multiple decision profile, see  
2746 **[Multi]**. Under other conditions, it is a syntax error if there are multiple <Attributes> elements with the  
2747 same Category (see Section 7.19.2 for error codes).

```
2748 <xs:element name="Request" type="xacml:RequestType"/>  
2749 <xs:complexType name="RequestType">  
2750   <xs:sequence>  
2751     <xs:element ref="xacml:RequestDefaults" minOccurs="0"/>  
2752     <xs:element ref="xacml:Attributes" maxOccurs="unbounded"/>  
2753     <xs:element ref="xacml:MultiRequests" minOccurs="0"/>  
2754   </xs:sequence>  
2755   <xs:attribute name="ReturnPolicyIdList" type="xs:boolean" use="required"/>  
2756   <xs:attribute name="CombinedDecision" type="xs:boolean" use="required" />  
2757 </xs:complexType>
```

2758 The <Request> element is of RequestType complex type.

2759 The <Request> element contains the following elements and attributes:

2760 ReturnPolicyIdList [Required]

2761 This attribute is used to request that the **PDP** return a list of all fully applicable **policies** and  
2762 **policy sets** which were used in the decision as a part of the decision response.

2763 CombinedDecision [Required]

2764 This attribute is used to request that the **PDP** combines multiple decisions into a single decision.  
2765 The use of this attribute is specified in **[Multi]**. If the **PDP** does not implement the relevant  
2766 functionality in **[Multi]**, then the **PDP** must return an Indeterminate with a status code of  
2767 urn:oasis:names:tc:xacml:1.0:status:processing-error if it receives a request with this attribute set  
2768 to "true".

2769 <RequestDefaults> [Optional]

2770 Contains default values for the request, such as XPath version. See section 5.43.

2771 <Attributes> [One to Many]

2772 Specifies information about **attributes** of the request **context** by listing a sequence of  
2773 <Attribute> elements associated with an **attribute** category. One or more <Attributes>  
2774 elements are allowed. Different <Attributes> elements with different categories are used to  
2775 represent information about the **subject**, **resource**, **action**, **environment** or other categories of  
2776 the **access** request.

2777 <MultiRequests> [Optional]

2778 Lists multiple **request contexts** by references to the <Attributes> elements. Implementation  
2779 of this element is optional. The semantics of this element is defined in **[Multi]**. If the  
2780 implementation does not implement this element, it MUST return an Indeterminate result if it  
2781 encounters this element. See section 5.50.

## 2782 5.43 Element <RequestDefaults>

2783 The <RequestDefaults> element SHALL specify default values that apply to the <Request> element.

```
2784 <xs:element name="RequestDefaults" type="xacml:RequestDefaultsType"/>
2785 <xs:complexType name="RequestDefaultsType">
2786   <xs:sequence>
2787     <xs:choice>
2788       <xs:element ref="xacml:XPathVersion"/>
2789     </xs:choice>
2790   </xs:sequence>
2791 </xs:complexType>
```

2792 <RequestDefaults> element is of RequestDefaultsType complex type.

2793 The <RequestDefaults> element contains the following elements:

2794 <XPathVersion> [Optional]

2795 Default XPath version for XPath expressions occurring in the request.

## 2796 5.44 Element <Attributes>

2797 The <Attributes> element specifies **attributes** of a **subject**, **resource**, **action**, **environment** or  
2798 another category by listing a sequence of <Attribute> elements associated with the category.

```
2799 <xs:element name="Attributes" type="xacml:AttributesType"/>
2800 <xs:complexType name="AttributesType">
2801   <xs:sequence>
2802     <xs:element ref="xacml:Content" minOccurs="0"/>
2803     <xs:element ref="xacml:Attribute" minOccurs="0"
2804       maxOccurs="unbounded"/>
2805   </xs:sequence>
2806   <xs:attribute name="Category" type="xs:anyURI" use="required"/>
2807   <xs:attribute ref="xml:id" use="optional"/>
2808 </xs:complexType><xs:complexType name="SubjectType">
```

2809 The <Attributes> element is of AttributesType complex type.

2810 The <Attributes> element contains the following elements and attributes:

2811 Category [Required]

2812 This attribute indicates which **attribute** category the contained **attributes** belong to. The  
2813 Category attribute is used to differentiate between **attributes** of **subject**, **resource**, **action**,  
2814 **environment** or other categories.

2815 xml:id [Optional]

2816 This attribute provides a unique identifier for this <Attributes> element. See [XMLid] It is  
2817 primarily intended to be referenced in multiple requests. See [Multi].

2818 <Content> [Optional]

2819 Specifies additional sources of **attributes** in free form XML document format which can be  
2820 referenced using <AttributeSelector> elements.

2821 <Attribute> [Any Number]

2822 A sequence of **attributes** that apply to the category of the request.

## 2823 5.45 Element <Content>

2824 The <Content> element is a notional placeholder for additional **attributes**, typically the content of the  
2825 **resource**.

```
2826 <xs:element name="Content" type="xacml:ContentType"/>
```

```

2827 <xs:complexType name="ContentType" mixed="true">
2828   <xs:sequence>
2829     <xs:any namespace="##any" processContents="lax"/>
2830   </xs:sequence>
2831 </xs:complexType>

```

2832 The <Content> element is of ContentType complex type.

2833 The <Content> element has exactly one arbitrary type child element.

## 2834 5.46 Element <Attribute>

2835 The <Attribute> element is the central abstraction of the request **context**. It contains **attribute** meta-  
2836 data and one or more **attribute** values. The **attribute** meta-data comprises the **attribute** identifier and  
2837 the **attribute** issuer. <AttributeDesignator> elements in the **policy** MAY refer to **attributes** by  
2838 means of this meta-data.

```

2839 <xs:element name="Attribute" type="xacml:AttributeType"/>
2840 <xs:complexType name="AttributeType">
2841   <xs:sequence>
2842     <xs:element ref="xacml:AttributeValue" maxOccurs="unbounded"/>
2843   </xs:sequence>
2844   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
2845   <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2846   <xs:attribute name="IncludeInResult" type="xs:boolean" use="required"/>
2847 </xs:complexType>

```

2848 The <Attribute> element is of AttributeType complex type.

2849 The <Attribute> element contains the following attributes and elements:

2850 AttributeId [Required]

2851 The **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly  
2852 used **attributes**. See Appendix Appendix B.

2853 Issuer [Optional]

2854 The **Attribute** issuer. For example, this attribute value MAY be an x500Name that binds to a  
2855 public key, or it may be some other identifier exchanged out-of-band by issuing and relying  
2856 parties.

2857 IncludeInResult [Default: false]

2858 Whether to include this **attribute** in the result. This is useful to correlate requests with their  
2859 responses in case of multiple requests.

2860 <AttributeValue> [One to Many]

2861 One or more **attribute** values. Each **attribute** value MAY have contents that are empty, occur  
2862 once or occur multiple times.

## 2863 5.47 Element <Response>

2864 The <Response> element is an abstraction layer used by the **policy** language. Any proprietary system  
2865 using the XACML specification MUST transform an XACML **context** <Response> element into the form  
2866 of its **authorization decision**.

2867 The <Response> element encapsulates the **authorization decision** produced by the **PDP**. It includes a  
2868 sequence of one or more results, with one <Result> element per requested **resource**. Multiple results  
2869 MAY be returned by some implementations, in particular those that support the XACML Profile for  
2870 Requests for Multiple Resources [**Multi**]. Support for multiple results is OPTIONAL.

```

2871 <xs:element name="Response" type="xacml:ResponseType"/>
2872 <xs:complexType name="ResponseType">
2873   <xs:sequence>

```

```
2874     <xs:element ref="xacml:Result" maxOccurs="unbounded"/>
2875     </xs:sequence>
2876 </xs:complexType>
```

2877 The <Response> element is of ResponseType complex type.

2878 The <Response> element contains the following elements:

2879 <Result> [One to Many]

2880 An **authorization decision** result. See Section 5.48.

## 2881 5.48 Element <Result>

2882 The <Result> element represents an **authorization decision** result. It MAY include a set of  
2883 **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or cannot fulfill an  
2884 **obligation**, then the action of the PEP is determined by its bias, see section 7.1. It MAY include a set of  
2885 **advice** with supplemental information which MAY be safely ignored by the **PEP**.

```
2886 <xs:complexType name="ResultType">
2887   <xs:sequence>
2888     <xs:element ref="xacml:Decision"/>
2889     <xs:element ref="xacml:Status" minOccurs="0"/>
2890     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2891     <xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
2892     <xs:element ref="xacml:Attributes" minOccurs="0"
2893       maxOccurs="unbounded"/>
2894     <xs:element ref="xacml:PolicyIdentifierList" minOccurs="0"/>
2895   </xs:sequence>
2896 </xs:complexType>
```

2897 The <Result> element is of ResultType complex type.

2898 The <Result> element contains the following attributes and elements:

2899 <Decision> [Required]

2900 The **authorization decision**: "Permit", "Deny", "Indeterminate" or "NotApplicable".

2901 <Status> [Optional]

2902 Indicates whether errors occurred during evaluation of the **decision request**, and optionally,  
2903 information about those errors. If the <Response> element contains <Result> elements whose  
2904 <Status> elements are all identical, and the <Response> element is contained in a protocol  
2905 wrapper that can convey status information, then the common status information MAY be placed  
2906 in the protocol wrapper and this <Status> element MAY be omitted from all <Result>  
2907 elements.

2908 <Obligations> [Optional]

2909 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or cannot  
2910 fulfill an **obligation**, then the action of the PEP is determined by its bias, see section 7.2. See  
2911 Section 7.18 for a description of how the set of **obligations** to be returned by the **PDP** is  
2912 determined.

2913 <AssociatedAdvice> [Optional]

2914 A list of **advice** that provide supplemental information to the **PEP**. If the **PEP** does not  
2915 understand an **advice**, the PEP may safely ignore the **advice**. See Section 7.18 for a description  
2916 of how the set of **advice** to be returned by the **PDP** is determined.

2917 <Attributes> [Optional]

2918 A list of **attributes** that were part of the request. The choice of which **attributes** are included here  
2919 is made with the IncludeInResult attribute of the <Attribute> elements of the request. See  
2920 section 5.46.

2921 **<PolicyIdentifierList>** [Optional]  
2922 If the `ReturnPolicyIdList` attribute in the `<Request>` is true (see section 5.42), a **PDP** that  
2923 implements this optional feature **MUST** return a list of all **policies** which were found to be fully  
2924 applicable. That is, all **policies** where both the `<Target>` matched and the `<Condition>`  
2925 evaluated to true, whether or not the `<Effect>` was the same or different from the `<Decision>`.

## 2926 **5.49 Element <PolicyIdentifierList>**

2927 The `<PolicyIdentifierList>` element contains a list of **policy** and **policy set** identifiers of **policies**  
2928 which have been applicable to a request. The list is unordered.

```
2929 <xs:element name="PolicyIdentifierList"  
2930   type="xacml:PolicyIdentifierListType"/>  
2931 <xs:complexType name="PolicyIdentifierListType">  
2932   <xs:choice minOccurs="0" maxOccurs="unbounded">  
2933     <xs:element ref="xacml:PolicyIdReference"/>  
2934     <xs:element ref="xacml:PolicySetIdReference"/>  
2935   </xs:choice>  
2936 </xs:complexType>
```

2937 The `<PolicyIdentifierList>` element is of `PolicyIdentifierListType` complex type.

2938 The `<PolicyIdentifierList>` element contains the following elements.

2939 `<PolicyIdReference>` [Any number]

2940 The identifier and version of a **policy** which was applicable to the request. See section 5.11. The  
2941 `<PolicyIdReference>` element **MUST** use the `Version` attribute to specify the version and  
2942 **MUST NOT** use the `LatestVersion` or `EarliestVersion` attributes.

2943 `<PolicySetIdReference>` [Any number]

2944 The identifier and version of a **policy set** which was applicable to the request. See section 5.10.  
2945 The `<PolicySetIdReference>` element **MUST** use the `Version` attribute to specify the  
2946 version and **MUST NOT** use the `LatestVersion` or `EarliestVersion` attributes.

## 2947 **5.50 Element <MultiRequests>**

2948 The `<MultiRequests>` element contains a list of requests by reference to `<Attributes>` elements in  
2949 the enclosing `<Request>` element. The semantics of this element are defined in **[Multi]**. Support for this  
2950 element is optional. If an implementation does not support this element, but receives it, the  
2951 implementation **MUST** generate an "Indeterminate" response.

```
2952 <xs:element name="MultiRequests" type="xacml:MultiRequestsType"/>  
2953 <xs:complexType name="MultiRequestsType">  
2954   <xs:sequence>  
2955     <xs:element ref="xacml:RequestReference" maxOccurs="unbounded"/>  
2956   </xs:sequence>  
2957 </xs:complexType>
```

2958 The `<MultiRequests>` element contains the following elements.

2959 `<RequestReference>` [one to many]

2960 Defines a request instance by reference to `<Attributes>` elements in the enclosing  
2961 `<Request>` element. See section 5.51.

## 2962 **5.51 Element <RequestReference>**

2963 The `<RequestReference>` element defines an instance of a request in terms of references to  
2964 `<Attributes>` elements. The semantics of this element are defined in **[Multi]**. Support for this element  
2965 is optional.

```

2966 <xs:element name="RequestReference" type="xacml:RequestReference" />
2967 <xs:complexType name="RequestReferenceType">
2968   <xs:sequence>
2969     <xs:element ref="xacml:AttributesReference" maxOccurs="unbounded" />
2970   </xs:sequence>
2971 </xs:complexType>

```

2972 The <RequestReference> element contains the following elements.

2973 <AttributesReference> [one to many]

2974 A reference to an <Attributes> element in the enclosing <Request> element. See section  
2975 5.52.

## 2976 5.52 Element <AttributesReference>

2977 The <AttributesReference> element makes a reference to an <Attributes> element. The  
2978 meaning of this element is defined in [Multi]. Support for this element is optional.

```

2979 <xs:element name="AttributesReference" type="xacml:AttributesReference" />
2980 <xs:complexType name="AttributesReferenceType">
2981   <xs:attribute name="ReferenceId" type="xs:IDREF" use="required" />
2982 </xs:complexType>

```

2983 The <AttributesReference> element contains the following attributes.

2984 ReferenceId [required]

2985 A reference to the xml:id attribute of an <Attributes> element in the enclosing <Request>  
2986 element.

## 2987 5.53 Element <Decision>

2988 The <Decision> element contains the result of *policy* evaluation.

```

2989 <xs:element name="Decision" type="xacml:DecisionType" />
2990 <xs:simpleType name="DecisionType">
2991   <xs:restriction base="xs:string">
2992     <xs:enumeration value="Permit" />
2993     <xs:enumeration value="Deny" />
2994     <xs:enumeration value="Indeterminate" />
2995     <xs:enumeration value="NotApplicable" />
2996   </xs:restriction>
2997 </xs:simpleType>

```

2998 The <Decision> element is of DecisionType simple type.

2999 The values of the <Decision> element have the following meanings:

3000 "Permit": the requested **access** is permitted.

3001 "Deny": the requested **access** is denied.

3002 "Indeterminate": the **PDP** is unable to evaluate the requested **access**. Reasons for such inability  
3003 include: missing **attributes**, network errors while retrieving **policies**, division by zero during  
3004 **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc.

3005 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

## 3006 5.54 Element <Status>

3007 The <Status> element represents the status of the **authorization decision** result.

```

3008 <xs:element name="Status" type="xacml:StatusType" />
3009 <xs:complexType name="StatusType">
3010   <xs:sequence>

```

3011  
3012  
3013  
3014  
3015

```
<xs:element ref="xacml:StatusCode"/>
<xs:element ref="xacml:StatusMessage" minOccurs="0"/>
<xs:element ref="xacml:StatusDetail" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

3016 The <Status> element is of StatusType complex type.

3017 The <Status> element contains the following elements:

3018 <StatusCode> [Required]

3019 Status code.

3020 <StatusMessage> [Optional]

3021 A status message describing the status code.

3022 <StatusDetail> [Optional]

3023 Additional status information.

## 3024 5.55 Element <StatusCode>

3025 The <StatusCode> element contains a major status code value and an optional sequence of minor  
3026 status codes.

3027  
3028  
3029  
3030  
3031  
3032  
3033

```
<xs:element name="StatusCode" type="xacml:StatusCodeType"/>
<xs:complexType name="StatusCodeType">
  <xs:sequence>
    <xs:element ref="xacml:StatusCode" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="Value" type="xs:anyURI" use="required"/>
</xs:complexType>
```

3034 The <StatusCode> element is of StatusCodeType complex type.

3035 The <StatusCode> element contains the following attributes and elements:

3036 Value [Required]

3037 See Section B.8 for a list of values.

3038 <StatusCode> [Any Number]

3039 Minor status code. This status code qualifies its parent status code.

## 3040 5.56 Element <StatusMessage>

3041 The <StatusMessage> element is a free-form description of the status code.

3042

```
<xs:element name="StatusMessage" type="xs:string"/>
```

3043 The <StatusMessage> element is of xs:string type.

## 3044 5.57 Element <StatusDetail>

3045 The <StatusDetail> element qualifies the <Status> element with additional information.

3046  
3047  
3048  
3049  
3050  
3051  
3052

```
<xs:element name="StatusDetail" type="xacml:StatusDetailType"/>
<xs:complexType name="StatusDetailType">
  <xs:sequence>
    <xs:any namespace="##any" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

3053 The <StatusDetail> element is of StatusDetailType complex type.

3054 The <StatusDetail> element allows arbitrary XML content.

3055 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the following

3056 XACML-defined <StatusCode> values and includes a <StatusDetail> element, then the following

3057 rules apply.

3058 urn:oasis:names:tc:xacml:1.0:status:ok

3059 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

3060 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

3061 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a

3062 <StatusDetail> element containing one or more <MissingAttributeDetail> elements.

3063 urn:oasis:names:tc:xacml:1.0:status:syntax-error

3064 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status

3065 value. A syntax error may represent either a problem with the **policy** being used or with the request

3066 **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

3067 urn:oasis:names:tc:xacml:1.0:status:processing-error

3068 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error” status

3069 value. This status code indicates an internal problem in the **PDP**. For security reasons, the **PDP** MAY

3070 choose to return no further information to the **PEP**. In the case of a divide-by-zero error or other

3071 computational error, the **PDP** MAY return a <StatusMessage> describing the nature of the error.

## 3072 5.58 Element <MissingAttributeDetail>

3073 The <MissingAttributeDetail> element conveys information about **attributes** required for **policy**

3074 evaluation that were missing from the request **context**.

```

3075 <xs:element name="MissingAttributeDetail"
3076 type="xacml:MissingAttributeDetailType"/>
3077 <xs:complexType name="MissingAttributeDetailType">
3078 <xs:sequence>
3079 <xs:element ref="xacml:AttributeValue" minOccurs="0"
3080 maxOccurs="unbounded"/>
3081 </xs:sequence>
3082 <xs:attribute name="Category" type="xs:anyURI" use="required"/>
3083 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
3084 <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
3085 <xs:attribute name="Issuer" type="xs:string" use="optional"/>
3086 </xs:complexType>

```

3087 The <MissingAttributeDetail> element is of MissingAttributeDetailType complex type.

3088 The <MissingAttributeDetail> element contains the following attributes and elements:

3089 <AttributeValue> [Optional]

3090 The required value of the missing **attribute**.

3091 Category [Required]

3092 The category identifier of the missing **attribute**.

3093 AttributeId [Required]

3094 The identifier of the missing **attribute**.

3095 DataType [Required]

3096 The data-type of the missing **attribute**.

3097 Issuer [Optional]

3098 This attribute, if supplied, SHALL specify the required Issuer of the missing **attribute**.

3099 If the **PDP** includes <AttributeValue> elements in the <MissingAttributeDetail> element, then  
3100 this indicates the acceptable values for that **attribute**. If no <AttributeValue> elements are included,  
3101 then this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list of  
3102 **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the missing  
3103 values or **attributes** will be sufficient to satisfy the **policy**.

---

## 3104 6 XPath 2.0 definitions

3105 The XPath 2.0 specification leaves a number of aspects of behavior implementation defined. This section  
3106 defines how XPath 2.0 SHALL behave when hosted in XACML.

3107 <http://www.w3.org/TR/2007/REC-xpath20-20070123/#id-impl-defined-items> defines the following items:

- 3108 1. The version of Unicode that is used to construct expressions.  
3109 XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.
- 3110 2. The statically-known collations.  
3111 XACML leaves this implementation defined.
- 3112 3. The implicit timezone.  
3113 XACML defined the implicit time zone as UTC.
- 3114 4. The circumstances in which warnings are raised, and the ways in which warnings are handled.  
3115 XACML leaves this implementation defined.
- 3116 5. The method by which errors are reported to the external processing environment.  
3117 An XPath error causes an XACML Indeterminate value in the element where the XPath error  
3118 occurs. The StatusCode value SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-error".  
3119 Implementations MAY provide additional details about the error in the response or by some other  
3120 means.
- 3121 6. Whether the implementation is based on the rules of XML 1.0 or 1.1.  
3122 XACML is based on XML 1.0.
- 3123 7. Whether the implementation supports the namespace axis.  
3124 XACML leaves this implementation defined. It is RECOMMENDED that users of XACML do not  
3125 make use of the namespace axis.
- 3126 8. Any static typing extensions supported by the implementation, if the Static Typing Feature is  
3127 supported.  
3128 XACML leaves this implementation defined.

3129

3130 <http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/#implementation-defined> defines the  
3131 following items:

- 3132 1. Support for additional user-defined or implementation-defined types is implementation-defined.  
3133 It is RECOMMENDED that implementations of XACML do not define any additional types and it is  
3134 RECOMMENDED that users of XACML do not make user of any additional types.
- 3135 2. Some typed values in the data model are undefined. Attempting to access an undefined property  
3136 is always an error. Behavior in these cases is implementation-defined and the host language is  
3137 responsible for determining the result.  
3138 An XPath error causes an XACML Indeterminate value in the element where the XPath error  
3139 occurs. The StatusCode value SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-error".  
3140 Implementations MAY provide additional details about the error in the response or by some other  
3141 means.

3142

3143 <http://www.w3.org/TR/2007/REC-xpath-functions-20070123/#impl-def> defines the following items:

- 3144 1. The destination of the trace output is implementation-defined.  
3145 XACML leaves this implementation defined.
- 3146 2. For xs:integer operations, implementations that support limited-precision integer operations must  
3147 either raise an error [err:FOAR0002] or provide an implementation-defined mechanism that  
3148 allows users to choose between raising an error and returning a result that is modulo the largest  
3149 representable integer value.  
3150 XACML leaves this implementation defined. If an implementation chooses to raise an error, the

- 3151            StatusCode value SHALL be “urn:oasis:names:tc:xacml:1.0:status:processing-error”.
- 3152            Implementations MAY provide additional details about the error in the response or by some other
- 3153            means.
- 3154            3. For xs:decimal values the number of digits of precision returned by the numeric operators is
- 3155            implementation-defined.
- 3156            XACML leaves this implementation defined.
- 3157            4. If the number of digits in the result of a numeric operation exceeds the number of digits that the
- 3158            implementation supports, the result is truncated or rounded in an implementation-defined manner.
- 3159            XACML leaves this implementation defined.
- 3160            5. It is implementation-defined which version of Unicode is supported.
- 3161            XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.
- 3162            6. For fn:normalize-unicode, conforming implementations must support normalization form "NFC"
- 3163            and may support normalization forms "NFD", "NFKC", "NFKD", "FULLY-NORMALIZED". They
- 3164            may also support other normalization forms with implementation-defined semantics.
- 3165            XACML leaves this implementation defined.
- 3166            7. The ability to decompose strings into collation units suitable for substring matching is an
- 3167            implementation-defined property of a collation.
- 3168            XACML leaves this implementation defined.
- 3169            8. All minimally conforming processors must support year values with a minimum of 4 digits (i.e.,
- 3170            YYYY) and a minimum fractional second precision of 1 millisecond or three digits (i.e., s.sss).
- 3171            However, conforming processors may set larger implementation-defined limits on the maximum
- 3172            number of digits they support in these two situations.
- 3173            XACML leaves this implementation defined, and it is RECOMMENDED that users of XACML do
- 3174            not expect greater limits and precision.
- 3175            9. The result of casting a string to xs:decimal, when the resulting value is not too large or too small
- 3176            but nevertheless has too many decimal digits to be accurately represented, is implementation-
- 3177            defined.
- 3178            XACML leaves this implementation defined.
- 3179            10. Various aspects of the processing provided by fn:doc are implementation-defined.
- 3180            Implementations may provide external configuration options that allow any aspect of the
- 3181            processing to be controlled by the user.
- 3182            XACML leaves this implementation defined.
- 3183            11. The manner in which implementations provide options to weaken the stable characteristic of
- 3184            fn:collection and fn:doc are implementation-defined.
- 3185            XACML leaves this implementation defined.

---

## 3186 7 Functional requirements

3187 This section specifies certain functional requirements that are not directly associated with the production  
3188 or consumption of a particular XACML element.

3189 Note that in each case an implementation is conformant as long as it produces the same result as is  
3190 specified here, regardless of how and in what order the implementation behaves internally.

### 3191 7.1 Unicode issues

#### 3192 7.1.1 Normalization

3193 In Unicode, some equivalent characters can be represented by more than one different Unicode  
3194 character sequence. See [CMF]. The process of converting Unicode strings into equivalent character  
3195 sequences is called "normalization" [UAX15]. Some operations, such as string comparison, are sensitive  
3196 to normalization. An operation is normalization-sensitive if its output(s) are different depending on the  
3197 state of normalization of the input(s); if the output(s) are textual, they are deemed different only if they  
3198 would remain different were they to be normalized.

3199 For more information on normalization see [CM].

3200 An XACML implementation MUST behave as if each normalization-sensitive operation normalizes input  
3201 strings into Unicode Normalization Form C ("NFC"). An implementation MAY use some other form of  
3202 internal processing (such as using a non-Unicode, "legacy" character encoding) as long as the externally  
3203 visible results are identical to this specification.

#### 3204 7.1.2 Version of Unicode

3205 The version of Unicode used by XACML is implementation defined. It is RECOMMENDED that the latest  
3206 version is used. Also note security issues in section 9.3.

### 3207 7.2 Policy enforcement point

3208 This section describes the requirements for the *PEP*.

3209 An application functions in the role of the *PEP* if it guards *access* to a set of *resources* and asks the  
3210 *PDP* for an *authorization decision*. The *PEP* MUST abide by the *authorization decision* as described  
3211 in one of the following sub-sections

3212 In any case any *advice* in the *decision* may be safely ignored by the *PEP*.

#### 3213 7.2.1 Base PEP

3214 If the *decision* is "Permit", then the *PEP* SHALL permit *access*. If *obligations* accompany the *decision*,  
3215 then the *PEP* SHALL permit *access* only if it understands and it can and will discharge those  
3216 *obligations*.

3217 If the *decision* is "Deny", then the *PEP* SHALL deny *access*. If *obligations* accompany the *decision*,  
3218 then the *PEP* shall deny *access* only if it understands, and it can and will discharge those *obligations*.

3219 If the *decision* is "Not Applicable", then the *PEP*'s behavior is undefined.

3220 If the *decision* is "Indeterminate", then the *PEP*'s behavior is undefined.

#### 3221 7.2.2 Deny-biased PEP

3222 If the *decision* is "Permit", then the *PEP* SHALL permit *access*. If *obligations* accompany the *decision*,  
3223 then the *PEP* SHALL permit *access* only if it understands and it can and will discharge those  
3224 *obligations*.

3225 All other **decisions** SHALL result in the denial of **access**.

3226 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of  
3227 the **decision request**, etc., are not prohibited.

### 3228 7.2.3 Permit-biased PEP

3229 If the **decision** is "Deny", then the **PEP** SHALL deny **access**. If **obligations** accompany the **decision**,  
3230 then the **PEP** shall deny **access** only if it understands, and it can and will discharge those **obligations**.

3231 All other **decisions** SHALL result in the permission of **access**.

3232 Note: other actions, e.g. consultation of additional **PDPs**, reformulation/resubmission of  
3233 the **decision request**, etc., are not prohibited.

## 3234 7.3 Attribute evaluation

3235 **Attributes** are represented in the request **context** by the **context handler**, regardless of whether or not  
3236 they appeared in the original **decision request**, and are referred to in the **policy** by attribute designators  
3237 and attribute selectors. A **named attribute** is the term used for the criteria that the specific attribute  
3238 designators use to refer to particular **attributes** in the <Attributes> elements of the request **context**.

### 3239 7.3.1 Structured attributes

3240 <AttributeValue> elements MAY contain an instance of a structured XML data-type, for example  
3241 <ds:KeyInfo>. XACML 3.0 supports several ways for comparing the contents of such elements.

3242 1. In some cases, such elements MAY be compared using one of the XACML string functions, such  
3243 as "string-regexp-match", described below. This requires that the element be given the data-type  
3244 "http://www.w3.org/2001/XMLSchema#string". For example, a structured data-type that is  
3245 actually a ds:KeyInfo/KeyName would appear in the **Context** as:

```
3246 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3247 <ds:KeyName>jhibbert-key</ds:KeyName>  
3248 </AttributeValue>
```

3249 In general, this method will not be adequate unless the structured data-type is quite simple.

3250 2. The structured **attribute** MAY be made available in the <Content> element of the appropriate  
3251 **attribute** category and an <AttributeSelector> element MAY be used to select the contents  
3252 of a leaf sub-element of the structured data-type by means of an XPath expression. That value  
3253 MAY then be compared using one of the supported XACML functions appropriate for its primitive  
3254 data-type. This method requires support by the **PDP** for the optional XPath expressions feature.

3255 3. The structured **attribute** MAY be made available in the <Content> element of the appropriate  
3256 **attribute** category and an <AttributeSelector> element MAY be used to select any node in  
3257 the structured data-type by means of an XPath expression. This node MAY then be compared  
3258 using one of the XPath-based functions described in Section A.3.15. This method requires  
3259 support by the **PDP** for the optional XPath expressions and XPath functions features.

3260 See also Section 7.3.

### 3261 7.3.2 Attribute bags

3262 XACML defines implicit collections of its data-types. XACML refers to a collection of values that are of a  
3263 single data-type as a **bag**. **Bags** of data-types are needed because selections of nodes from an XML  
3264 **resource** or XACML request **context** may return more than one value.

3265 The <AttributeSelector> element uses an XPath expression to specify the selection of data from  
3266 free form XML. The result of an XPath expression is termed a node-set, which contains all the nodes  
3267 from the XML content that match the **predicate** in the XPath expression. Based on the various indexing  
3268 functions provided in the XPath specification, it SHALL be implied that a resultant node-set is the

3269 collection of the matching nodes. XACML also defines the <AttributeDesignator> element to have  
3270 the same matching methodology for **attributes** in the XACML request **context**.

3271 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be no  
3272 notion of a **bag** containing **bags**, or a **bag** containing values of differing types; i.e., a **bag** in XACML  
3273 SHALL contain only values that are of the same data-type.

### 3274 7.3.3 Multivalued attributes

3275 If a single <Attribute> element in a request **context** contains multiple <AttributeValue> child  
3276 elements, then the **bag** of values resulting from evaluation of the <Attribute> element MUST be  
3277 identical to the **bag** of values that results from evaluating a **context** in which each <AttributeValue>  
3278 element appears in a separate <Attribute> element, each carrying identical meta-data.

### 3279 7.3.4 Attribute Matching

3280 A **named attribute** includes specific criteria with which to match **attributes** in the **context**. An **attribute**  
3281 specifies a Category, AttributeId and DataType, and a **named attribute** also specifies the  
3282 Issuer. A **named attribute** SHALL match an **attribute** if the values of their respective Category,  
3283 AttributeId, DataType and optional Issuer attributes match. The Category of the **named**  
3284 **attribute** MUST match, by **identifier equality**, the Category of the corresponding **context attribute**.  
3285 The AttributeId of the **named attribute** MUST match, by **identifier equality**, the AttributeId of  
3286 the corresponding **context attribute**. The DataType of the **named attribute** MUST match, by **identifier**  
3287 **equality**, the DataType of the corresponding **context attribute**. If Issuer is supplied in the **named**  
3288 **attribute**, then it MUST match, using the urn:oasis:names:tc:xacml:1.0:function:string-equal function, the  
3289 Issuer of the corresponding **context attribute**. If Issuer is not supplied in the **named attribute**, then  
3290 the matching of the **context attribute** to the **named attribute** SHALL be governed by AttributeId and  
3291 DataType alone, regardless of the presence, absence, or actual value of Issuer in the corresponding  
3292 **context attribute**. In the case of an attribute selector, the matching of the **attribute** to the **named**  
3293 **attribute** SHALL be governed by the XPath expression and DataType.

### 3294 7.3.5 Attribute Retrieval

3295 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**. The  
3296 **context handler** MAY also add **attributes** to the request **context** without the **PDP** requesting them. The  
3297 **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document, but the  
3298 **context handler** is responsible for obtaining and supplying the requested values by whatever means it  
3299 deems appropriate, including by retrieving them from one or more Policy Information Points. The **context**  
3300 **handler** SHALL return the values of **attributes** that match the attribute designator or attribute selector  
3301 and form them into a **bag** of values with the specified data-type. If no **attributes** from the request  
3302 **context** match, then the **attribute** SHALL be considered missing. If the **attribute** is missing, then  
3303 MustBePresent governs whether the attribute designator or attribute selector returns an empty **bag** or  
3304 an “Indeterminate” result. If MustBePresent is “False” (default value), then a missing **attribute** SHALL  
3305 result in an empty **bag**. If MustBePresent is “True”, then a missing **attribute** SHALL result in  
3306 “Indeterminate”. This “Indeterminate” result SHALL be handled in accordance with the specification of the  
3307 encompassing expressions, **rules**, **policies** and **policy sets**. If the result is “Indeterminate”, then the  
3308 AttributeId, DataType and Issuer of the **attribute** MAY be listed in the **authorization decision** as  
3309 described in Section 7.17. However, a **PDP** MAY choose not to return such information for security  
3310 reasons.

3311 Regardless of any dynamic modifications of the request **context** during policy evaluation, the **PDP**  
3312 SHALL behave as if each bag of **attribute** values is fully populated in the **context** before it is first tested,  
3313 and is thereafter immutable during evaluation. (That is, every subsequent test of that **attribute** shall use  
3314 the same bag of values that was initially tested.)

## 3315 7.3.6 Environment Attributes

3316 Standard *environment attributes* are listed in Section B.7. If a value for one of these *attributes* is  
3317 supplied in the *decision request*, then the *context handler* SHALL use that value. Otherwise, the  
3318 *context handler* SHALL supply a value. In the case of date and time *attributes*, the supplied value  
3319 SHALL have the semantics of the "date and time that apply to the *decision request*".

## 3320 7.3.7 AttributeSelector evaluation

3321 An <AttributeSelector> element will be evaluated according to the following processing model.

3322

3323 NOTE: It is not necessary for an implementation to actually follow these steps. It is only  
3324 necessary to produce results identical to those that would be produced by following these  
3325 steps.

- 3326 1. Construct an XML data structure suitable for xpath processing from the <Content> element in  
3327 the *attributes* category given by the *Category* attribute. The data structure shall be constructed  
3328 so that the document node of this structure contains a single document element which  
3329 corresponds to the single child element of the <Content> element. The constructed data  
3330 structure shall be equivalent to one that would result from parsing a stand-alone XML document  
3331 consisting of the contents of the <Content> element (including any comment and processing-  
3332 instruction markup). Namespace declarations which are not "visibly utilized", as defined by [exc-  
3333 c14n], MAY not be present and MUST NOT be utilized by the XPath expression in step 3. The  
3334 data structure must meet the requirements of the applicable xpath version.
- 3335 2. Select a context node for xpath processing from this data structure. If there is a  
3336 *ContextSelectorId* attribute, the context node shall be the node selected by applying the  
3337 XPath expression given in the *attribute* value of the designated *attribute* (in the *attributes*  
3338 category given by the <AttributeSelector> *Category* attribute). It shall be an error if this  
3339 evaluation returns no node or more than one node, in which case the return value MUST be an  
3340 "Indeterminate" with a status code "urn:oasis:names:tc:xacml:1.0:status:syntax-error". If there is  
3341 no *ContextSelectorId*, the document node of the data structure shall be the context node.
- 3342 3. Evaluate the XPath expression given in the *Path* attribute against the xml data structure, using  
3343 the context node selected in the previous step. It shall be an error if this evaluation returns  
3344 anything other than a sequence of nodes (possibly empty), in which case the  
3345 <AttributeSelector> MUST return "Indeterminate" with a status code  
3346 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".
- 3347 4. If the data type is a primitive data type, convert the text value of each selected node to the  
3348 desired data type, as specified in the *DataType* attribute. Each value shall be constructed using  
3349 the appropriate constructor function from [XF] Section 5 listed below, corresponding to the  
3350 specified data type.

3351  
3352 xs:string()  
3353 xs:boolean()  
3354 xs:integer()  
3355 xs:double()  
3356 xs:dateTime()  
3357 xs:date()  
3358 xs:time()  
3359 xs:hexBinary()  
3360 xs:base64Binary()  
3361 xs:anyURI()  
3362 xs:yearMonthDuration()  
3363 xs:dayTimeDuration()  
3364

3365 If the *DataType* is not one of the primitive types listed above, then the return values shall be

3366 constructed from the nodeset in a manner specified by the particular `DataType` extension  
3367 specification. If the data type extension does not specify an appropriate constructor function, then  
3368 the `<AttributeSelector>` MUST return "Indeterminate" with a status code  
3369 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".  
3370

3371 If an error occurs when converting the values returned by the XPath expression to the specified  
3372 `DataType`, then the result of the `<AttributeSelector>` MUST be "Indeterminate", with a  
3373 status code "urn:oasis:names:tc:xacml:1.0:status:processing-error"

## 3374 7.4 Expression evaluation

3375 XACML specifies expressions in terms of the elements listed below, of which the `<Apply>` and  
3376 `<Condition>` elements recursively compose greater expressions. Valid expressions SHALL be type  
3377 correct, which means that the types of each of the elements contained within `<Apply>` elements SHALL  
3378 agree with the respective argument types of the function that is named by the `FunctionId` attribute.  
3379 The resultant type of the `<Apply>` element SHALL be the resultant type of the function, which MAY be  
3380 narrowed to a primitive data-type, or a **bag** of a primitive data-type, by type-unification. XACML defines  
3381 an evaluation result of "Indeterminate", which is said to be the result of an invalid expression, or an  
3382 operational error occurring during the evaluation of the expression.

3383 XACML defines these elements to be in the substitution group of the `<Expression>` element:

- 3384 • `<xacml:AttributeValue>`
- 3385 • `<xacml:AttributeDesignator>`
- 3386 • `<xacml:AttributeSelector>`
- 3387 • `<xacml:Apply>`
- 3388 • `<xacml:Function>`
- 3389 • `<xacml:VariableReference>`

## 3390 7.5 Arithmetic evaluation

3391 IEEE 754 [IEEE754] specifies how to evaluate arithmetic functions in a context, which specifies defaults  
3392 for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all integer and  
3393 double functions relying on the Extended Default Context, enhanced with double precision:

3394 flags - all set to 0

3395 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap enabler,  
3396 which SHALL be set to 1

3397 precision - is set to the designated double precision

3398 rounding - is set to round-half-even (IEEE 854 §4.1)

## 3399 7.6 Match evaluation

3400 The **attribute** matching element `<Match>` appears in the `<Target>` element of **rules**, **policies** and  
3401 **policy sets**.

3402 This element represents a Boolean expression over **attributes** of the request **context**. A matching  
3403 element contains a `MatchId` attribute that specifies the function to be used in performing the match  
3404 evaluation, an `<AttributeValue>` and an `<AttributeDesignator>` or `<AttributeSelector>`  
3405 element that specifies the **attribute** in the **context** that is to be matched against the specified value.

3406 The `MatchId` attribute SHALL specify a function that takes two arguments, returning a result type of  
3407 "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the matching element  
3408 SHALL be supplied to the `MatchId` function as its first argument. An element of the **bag** returned by the  
3409 `<AttributeDesignator>` or `<AttributeSelector>` element SHALL be supplied to the `MatchId`  
3410 function as its second argument, as explained below. The `DataType` of the `<AttributeValue>`

3411 SHALL match the data-type of the first argument expected by the `MatchId` function. The `DataType` of  
3412 the `<AttributeDesignator>` or `<AttributeSelector>` element SHALL match the data-type of the  
3413 second argument expected by the `MatchId` function.

3414 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the  
3415 `MatchId` attribute, and those functions MAY use data-types that are also extensions, so long as the  
3416 extension function returns a Boolean result and takes two single base types as its inputs. The function  
3417 used as the value for the `MatchId` attribute SHOULD be easily indexable. Use of non-indexable or  
3418 complex functions may prevent efficient evaluation of **decision requests**.

3419 The evaluation semantics for a matching element is as follows. If an operational error were to occur while  
3420 evaluating the `<AttributeDesignator>` or `<AttributeSelector>` element, then the result of the  
3421 entire expression SHALL be "Indeterminate". If the `<AttributeDesignator>` or  
3422 `<AttributeSelector>` element were to evaluate to an empty **bag**, then the result of the expression  
3423 SHALL be "False". Otherwise, the `MatchId` function SHALL be applied between the  
3424 `<AttributeValue>` and each element of the **bag** returned from the `<AttributeDesignator>` or  
3425 `<AttributeSelector>` element. If at least one of those function applications were to evaluate to  
3426 "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the function  
3427 applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally, if all function  
3428 applications evaluate to "False", then the result of the entire expression SHALL be "False".

3429 It is also possible to express the semantics of a **target** matching element in a **condition**. For instance,  
3430 the **target** match expression that compares a "**subject-name**" starting with the name "John" can be  
3431 expressed as follows:

```
3432 <Match  
3433 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">  
3434   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3435     John.*  
3436   </AttributeValue>  
3437   <AttributeDesignator  
3438     Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-  
3439 subject"  
3440     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3441     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3442 </Match>
```

3443 Alternatively, the same match semantics can be expressed as an `<Apply>` element in a **condition** by  
3444 using the "urn:oasis:names:tc:xacml:3.0:function:any-of" function, as follows:

```
3445 <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">  
3446   <Function  
3447   FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match"/>  
3448   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3449     John.*  
3450   </AttributeValue>  
3451   <AttributeDesignator  
3452     Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-  
3453 subject"  
3454     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3455     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3456 </Apply>
```

## 3457 7.7 Target evaluation

3458 An empty **target** matches any request. Otherwise the **target** value SHALL be "Match" if all the AnyOf  
3459 specified in the **target** match values in the request **context**. Otherwise, if any one of the AnyOf specified  
3460 in the **target** is "No Match", then the **target** SHALL be "No Match". Otherwise, the **target** SHALL be  
3461 "Indeterminate". The **target** match table is shown in Table 1.

<AnyOf> values	Target value
----------------	--------------

All "Match"	"Match"
At least one "No Match"	"No Match"
Otherwise	"Indeterminate"

3462 Table 1 Target match table

3463 The AnyOf SHALL match values in the request **context** if at least one of their <AllOf> elements  
 3464 matches a value in the request **context**. The AnyOf table is shown in Table 2.

<AllOf> values	<AnyOf> Value
At least one "Match"	"Match"
None matches and at least one "Indeterminate"	"Indeterminate"
All "No match"	"No match"

3465 Table 2 AnyOf match table

3466 An AllOf SHALL match a value in the request **context** if the value of all its <Match> elements is "True".  
 3467 The AllOf table is shown in Table 3.

<Match> values	<AllOf> Value
All "True"	"Match"
No "False" and at least one "Indeterminate"	"Indeterminate"
At least one "False"	"No match"

3468 Table 3 AllOf match table

## 3469 7.8 VariableReference Evaluation

3470 The <VariableReference> element references a single <VariableDefinition> element contained  
 3471 within the same <Policy> element. A <VariableReference> that does not reference a particular  
 3472 <VariableDefinition> element within the encompassing <Policy> element is called an undefined  
 3473 reference. **Policies** with undefined references are invalid.

3474 In any place where a <VariableReference> occurs, it has the effect as if the text of the  
 3475 <Expression> element defined in the <VariableDefinition> element replaces the  
 3476 <VariableReference> element. Any evaluation scheme that preserves this semantic is acceptable.  
 3477 For instance, the expression in the <VariableDefinition> element may be evaluated to a particular  
 3478 value and cached for multiple references without consequence. (I.e. the value of an <Expression>  
 3479 element remains the same for the entire **policy** evaluation.) This characteristic is one of the benefits of  
 3480 XACML being a declarative language.

3481 A variable reference containing circular references is invalid. The PDP MUST detect circular references  
 3482 either at policy loading time or during runtime evaluation. If the PDP detects a circular reference during  
 3483 runtime the variable reference evaluates to "Indeterminate" with status code  
 3484 urn:oasis:names:tc:xacml:1.0:status:processing-error.

## 3485 7.9 Condition evaluation

3486 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to "True".  
 3487 Its value SHALL be "False" if the <Condition> element evaluates to "False". The **condition** value

3488 SHALL be "Indeterminate", if the expression contained in the <Condition> element evaluates to  
 3489 "Indeterminate."

## 3490 7.10 Extended Indeterminate

3491 Some **combining algorithms** are defined in terms of an extended set of "Indeterminate" values. The  
 3492 extended set associated with the "Indeterminate" contains the potential effect values which could have  
 3493 occurred if there would not have been an error causing the "Indeterminate". The possible extended set  
 3494 "Indeterminate" values are

- 3495 • "Indeterminate{D}": an "Indeterminate" from a **policy** or **rule** which could have evaluated to "Deny",  
 3496 but not "Permit"
- 3497 • "Indeterminate{P}": an "Indeterminate" from a **policy** or **rule** which could have evaluated to "Permit",  
 3498 but not "Deny"
- 3499 • "Indeterminate{DP}": an "Indeterminate" from a **policy** or **rule** which could have evaluated to "Deny"  
 3500 or "Permit".

3501 The **combining algorithms** which are defined in terms of the extended "Indeterminate" make use of the  
 3502 additional information to allow for better treatment of errors in the algorithms.

3503 The final decision returned by a **PDP** cannot be an extended Indeterminate. Any such decision at the top  
 3504 level **policy** or **policy set** is returned as a plain Indeterminate in the response from the **PDP**.

3505 The tables in the following four sections define how extended "Indeterminate" values are produced during  
 3506 **Rule**, **Policy** and **PolicySet** evaluation.

## 3507 7.11 Rule evaluation

3508 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves separate  
 3509 evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 4.

<b>Target</b>	Condition	<b>Rule</b> Value
"Match" or no target	"True"	<b>Effect</b>
"Match" or no target	"False"	"NotApplicable"
"Match" or no target	"Indeterminate"	"Indeterminate{P}" if the <b>Effect</b> is Permit, or "Indeterminate{D}" if the <b>Effect</b> is Deny
"No-match"	Don't care	"NotApplicable"
"Indeterminate"	Don't care	"Indeterminate{P}" if the <b>Effect</b> is Permit, or "Indeterminate{D}" if the <b>Effect</b> is Deny

3510 Table 4 Rule truth table.

## 3511 7.12 Policy evaluation

3512 The value of a **policy** SHALL be determined only by its contents, considered in relation to the contents of  
 3513 the request **context**. A **policy's** value SHALL be determined by evaluation of the **policy's target** and,  
 3514 according to the specified **rule-combining algorithm, rules**,.

3515 The **policy** truth table is shown in Table 5.

<b>Target</b>	<b>Rule values</b>	<b>Policy Value</b>
“Match”	Don’t care	Specified by the <b>rule-combining algorithm</b>
“No-match”	Don’t care	“NotApplicable”
“Indeterminate”	See Table 7	See Table 7

3516 Table 5 Policy truth table

3517 Note that none of the **rule-combining algorithms** defined by XACML 3.0 take parameters. However,  
 3518 non-standard combining algorithms MAY take parameters. In such a case, the values of these  
 3519 parameters associated with the **rules**, MUST be taken into account when evaluating the **policy**. The  
 3520 parameters and their types should be defined in the specification of the combining algorithm. If the  
 3521 implementation supports combiner parameters and if combiner parameters are present in a **policy**, then  
 3522 the parameter values MUST be supplied to the combining algorithm implementation.

### 3523 7.13 Policy Set evaluation

3524 The value of a **policy set** SHALL be determined by its contents, considered in relation to the contents of  
 3525 the request **context**. A **policy set**'s value SHALL be determined by evaluation of the **policy set**'s **target**,  
 3526 and, according to the specified **policy-combining algorithm**, **policies** and **policy sets**,

3527 The **policy set** truth table is shown in Table 6.

<b>Target</b>	<b>Policy values</b>	<b>Policy set Value</b>
“Match”	Don’t care	Specified by the <b>policy-combining algorithm</b>
“No-match”	Don’t care	“NotApplicable”
“Indeterminate”	See Table 7	See Table 7

3528 Table 6 Policy set truth table

3529 Note that none of the **policy-combining algorithms** defined by XACML 3.0 take parameters. However,  
 3530 non-standard combining algorithms MAY take parameters. In such a case, the values of these  
 3531 parameters associated with the **policies**, MUST be taken into account when evaluating the **policy set**.  
 3532 The parameters and their types should be defined in the specification of the combining algorithm. If the  
 3533 implementation supports combiner parameters and if combiner parameters are present in a **policy**, then  
 3534 the parameter values MUST be supplied to the combining algorithm implementation.

### 3535 7.14 Policy and Policy set value for Indeterminate Target

3536 If the **target** of a **policy** or **policy set** evaluates to “Indeterminate”, the value of the **policy** or **policy set**  
 3537 as a whole is determined by the value of the **combining algorithm** according to Table 7.

<b>Combining algorithm Value</b>	<b>Policy set or policy Value</b>
“NotApplicable”	“NotApplicable”
“Permit”	“Indeterminate{P}”
“Deny”	“Indeterminate{D}”
“Indeterminate”	“Indeterminate{DP}”
“Indeterminate{DP}”	“Indeterminate{DP}”
“Indeterminate{P}”	“Indeterminate{P}”

"Indeterminate{D}"
--------------------

"Indeterminate{D}"
--------------------

3538 Table 7 The value of a **policy** or **policy set** when the target is "Indeterminate".

## 3539 7.15 PolicySetIdReference and PolicyIdReference evaluation

3540 A policy set id reference or a policy id reference is evaluated by resolving the reference and evaluating  
3541 the referenced policy set or policy.

3542 If resolving the reference fails, the reference evaluates to "Indeterminate" with status code  
3543 urn:oasis:names:tc:xacml:1.0:status:processing-error.

3544 A policy set id reference or a policy id reference containing circular references is invalid. The PDP MUST  
3545 detect circular references either at policy loading time or during runtime evaluation. If the PDP detects a  
3546 circular reference during runtime the reference evaluates to "Indeterminate" with status code  
3547 urn:oasis:names:tc:xacml:1.0:status:processing-error.

## 3548 7.16 Hierarchical resources

3549 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document). XACML  
3550 provides several optional mechanisms for supporting hierarchical **resources**. These are described in the  
3551 XACML Profile for Hierarchical Resources [**Hier**] and in the XACML Profile for Requests for Multiple  
3552 Resources [**Multi**].

## 3553 7.17 Authorization decision

3554 In relation to a particular **decision request**, the **PDP** is defined by a **policy-combining algorithm** and a  
3555 set of **policies** and/or **policy sets**. The **PDP** SHALL return a response **context** as if it had evaluated a  
3556 single **policy set** consisting of this **policy-combining algorithm** and the set of **policies** and/or **policy**  
3557 **sets**.

3558 The **PDP** MUST evaluate the **policy set** as specified in Sections 5 and 7. The **PDP** MUST return a  
3559 response **context**, with one <Decision> element of value "Permit", "Deny", "Indeterminate" or  
3560 "NotApplicable".

3561 If the **PDP** cannot make a **decision**, then an "Indeterminate" <Decision> element SHALL be returned.

## 3562 7.18 Obligations and advice

3563 A **rule**, **policy**, or **policy set** may contain one or more **obligation** or **advice** expressions. When such a  
3564 **rule**, **policy**, or **policy set** is evaluated, the **obligation** or **advice** expression SHALL be evaluated to an  
3565 **obligation** or **advice** respectively, which SHALL be passed up to the next level of evaluation (the  
3566 enclosing or referencing **policy**, **policy set**, or **authorization decision**) only if the result of the **rule**,  
3567 **policy**, or **policy set** being evaluated matches the value of the FulfillOn attribute of the **obligation** or  
3568 the AppliesTo attribute of the **advice**. If any of the **attribute** assignment expressions in an **obligation**  
3569 or **advice** expression with a matching FulfillOn or AppliesTo attribute evaluates to "Indeterminate",  
3570 then the whole **rule**, **policy**, or **policy set** SHALL be "Indeterminate". If the FulfillOn or AppliesTo  
3571 attribute does not match the result of the combining algorithm or the **rule** evaluation, then any  
3572 indeterminate in an **obligation** or **advice** expression has no effect.

3573 As a consequence of this procedure, no **obligations** or **advice** SHALL be returned to the **PEP** if the **rule**,  
3574 **policies**, or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is  
3575 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **rule**, **policy**, or **policy**  
3576 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3577 If the **PDP**'s evaluation is viewed as a tree of **rules**, **policy sets** and **policies**, each of which returns  
3578 "Permit" or "Deny", then the set of **obligations** and **advice** returned by the **PDP** to the **PEP** will include  
3579 only the **obligations** and **advice** associated with those paths where the result at each level of evaluation  
3580 is the same as the result being returned by the **PDP**. In situations where any lack of determinism is  
3581 unacceptable, a deterministic combining algorithm, such as ordered-deny-overrides, should be used.

3582 Also see Section 7.2.

## 3583 7.19 Exception handling

3584 XACML specifies behavior for the **PDP** in the following situations.

### 3585 7.19.1 Unsupported functionality

3586 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or function  
3587 that the **PDP** does not support, then the **PDP** SHALL return a <Decision> value of "Indeterminate". If a  
3588 <StatusCode> element is also returned, then its value SHALL be  
3589 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and  
3590 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported function.

### 3591 7.19.2 Syntax and type errors

3592 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision request** is  
3593 received, then the result of that **policy** SHALL be "Indeterminate" with a `StatusCode` value of  
3594 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3595 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a **decision**  
3596 **request** is received, then the result of that **policy** SHALL be "Indeterminate" with a `StatusCode` value of  
3597 "urn:oasis:names:tc:xacml:1.0:status:processing-error".

### 3598 7.19.3 Missing attributes

3599 The absence of matching **attributes** in the request **context** for any of the attribute designators attribute or  
3600 selectors that are found in the **policy** will result in an enclosing <AllOf> element to return a value of  
3601 "Indeterminate", if the designator or selector has the `MustBePresent` XML attribute set to true, as  
3602 described in Sections 5.29 and 5.30 and may result in a <Decision> element containing the  
3603 "Indeterminate" value. If, in this case a status code is supplied, then the value

3604 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3605 SHALL be used, to indicate that more information is needed in order for a definitive **decision** to be  
3606 rendered. In this case, the <Status> element MAY list the names and data-types of any **attributes** that  
3607 are needed by the **PDP** to refine its **decision** (see Section 5.58). A **PEP** MAY resubmit a refined request  
3608 **context** in response to a <Decision> element contents of "Indeterminate" with a status code of

3609 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3610 by adding **attribute** values for the **attribute** names that were listed in the previous response. When the  
3611 **PDP** returns a <Decision> element contents of "Indeterminate", with a status code of

3612 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute",

3613 it MUST NOT list the names and data-types of any **attribute** for which values were supplied in the original  
3614 request. Note, this requirement forces the **PDP** to eventually return an **authorization decision** of  
3615 "Permit", "Deny", or "Indeterminate" with some other status code, in response to successively-refined  
3616 requests.

## 3617 7.20 Identifier equality

3618 XACML makes use of URIs and strings as identifiers. When such identifiers are compared for equality,  
3619 the comparison MUST be done so that the identifiers are equal if they have the same length and the  
3620 characters in the two identifiers are equal codepoint by codepoint.

3621 The following is a list of the identifiers which MUST use this definition of equality.

3622 The content of the element <XPathVersion>.

3623 The XML attribute `Value` in the element <StatusCode>.

- 3624 The XML attributes `Category`, `AttributeId`, `DataType` and `Issuer` in the element  
3625 `<MissingAttributeDetail>`.
- 3626 The XML attribute `Category` in the element `<Attributes>`.
- 3627 The XML attributes `AttributeId` and `Issuer` in the element `<Attribute>`.
- 3628 The XML attribute `ObligationId` in the element `<Obligation>`.
- 3629 The XML attribute `AdviceId` in the element `<Advice>`.
- 3630 The XML attributes `AttributeId` and `Category` in the element `<AttributeAssignment>`.
- 3631 The XML attribute `ObligationId` in the element `<ObligationExpression>`.
- 3632 The XML attribute `AdviceId` in the element `<AdviceExpression>`.
- 3633 The XML attributes `AttributeId`, `Category` and `Issuer` in the element  
3634 `<AttributeAssignmentExpression>`.
- 3635 The XML attributes `PolicySetId` and `PolicyCombiningAlgId` in the element `<PolicySet>`.
- 3636 The XML attribute `ParameterName` in the element `<CombinerParameter>`.
- 3637 The XML attribute `RuleIdRef` in the element `<RuleCombinerParameters>`.
- 3638 The XML attribute `PolicyIdRef` in the element `<PolicyCombinerParameters>`.
- 3639 The XML attribute `PolicySetIdRef` in the element `<PolicySetCombinerParameters>`.
- 3640 The anyURI in the content of the complex type `IdReferenceType`.
- 3641 The XML attributes `PolicyId` and `RuleCombiningAlgId` in the element `<Policy>`.
- 3642 The XML attribute `RuleId` in the element `<Rule>`.
- 3643 The XML attribute `MatchId` in the element `<Match>`.
- 3644 The XML attribute `VariableId` in the element `<VariableDefinition>`.
- 3645 The XML attribute `VariableId` in the element `<VariableReference>`.
- 3646 The XML attributes `Category`, `ContextSelectorId` and `DataType` in the element  
3647 `<AttributeSelector>`.
- 3648 The XML attributes `Category`, `AttributeId`, `DataType` and `Issuer` in the element  
3649 `<AttributeDesignator>`.
- 3650 The XML attribute `DataType` in the element `<AttributeValue>`.
- 3651 The XML attribute `FunctionId` in the element `<Function>`.
- 3652 The XML attribute `FunctionId` in the element `<Apply>`.
- 3653
- 3654 It is RECOMMENDED that extensions to XACML use the same definition of identifier equality for similar  
3655 identifiers.
- 3656 It is RECOMMENDED that extensions which define identifiers do not define identifiers which could be  
3657 easily misinterpreted by people as being subject to other kind of processing, such as URL character  
3658 escaping, before matching.

---

## 3659 8 XACML extensibility points (non-normative)

3660 This section describes the points within the XACML model and schema where extensions can be added.

### 3661 8.1 Extensible XML attribute types

3662 The following XML attributes have values that are URIs. These may be extended by the creation of new  
3663 URIs associated with new semantics for these attributes.

3664 `Category,`

3665 `AttributeId,`

3666 `DataType,`

3667 `FunctionId,`

3668 `MatchId,`

3669 `ObligationId,`

3670 `AdviceId,`

3671 `PolicyCombiningAlgId,`

3672 `RuleCombiningAlgId,`

3673 `StatusCode,`

3674 `SubjectCategory.`

3675 See Section 5 for definitions of these *attribute* types.

### 3676 8.2 Structured attributes

3677 <AttributeValue> elements MAY contain an instance of a structured XML data-type. Section 7.3.1  
3678 describes a number of standard techniques to identify data items within such a structured *attribute*.  
3679 Listed here are some additional techniques that require XACML extensions.

- 3680 1. For a given structured data-type, a community of XACML users MAY define new *attribute*  
3681 identifiers for each leaf sub-element of the structured data-type that has a type conformant with  
3682 one of the XACML-defined primitive data-types. Using these new *attribute* identifiers, the *PEPs*  
3683 or *context handlers* used by that community of users can flatten instances of the structured  
3684 data-type into a sequence of individual <Attribute> elements. Each such <Attribute>  
3685 element can be compared using the XACML-defined functions. Using this method, the structured  
3686 data-type itself never appears in an <AttributeValue> element.
- 3687 2. A community of XACML users MAY define a new function that can be used to compare a value of  
3688 the structured data-type against some other value. This method may only be used by *PDPs* that  
3689 support the new function.

---

## 3690 9 Security and privacy considerations (non- 3691 normative)

3692 This section identifies possible security and privacy compromise scenarios that should be considered  
3693 when implementing an XACML-based system. The section is informative only. It is left to the  
3694 implementer to decide whether these compromise scenarios are practical in their environment and to  
3695 select appropriate safeguards.

### 3696 9.1 Threat model

3697 We assume here that the adversary has access to the communication channel between the XACML  
3698 actors and is able to interpret, insert, delete, and modify messages or parts of messages.

3699 Additionally, an actor may use information from a former message maliciously in subsequent transactions.  
3700 It is further assumed that *rules* and *policies* are only as reliable as the actors that create and use them.  
3701 Thus it is incumbent on each actor to establish appropriate trust in the other actors upon which it relies.  
3702 Mechanisms for trust establishment are outside the scope of this specification.

3703 The messages that are transmitted between the actors in the XACML model are susceptible to attack by  
3704 malicious third parties. Other points of vulnerability include the *PEP*, the *PDP*, and the *PAP*. While some  
3705 of these entities are not strictly within the scope of this specification, their compromise could lead to the  
3706 compromise of *access control* enforced by the *PEP*.

3707 It should be noted that there are other components of a distributed system that may be compromised,  
3708 such as an operating system and the domain-name system (DNS) that are outside the scope of this  
3709 discussion of threat models. Compromise in these components may also lead to a policy violation.

3710 The following sections detail specific compromise scenarios that may be relevant to an XACML system.

#### 3711 9.1.1 Unauthorized disclosure

3712 XACML does not specify any inherent mechanisms to protect the confidentiality of the messages  
3713 exchanged between actors. Therefore, an adversary could observe the messages in transit. Under  
3714 certain security *policies*, disclosure of this information is a violation. Disclosure of *attributes* or the types  
3715 of *decision requests* that a *subject* submits may be a breach of privacy policy. In the commercial  
3716 sector, the consequences of unauthorized disclosure of personal data may range from embarrassment to  
3717 the custodian, to imprisonment and/or large fines in the case of medical or financial data.

3718 Unauthorized disclosure is addressed by confidentiality safeguards.

#### 3719 9.1.2 Message replay

3720 A message replay attack is one in which the adversary records and replays legitimate messages between  
3721 XACML actors. This attack may lead to denial of service, the use of out-of-date information or  
3722 impersonation.

3723 Prevention of replay attacks requires the use of message freshness safeguards.

3724 Note that encryption of the message does not mitigate a replay attack since the message is simply  
3725 replayed and does not have to be understood by the adversary.

#### 3726 9.1.3 Message insertion

3727 A message insertion attack is one in which the adversary inserts messages in the sequence of messages  
3728 between XACML actors.

3729 The solution to a message insertion attack is to use mutual authentication and message sequence  
3730 integrity safeguards between the actors. It should be noted that just using SSL mutual authentication is  
3731 not sufficient. This only proves that the other party is the one identified by the *subject* of the X.509

3732 certificate. In order to be effective, it is necessary to confirm that the certificate **subject** is authorized to  
3733 send the message.

#### 3734 9.1.4 Message deletion

3735 A message deletion attack is one in which the adversary deletes messages in the sequence of messages  
3736 between XACML actors. Message deletion may lead to denial of service. However, a properly designed  
3737 XACML system should not render an incorrect **authorization decision** as a result of a message deletion  
3738 attack.

3739 The solution to a message deletion attack is to use message sequence integrity safeguards between the  
3740 actors.

#### 3741 9.1.5 Message modification

3742 If an adversary can intercept a message and change its contents, then they may be able to alter an  
3743 **authorization decision**. A message integrity safeguard can prevent a successful message modification  
3744 attack.

#### 3745 9.1.6 NotApplicable results

3746 A result of "NotApplicable" means that the **PDP** could not locate a **policy** whose **target** matched the  
3747 information in the **decision request**. In general, it is highly recommended that a "Deny" **effect policy** be  
3748 used, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is returned instead.

3749 In some security models, however, such as those found in many web servers, an **authorization decision**  
3750 of "NotApplicable" is treated as equivalent to "Permit". There are particular security considerations that  
3751 must be taken into account for this to be safe. These are explained in the following paragraphs.

3752 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the **policy** to  
3753 match elements in the **decision request** be closely aligned with the data syntax used by the applications  
3754 that will be submitting the **decision request**. A failure to match will result in "NotApplicable" and be  
3755 treated as "Permit". So an unintended failure to match may allow unintended **access**.

3756 Commercial http responders allow a variety of syntaxes to be treated equivalently. The "%" can be used  
3757 to represent characters by hex value. The URL path "/./" provides multiple ways of specifying the same  
3758 value. Multiple character sets may be permitted and, in some cases, the same printed character can be  
3759 represented by different binary values. Unless the matching algorithm used by the **policy** is sophisticated  
3760 enough to catch these variations, unintended **access** may be permitted.

3761 It may be safe to treat "NotApplicable" as "Permit" only in a closed environment where all applications that  
3762 formulate a **decision request** can be guaranteed to use the exact syntax expected by the **policies**. In a  
3763 more open environment, where **decision requests** may be received from applications that use any legal  
3764 syntax, it is strongly recommended that "NotApplicable" NOT be treated as "Permit" unless matching  
3765 **rules** have been very carefully designed to match all possible applicable inputs, regardless of syntax or  
3766 type variations. Note, however, that according to Section 7.2, a **PEP** must deny **access** unless it  
3767 receives an explicit "Permit" **authorization decision**.

#### 3768 9.1.7 Negative rules

3769 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care, negative  
3770 **rules** can lead to policy violations, therefore some authorities recommend that they not be used.  
3771 However, negative **rules** can be extremely efficient in certain cases, so XACML has chosen to include  
3772 them. Nevertheless, it is recommended that they be used with care and avoided if possible.

3773 A common use for negative **rules** is to deny **access** to an individual or subgroup when their membership  
3774 in a larger group would otherwise permit them **access**. For example, we might want to write a **rule** that  
3775 allows all vice presidents to see the unpublished financial data, except for Joe, who is only a ceremonial  
3776 vice president and can be indiscreet in his communications. If we have complete control over the  
3777 administration of **subject attributes**, a superior approach would be to define "Vice President" and  
3778 "Ceremonial Vice President" as distinct groups and then define **rules** accordingly. However, in some

3779 environments this approach may not be feasible. (It is worth noting in passing that referring to individuals  
3780 in **rules** does not scale well. Generally, shared **attributes** are preferred.)

3781 If not used with care, negative **rules** can lead to policy violations in two common cases: when **attributes**  
3782 are suppressed and when the base group changes. An example of suppressed **attributes** would be if we  
3783 have a **policy** that **access** should be permitted, unless the **subject** is a credit risk. If it is possible that  
3784 the **attribute** of being a credit risk may be unknown to the **PDP** for some reason, then unauthorized  
3785 **access** may result. In some environments, the **subject** may be able to suppress the publication of  
3786 **attributes** by the application of privacy controls, or the server or repository that contains the information  
3787 may be unavailable for accidental or intentional reasons.

3788 An example of a changing base group would be if there is a **policy** that everyone in the engineering  
3789 department may change software source code, except for secretaries. Suppose now that the department  
3790 was to merge with another engineering department and the intent is to maintain the same **policy**.  
3791 However, the new department also includes individuals identified as administrative assistants, who ought  
3792 to be treated in the same way as secretaries. Unless the **policy** is altered, they will unintentionally be  
3793 permitted to change software source code. Problems of this type are easy to avoid when one individual  
3794 administers all **policies**, but when administration is distributed, as XACML allows, this type of situation  
3795 must be explicitly guarded against.

### 3796 9.1.8 Denial of service

3797 A denial of service attack is one in which the adversary overloads an XACML actor with excessive  
3798 computations or network traffic such that legitimate users cannot access the services provided by the  
3799 actor.

3800 The urn:oasis:names:tc:xacml:3.0:function:access-permitted function may lead to hard to predict behavior  
3801 in the **PDP**. It is possible that the function is invoked during the recursive invocations of the **PDP** such that  
3802 loops are formed. Such loops may in some cases lead to large numbers of requests to be generated  
3803 before the **PDP** can detect the loop and abort evaluation. Such loops could cause a denial of service at  
3804 the **PDP**, either because of a malicious **policy** or because of a mistake in a **policy**.

## 3805 9.2 Safeguards

### 3806 9.2.1 Authentication

3807 Authentication provides the means for one party in a transaction to determine the identity of the other  
3808 party in the transaction. Authentication may be in one direction, or it may be bilateral.

3809 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the  
3810 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an adversary  
3811 could provide false or invalid **authorization decisions**, leading to a policy violation.

3812 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust to  
3813 determine what, if any, sensitive data should be passed. One should keep in mind that even simple  
3814 "Permit" or "Deny" responses could be exploited if an adversary were allowed to make unlimited requests  
3815 to a **PDP**.

3816 Many different techniques may be used to provide authentication, such as co-located code, a private  
3817 network, a VPN, or digital signatures. Authentication may also be performed as part of the  
3818 communication protocol used to exchange the **contexts**. In this case, authentication may be performed  
3819 either at the message level or at the session level.

### 3820 9.2.2 Policy administration

3821 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects** may  
3822 use this information to determine how to gain unauthorized **access**.

3823 To prevent this threat, the repository used for the storage of **policies** may itself require **access control**.  
3824 In addition, the <Status> element should be used to return values of missing **attributes** only when  
3825 exposure of the identities of those **attributes** will not compromise security.

## 3826 9.2.3 Confidentiality

3827 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired  
3828 recipients and not by anyone else who encounters the message while it is in transit. There are two areas  
3829 in which confidentiality should be considered: one is confidentiality during transmission; the other is  
3830 confidentiality within a <Policy> element.

### 3831 9.2.3.1 Communication confidentiality

3832 In some environments it is deemed good practice to treat all data within an **access control** system as  
3833 confidential. In other environments, **policies** may be made freely available for distribution, inspection,  
3834 and audit. The idea behind keeping **policy** information secret is to make it more difficult for an adversary  
3835 to know what steps might be sufficient to obtain unauthorized **access**. Regardless of the approach  
3836 chosen, the security of the **access control** system should not depend on the secrecy of the **policy**.

3837 Any security considerations related to transmitting or exchanging XACML <Policy> elements are  
3838 outside the scope of the XACML standard. While it is important to ensure that the integrity and  
3839 confidentiality of <Policy> elements is maintained when they are exchanged between two parties, it is  
3840 left to the implementers to determine the appropriate mechanisms for their environment.

3841 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL. Using a  
3842 point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points is  
3843 compromised.

### 3844 9.2.3.2 Statement level confidentiality

3845 In some cases, an implementation may want to encrypt only parts of an XACML <Policy> element.

3846 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used to  
3847 encrypt all or parts of an XML document. This specification is recommended for use with XACML.

3848 It should go without saying that if a repository is used to facilitate the communication of cleartext (i.e.,  
3849 unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to store this  
3850 sensitive data.

## 3851 9.2.4 Policy integrity

3852 The XACML **policy** used by the **PDP** to evaluate the request **context** is the heart of the system.  
3853 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of the  
3854 **policy**. One is to ensure that <Policy> elements have not been altered since they were originally  
3855 created by the **PAP**. The other is to ensure that <Policy> elements have not been inserted or deleted  
3856 from the set of **policies**.

3857 In many cases, both aspects can be achieved by ensuring the integrity of the actors and implementing  
3858 session-level mechanisms to secure the communication between actors. The selection of the appropriate  
3859 mechanisms is left to the implementers. However, when **policy** is distributed between organizations to  
3860 be acted on at a later time, or when the **policy** travels with the protected **resource**, it would be useful to  
3861 sign the **policy**. In these cases, the XML Signature Syntax and Processing standard from W3C is  
3862 recommended to be used with XACML.

3863 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures should  
3864 not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not request a **policy**  
3865 based on who signed it or whether or not it has been signed (as such a basis for selection would, itself,  
3866 be a matter of policy). However, the **PDP** must verify that the key used to sign the **policy** is one  
3867 controlled by the purported **issuer** of the **policy**. The means to do this are dependent on the specific  
3868 signature technology chosen and are outside the scope of this document.

## 3869 9.2.5 Policy identifiers

3870 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure that  
3871 these are unique. Confusion between identifiers could lead to misidentification of the **applicable policy**.

3872 This specification is silent on whether a **PAP** must generate a new identifier when a **policy** is modified or  
3873 may use the same identifier in the modified **policy**. This is a matter of administrative practice. However,  
3874 care must be taken in either case. If the identifier is reused, there is a danger that other **policies** or  
3875 **policy sets** that reference it may be adversely affected. Conversely, if a new identifier is used, these  
3876 other **policies** may continue to use the prior **policy**, unless it is deleted. In either case the results may  
3877 not be what the **policy** administrator intends.

3878 If a **PDP** is provided with **policies** from distinct sources which might not be fully trusted, as in the use of  
3879 the administration profile [**XACMLAdmin**], there is a concern that someone could intentionally publish a  
3880 **policy** with an id which collides with another **policy**. This could cause **policy** references that point to the  
3881 wrong **policy**, and may cause other unintended consequences in an implementation which is predicated  
3882 upon having unique **policy** identifiers.

3883 If this issue is a concern it is RECOMMENDED that distinct **policy** issuers or sources are assigned  
3884 distinct namespaces for **policy** identifiers. One method is to make sure that the **policy** identifier begins  
3885 with a string which has been assigned to the particular **policy** issuer or source. The remainder of the  
3886 **policy** identifier is an issuer-specific unique part. For instance, Alice from Example Inc. could be assigned  
3887 the **policy** identifiers which begin with `http://example.com/xacml/policyId/alice/`. The **PDP** or another  
3888 trusted component can then verify that the authenticated source of the **policy** is Alice at Example Inc, or  
3889 otherwise reject the **policy**. Anyone else will be unable to publish **policies** with identifiers which collide  
3890 with the **policies** of Alice.

## 3891 9.2.6 Trust model

3892 Discussions of authentication, integrity and confidentiality safeguards necessarily assume an underlying  
3893 trust model: how can one actor come to believe that a given key is uniquely associated with a specific,  
3894 identified actor so that the key can be used to encrypt data for that actor or verify signatures (or other  
3895 integrity structures) from that actor? Many different types of trust models exist, including strict  
3896 hierarchies, distributed authorities, the Web, the bridge, and so on.

3897 It is worth considering the relationships between the various actors of the **access control** system in terms  
3898 of the interdependencies that do and do not exist.

- 3899 • None of the entities of the authorization system are dependent on the **PEP**. They may collect data  
3900 from it, (for example authentication data) but are responsible for verifying it themselves.
- 3901 • The correct operation of the system depends on the ability of the **PEP** to actually enforce **policy**  
3902 **decisions**.
- 3903 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the **PDP** is  
3904 supplied with the correct inputs. Other than that, the **PDP** does not depend on the **PEP**.
- 3905 • The **PDP** depends on the **PAP** to supply appropriate **policies**. The **PAP** is not dependent on other  
3906 components.

## 3907 9.2.7 Privacy

3908 It is important to be aware that any transactions that occur with respect to **access control** may reveal  
3909 private information about the actors. For example, if an XACML **policy** states that certain data may only  
3910 be read by **subjects** with "Gold Card Member" status, then any transaction in which a **subject** is  
3911 permitted **access** to that data leaks information to an adversary about the **subject's** status. Privacy  
3912 considerations may therefore lead to encryption and/or to **access control** requirements surrounding the  
3913 enforcement of XACML **policy** instances themselves: confidentiality-protected channels for the  
3914 request/response protocol messages, protection of **subject attributes** in storage and in transit, and so  
3915 on.

3916 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope of  
3917 XACML. The **decision** regarding whether, how, and when to deploy such mechanisms is left to the  
3918 implementers associated with the environment.

3919 **9.3 Unicode security issues**

3920 There are many security considerations related to use of Unicode. An XACML implementation SHOULD  
3921 follow the advice given in the relevant version of **[UTR36]**.

3922 **9.4 Identifier equality**

3923 Section 7.20 defines the identifier equality operation for XACML. This definition of equality does not do  
3924 any kind of canonicalization or escaping of characters. The identifiers defined in the XACML specification  
3925 have been selected to not include any ambiguity regarding these aspects. It is RECOMMENDED that  
3926 identifiers defined by extensions also do not introduce any identifiers which might be mistaken for being  
3927 subject to processing, like for instance URL character encoding using “%”.

---

3928 **10 Conformance**

3929 **10.1 Introduction**

3930 The XACML specification addresses the following aspect of conformance:

3931 The XACML specification defines a number of functions, etc. that have somewhat special applications,  
3932 therefore they are not required to be implemented in an implementation that claims to conform with the  
3933 OASIS standard.

3934 **10.2 Conformance tables**

3935 This section lists those portions of the specification that **MUST** be included in an implementation of a **PDP**  
3936 that claims to conform to XACML v3.0. A set of test cases has been created to assist in this process.  
3937 These test cases can be located from the OASIS XACML TC Web page. The site hosting the test cases  
3938 contains a full description of the test cases and how to execute them.

3939 Note: "M" means mandatory-to-implement. "O" means optional.

3940 The implementation **MUST** follow sections 5, 6, 7, Appendix A, Appendix B and Appendix C where they  
3941 apply to implemented items in the following tables.

3942 **10.2.1 Schema elements**

3943 The implementation **MUST** support those schema elements that are marked "M".

Element name	M/O
xacml:Advice	M
xacml:AdviceExpression	M
xacml:AdviceExpressions	M
xacml:AllOf	M
xacml:AnyOf	M
xacml:Apply	M
xacml:AssociatedAdvice	M
xacml:Attribute	M
xacml:AttributeAssignment	M
xacml:AttributeAssignmentExpression	M
xacml:AttributeDesignator	M
xacml:Attributes	M
xacml:AttributeSelector	O
xacml:AttributesReference	O
xacml:AttributeValue	M
xacml:CombinerParameter	O
xacml:CombinerParameters	O
xacml:Condition	M
xacml:Content	O
xacml:Decision	M
xacml:Description	M
xacml:Expression	M
xacml:Function	M
xacml:Match	M
xacml:MissingAttributeDetail	M
xacml:MultiRequests	O
xacml:Obligation	M
xacml:ObligationExpression	M
xacml:ObligationExpressions	M
xacml:Obligations	M

xacml:Policy	M
xacml:PolicyCombinerParameters	O
xacml:PolicyDefaults	O
xacml:PolicyIdentifierList	O
xacml:PolicyIdReference	M
xacml:PolicyIssuer	O
xacml:PolicySet	M
xacml:PolicySetDefaults	O
xacml:PolicySetIdReference	M
xacml:Request	M
xacml:RequestDefaults	O
xacml:RequestReference	O
xacml:Response	M
xacml:Result	M
xacml:Rule	M
xacml:RuleCombinerParameters	O
xacml:Status	M
xacml:StatusCode	M
xacml:StatusDetail	O
xacml:StatusMessage	O
xacml:Target	M
xacml:VariableDefinition	M
xacml:VariableReference	M
xacml:XPathVersion	O

3944 **10.2.2 Identifier Prefixes**

3945 The following identifier prefixes are reserved by XACML.

Identifier
urn:oasis:names:tc:xacml:3.0
urn:oasis:names:tc:xacml:2.0
urn:oasis:names:tc:xacml:2.0:conformance-test
urn:oasis:names:tc:xacml:2.0:context
urn:oasis:names:tc:xacml:2.0:example
urn:oasis:names:tc:xacml:1.0:function
urn:oasis:names:tc:xacml:2.0:function
urn:oasis:names:tc:xacml:2.0:policy
urn:oasis:names:tc:xacml:1.0:subject
urn:oasis:names:tc:xacml:1.0:resource
urn:oasis:names:tc:xacml:1.0:action
urn:oasis:names:tc:xacml:1.0:environment
urn:oasis:names:tc:xacml:1.0:status

3946 **10.2.3 Algorithms**

3947 The implementation MUST include the **rule-** and **policy-combining algorithms** associated with the  
3948 following identifiers that are marked "M".

Algorithm	M/O
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides	M
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides	M
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides	M
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides	M
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-	M

applicable	
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides	M
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-overrides	M
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-overrides	M
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-overrides	M
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit	M
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit	M
urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny	M
urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny	M

## 3949 10.2.4 Status Codes

3950 Implementation support for the <StatusCode> element is optional, but if the element is supported, then  
3951 the following status codes must be supported and must be used in the way XACML has specified.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:status:missing-attribute	M
urn:oasis:names:tc:xacml:1.0:status:ok	M
urn:oasis:names:tc:xacml:1.0:status:processing-error	M
urn:oasis:names:tc:xacml:1.0:status:syntax-error	M

## 3952 10.2.5 Attributes

3953 The implementation MUST support the **attributes** associated with the following identifiers as specified by  
3954 XACML. If values for these **attributes** are not present in the **decision request**, then their values MUST  
3955 be supplied by the **context handler**. So, unlike most other **attributes**, their semantics are not  
3956 transparent to the **PDP**.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:environment:current-time	M
urn:oasis:names:tc:xacml:1.0:environment:current-date	M
urn:oasis:names:tc:xacml:1.0:environment:current-dateTime	M

## 3957 10.2.6 Identifiers

3958 The implementation MUST use the **attributes** associated with the following identifiers in the way XACML  
3959 has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that uses XACML,  
3960 since the semantics of the **attributes** are transparent to the **PDP**.

Identifier	M/O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name	O
urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address	O
urn:oasis:names:tc:xacml:1.0:subject:authentication-method	O
urn:oasis:names:tc:xacml:1.0:subject:authentication-time	O
urn:oasis:names:tc:xacml:1.0:subject:key-info	O
urn:oasis:names:tc:xacml:1.0:subject:request-time	O
urn:oasis:names:tc:xacml:1.0:subject:session-start-time	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id	O
urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier	O
urn:oasis:names:tc:xacml:1.0:subject-category:access-subject	M
urn:oasis:names:tc:xacml:1.0:subject-category:codebase	O

urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject	O
urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine	O
urn:oasis:names:tc:xacml:1.0:resource:resource-location	O
urn:oasis:names:tc:xacml:1.0:resource:resource-id	M
urn:oasis:names:tc:xacml:1.0:resource:simple-file-name	O
urn:oasis:names:tc:xacml:1.0:action:action-id	O
urn:oasis:names:tc:xacml:1.0:action:implied-action	O

3961 **10.2.7 Data-types**

3962 The implementation MUST support the data-types associated with the following identifiers marked "M".

Data-type	M/O
http://www.w3.org/2001/XMLSchema#string	M
http://www.w3.org/2001/XMLSchema#boolean	M
http://www.w3.org/2001/XMLSchema#integer	M
http://www.w3.org/2001/XMLSchema#double	M
http://www.w3.org/2001/XMLSchema#time	M
http://www.w3.org/2001/XMLSchema#date	M
http://www.w3.org/2001/XMLSchema#dateTime	M
http://www.w3.org/2001/XMLSchema#dayTimeDuration	M
http://www.w3.org/2001/XMLSchema#yearMonthDuration	M
http://www.w3.org/2001/XMLSchema#anyURI	M
http://www.w3.org/2001/XMLSchema#hexBinary	M
http://www.w3.org/2001/XMLSchema#base64Binary	M
urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name	M
urn:oasis:names:tc:xacml:1.0:data-type:x500Name	M
urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression	O
urn:oasis:names:tc:xacml:2.0:data-type:ipAddress	M
urn:oasis:names:tc:xacml:2.0:data-type:dnsName	M

3963 **10.2.8 Functions**

3964 The implementation MUST properly process those functions associated with the identifiers marked with  
3965 an "M".

Function	M/O
urn:oasis:names:tc:xacml:1.0:function:string-equal	M
urn:oasis:names:tc:xacml:1.0:function:boolean-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-equal	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal	M
urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-add	M
urn:oasis:names:tc:xacml:1.0:function:double-add	M
urn:oasis:names:tc:xacml:1.0:function:integer-subtract	M
urn:oasis:names:tc:xacml:1.0:function:double-subtract	M
urn:oasis:names:tc:xacml:1.0:function:integer-multiply	M

urn:oasis:names:tc:xacml:1.0:function:double-multiply	M
urn:oasis:names:tc:xacml:1.0:function:integer-divide	M
urn:oasis:names:tc:xacml:1.0:function:double-divide	M
urn:oasis:names:tc:xacml:1.0:function:integer-mod	M
urn:oasis:names:tc:xacml:1.0:function:integer-abs	M
urn:oasis:names:tc:xacml:1.0:function:double-abs	M
urn:oasis:names:tc:xacml:1.0:function:round	M
urn:oasis:names:tc:xacml:1.0:function:floor	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-space	M
urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case	M
urn:oasis:names:tc:xacml:1.0:function:double-to-integer	M
urn:oasis:names:tc:xacml:1.0:function:integer-to-double	M
urn:oasis:names:tc:xacml:1.0:function:or	M
urn:oasis:names:tc:xacml:1.0:function:and	M
urn:oasis:names:tc:xacml:1.0:function:n-of	M
urn:oasis:names:tc:xacml:1.0:function:not	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than	M
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than	M
urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal	M
urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration	M
urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration	M
urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than	M
urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than	M
urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal	M
urn:oasis:names:tc:xacml:2.0:function:time-in-range	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than	M
urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:date-less-than	M
urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal	M
urn:oasis:names:tc:xacml:1.0:function:string-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:string-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:string-is-in	M
urn:oasis:names:tc:xacml:1.0:function:string-bag	M
urn:oasis:names:tc:xacml:1.0:function:boolean-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:boolean-is-in	M
urn:oasis:names:tc:xacml:1.0:function:boolean-bag	M
urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag-size	M

urn:oasis:names:tc:xacml:1.0:function:integer-is-in	M
urn:oasis:names:tc:xacml:1.0:function:integer-bag	M
urn:oasis:names:tc:xacml:1.0:function:double-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:double-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:double-is-in	M
urn:oasis:names:tc:xacml:1.0:function:double-bag	M
urn:oasis:names:tc:xacml:1.0:function:time-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:time-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:time-is-in	M
urn:oasis:names:tc:xacml:1.0:function:time-bag	M
urn:oasis:names:tc:xacml:1.0:function:date-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:date-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:date-is-in	M
urn:oasis:names:tc:xacml:1.0:function:date-bag	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-bag	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-is-in	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-bag	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-bag	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-is-in	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-bag	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-one-and-only	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-bag-size	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-is-in	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-bag	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-one-and-only	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-bag-size	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-is-in	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-is-in	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-bag	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-is-in	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-bag	M
urn:oasis:names:tc:xacml:2.0:function:ipAddress-one-and-only	M
urn:oasis:names:tc:xacml:2.0:function:ipAddress-bag-size	M
urn:oasis:names:tc:xacml:2.0:function:ipAddress-bag	M
urn:oasis:names:tc:xacml:2.0:function:dnsName-one-and-only	M
urn:oasis:names:tc:xacml:2.0:function:dnsName-bag-size	M
urn:oasis:names:tc:xacml:2.0:function:dnsName-bag	M
urn:oasis:names:tc:xacml:2.0:function:string-concatenate	M
urn:oasis:names:tc:xacml:3.0:function:boolean-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-boolean	M
urn:oasis:names:tc:xacml:3.0:function:integer-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-integer	M
urn:oasis:names:tc:xacml:3.0:function:double-from-string	M

urn:oasis:names:tc:xacml:3.0:function:string-from-double	M
urn:oasis:names:tc:xacml:3.0:function:time-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-time	M
urn:oasis:names:tc:xacml:3.0:function:date-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-date	M
urn:oasis:names:tc:xacml:3.0:function:dateTime-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-dateTime	M
urn:oasis:names:tc:xacml:3.0:function:anyURI-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-dayTimeDuration	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-yearMonthDuration	M
urn:oasis:names:tc:xacml:3.0:function:x500Name-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-x500Name	M
urn:oasis:names:tc:xacml:3.0:function:rfc822Name-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-rfc822Name	M
urn:oasis:names:tc:xacml:3.0:function:ipAddress-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-ipAddress	M
urn:oasis:names:tc:xacml:3.0:function:dnsName-from-string	M
urn:oasis:names:tc:xacml:3.0:function:string-from-dnsName	M
urn:oasis:names:tc:xacml:3.0:function:string-starts-with	M
urn:oasis:names:tc:xacml:3.0:function:anyURI-starts-with	M
urn:oasis:names:tc:xacml:3.0:function:string-ends-with	M
urn:oasis:names:tc:xacml:3.0:function:anyURI-ends-with	M
urn:oasis:names:tc:xacml:3.0:function:string-contains	M
urn:oasis:names:tc:xacml:3.0:function:anyURI-contains	M
urn:oasis:names:tc:xacml:3.0:function:string-substring	M
urn:oasis:names:tc:xacml:3.0:function:anyURI-substring	M
urn:oasis:names:tc:xacml:3.0:function:any-of	M
urn:oasis:names:tc:xacml:3.0:function:all-of	M
urn:oasis:names:tc:xacml:3.0:function:any-of-any	M
urn:oasis:names:tc:xacml:1.0:function:all-of-any	M
urn:oasis:names:tc:xacml:1.0:function:any-of-all	M
urn:oasis:names:tc:xacml:1.0:function:all-of-all	M
urn:oasis:names:tc:xacml:3.0:function:map	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-match	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match	M
urn:oasis:names:tc:xacml:1.0:function:string-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:dnsName-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:rfc822Name-regexp-match	M
urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match	M
urn:oasis:names:tc:xacml:3.0:function:xpath-node-count	O
urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal	O
urn:oasis:names:tc:xacml:3.0:function:xpath-node-match	O
urn:oasis:names:tc:xacml:1.0:function:string-intersection	M
urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:string-union	M
urn:oasis:names:tc:xacml:1.0:function:string-subset	M
urn:oasis:names:tc:xacml:1.0:function:string-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:boolean-intersection	M
urn:oasis:names:tc:xacml:1.0:function:boolean-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:boolean-union	M
urn:oasis:names:tc:xacml:1.0:function:boolean-subset	M
urn:oasis:names:tc:xacml:1.0:function:boolean-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:integer-intersection	M

urn:oasis:names:tc:xacml:1.0:function:integer-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:integer-union	M
urn:oasis:names:tc:xacml:1.0:function:integer-subset	M
urn:oasis:names:tc:xacml:1.0:function:integer-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:double-intersection	M
urn:oasis:names:tc:xacml:1.0:function:double-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:double-union	M
urn:oasis:names:tc:xacml:1.0:function:double-subset	M
urn:oasis:names:tc:xacml:1.0:function:double-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:time-intersection	M
urn:oasis:names:tc:xacml:1.0:function:time-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:time-union	M
urn:oasis:names:tc:xacml:1.0:function:time-subset	M
urn:oasis:names:tc:xacml:1.0:function:time-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:date-intersection	M
urn:oasis:names:tc:xacml:1.0:function:date-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:date-union	M
urn:oasis:names:tc:xacml:1.0:function:date-subset	M
urn:oasis:names:tc:xacml:1.0:function:date-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-intersection	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-union	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subset	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-intersection	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-union	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-subset	M
urn:oasis:names:tc:xacml:1.0:function:anyURI-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-intersection	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-union	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-subset	M
urn:oasis:names:tc:xacml:1.0:function:hexBinary-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-intersection	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-union	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-subset	M
urn:oasis:names:tc:xacml:1.0:function:base64Binary-set-equals	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-intersection	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-union	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-subset	M
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-set-equals	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-intersection	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-union	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-subset	M
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-intersection	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-union	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-subset	M
urn:oasis:names:tc:xacml:1.0:function:x500Name-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-intersection	M

urn:oasis:names:tc:xacml:1.0:function:rfc822Name-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-union	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-subset	M
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-set-equals	M
urn:oasis:names:tc:xacml:3.0:function:access-permitted	O

3966 **10.2.9 Identifiers planned for future deprecation**

3967 These identifiers are associated with previous versions of XACML and newer alternatives exist in XACML  
3968 3.0. They are planned to be deprecated at some unspecified point in the future. It is RECOMMENDED  
3969 that these identifiers not be used in new policies and requests.

3970 The implementation MUST properly process those features associated with the identifiers marked with an  
3971 "M".

Function	M/O
urn:oasis:names:tc:xacml:1.0:function:xpath-node-count	O
urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal	O
urn:oasis:names:tc:xacml:1.0:function:xpath-node-match	O
urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate	M
http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration	M
http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration	M
urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration	M
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides	M
urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides	M
urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides	M
urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides	M
urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides	M
urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides	M
urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-intersection	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-union	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-subset	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-intersection	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-at-least-one-member-of	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-union	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-subset	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-set-equals	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag-size	M

urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-one-and-only	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag-size	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-is-in	M
urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-bag	M
urn:oasis:names:tc:xacml:1.0:function:any-of	M
urn:oasis:names:tc:xacml:1.0:function:all-of	M
urn:oasis:names:tc:xacml:1.0:function:any-of-any	M
urn:oasis:names:tc:xacml:1.0:function:map	M

3972

3973

## Appendix A. Data-types and functions (normative)

3974

### A.1 Introduction

3975 This section specifies the data-types and functions used in XACML to create *predicates* for *conditions*  
3976 and *target* matches.

3977 This specification combines the various standards set forth by IEEE and ANSI for string representation of  
3978 numeric values, as well as the evaluation of arithmetic functions. It describes the primitive data-types and  
3979 *bags*. The standard functions are named and their operational semantics are described.

### A.2 Data-types

3981 Although XML instances represent all data-types as strings, an XACML *PDP* must operate on types of  
3982 data that, while they have string representations, are not just strings. Types such as Boolean, integer,  
3983 and double MUST be converted from their XML string representations to values that can be compared  
3984 with values in their domain of discourse, such as numbers. The following primitive data-types are  
3985 specified for use with XACML and have explicit data representations:

- 3986 • <http://www.w3.org/2001/XMLSchema#string>
- 3987 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3988 • <http://www.w3.org/2001/XMLSchema#integer>
- 3989 • <http://www.w3.org/2001/XMLSchema#double>
- 3990 • <http://www.w3.org/2001/XMLSchema#time>
- 3991 • <http://www.w3.org/2001/XMLSchema#date>
- 3992 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3993 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3994 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3995 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3996 • <http://www.w3.org/2001/XMLSchema#dayTimeDuration>
- 3997 • <http://www.w3.org/2001/XMLSchema#yearMonthDuration>
- 3998 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3999 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>
- 4000 • <urn:oasis:names:tc:xacml:2.0:data-type:ipAddress>
- 4001 • <urn:oasis:names:tc:xacml:2.0:data-type:dnsName>
- 4002 • <urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression>

4003 For the sake of improved interoperability, it is RECOMMENDED that all time references be in UTC time.

4004 An XACML *PDP* SHALL be capable of converting string representations into various primitive data-types.  
4005 For doubles, XACML SHALL use the conversions described in [IEEE754].

4006 XACML defines four data-types representing identifiers for *subjects* or *resources*; these are:

- 4007 “urn:oasis:names:tc:xacml:1.0:data-type:x500Name”,
- 4008 “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”
- 4009 “urn:oasis:names:tc:xacml:2.0:data-type:ipAddress” and
- 4010 “urn:oasis:names:tc:xacml:2.0:data-type:dnsName”

4011 These types appear in several standard applications, such as TLS/SSL and electronic mail.

#### 4012 X.500 directory name

4013 The "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" primitive type represents an ITU-T Rec.  
4014 X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253  
4015 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished  
4016 Names".

#### 4017 **RFC 822 name**

4018 The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents an electronic  
4019 mail address. The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2,  
4020 Command Argument Syntax, under the term "Mailbox".

#### 4021 **IP address**

4022 The "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" primitive type represents an IPv4 or IPv6  
4023 network address, with optional mask and optional port or port range. The syntax SHALL be:

4024 `ipAddress = address [ "/" mask ] [ ":" [ portrange ] ]`

4025 For an IPv4 address, the address and mask are formatted in accordance with the syntax for a  
4026 "host" in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2.

4027 For an IPv6 address, the address and mask are formatted in accordance with the syntax for an  
4028 "ipv6reference" in IETF RFC 2732 "Format for Literal IPv6 Addresses in URL's". (Note that an  
4029 IPv6 address or mask, in this syntax, is enclosed in literal "[" "]" brackets.)

#### 4030 **DNS name**

4031 The "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" primitive type represents a Domain  
4032 Name Service (DNS) host name, with optional port or port range. The syntax SHALL be:

4033 `dnsName = hostname [ ":" portrange ]`

4034 The hostname is formatted in accordance with IETF RFC 2396 "Uniform Resource Identifiers  
4035 (URI): Generic Syntax", section 3.2, except that a wildcard "\*" may be used in the left-most  
4036 component of the hostname to indicate "any subdomain" under the domain specified to its right.

4037 For both the "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and  
4038 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" data-types, the port or port range syntax  
4039 SHALL be

4040 `portrange = portnumber | "-"portnumber | portnumber "-"[portnumber]`

4041 where "portnumber" is a decimal port number. If the port number is of the form "-x", where "x" is  
4042 a port number, then the range is all ports numbered "x" and below. If the port number is of the  
4043 form "x-", then the range is all ports numbered "x" and above. [This syntax is taken from the Java  
4044 SocketPermission.]

#### 4045 **XPath expression**

4046 The "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" primitive type represents an  
4047 XPath expression over the XML in a <Content> element. The syntax is defined by the XPath  
4048 W3C recommendation. The content of this data type also includes the context in which  
4049 namespaces prefixes in the expression are resolved, which distinguishes it from a plain string and  
4050 the XACML **attribute** category of the <Content> element to which it applies. When the value is  
4051 encoded in an <AttributeValue> element, the namespace context is given by the  
4052 <AttributeValue> element and an XML attribute called XPathCategory gives the category of  
4053 the <Content> element where the expression applies.

4054 The XPath expression MUST be evaluated in a context which is equivalent of a stand alone XML  
4055 document with the only child of the <Content> element as the document element. Namespace  
4056 declarations which are not "visibly utilized", as defined by [exc-c14n], MAY not be present and  
4057 MUST NOT be utilized by the XPath expression. The context node of the XPath expression is the  
4058 document node of this stand alone document.

## 4059 A.3 Functions

4060 XACML specifies the following functions. Unless otherwise specified, if an argument of one of these  
4061 functions were to evaluate to "Indeterminate", then the function SHALL be set to "Indeterminate".

4062 Note that in each case an implementation is conformant as long as it produces the same result as is  
4063 specified here, regardless of how and in what order the implementation behaves internally.

### 4064 A.3.1 Equality predicates

4065 The following functions are the equality functions for the various primitive types. Each function for a  
4066 particular data-type follows a specified standard convention for that data-type.

- 4067 • urn:oasis:names:tc:xacml:1.0:function:string-equal

4068 This function SHALL take two arguments of data-type  
4069 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4070 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if  
4071 the value of both of its arguments are of equal length and each string is determined to be equal.  
4072 Otherwise, it SHALL return "False". The comparison SHALL use Unicode codepoint collation, as  
4073 defined for the identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.

- 4074 • urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case

4075 This function SHALL take two arguments of data-type  
4076 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4077 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be "True" if and only if the  
4078 two strings are equal as defined by urn:oasis:names:tc:xacml:1.0:function:string-equal after they  
4079 have both been converted to lower case with urn:oasis:names:tc:xacml:1.0:function:string-  
4080 normalize-to-lower-case.

- 4081 • urn:oasis:names:tc:xacml:1.0:function:boolean-equal

4082 This function SHALL take two arguments of data-type  
4083 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return an  
4084 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if  
4085 the arguments are equal. Otherwise, it SHALL return "False".

- 4086 • urn:oasis:names:tc:xacml:1.0:function:integer-equal

4087 This function SHALL take two arguments of data-type  
4088 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an  
4089 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if  
4090 the two arguments represent the same number.

- 4091 • urn:oasis:names:tc:xacml:1.0:function:double-equal

4092 This function SHALL take two arguments of data-type  
4093 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an  
4094 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on doubles  
4095 according to IEEE 754 **[IEEE754]**.

- 4096 • urn:oasis:names:tc:xacml:1.0:function:date-equal

4097 This function SHALL take two arguments of data-type  
4098 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4099 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to  
4100 the "op:date-equal" function **[XF]** Section 10.4.9.

- 4101 • urn:oasis:names:tc:xacml:1.0:function:time-equal

4102 This function SHALL take two arguments of data-type  
4103 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4104 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to  
4105 the "op:time-equal" function **[XF]** Section 10.4.12.

- 4106 • urn:oasis:names:tc:xacml:1.0:function:dateTime-equal  
 4107 This function SHALL take two arguments of data-type  
 4108 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
 4109 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to  
 4110 the "op:dateTime-equal" function [XF] Section 10.4.6.
- 4111 • urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal  
 4112 This function SHALL take two arguments of data-type  
 4113 "http://www.w3.org/2001/XMLSchema#dayTimeDuration" and SHALL return an  
 4114 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
 4115 according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical  
 4116 representation of each argument MUST be converted to a value expressed in fractional seconds  
 4117 [XF] Section 10.3.2.
- 4118 • urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal  
 4119 This function SHALL take two arguments of data-type  
 4120 "http://www.w3.org/2001/XMLSchema#yearMonthDuration" and SHALL return an  
 4121 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
 4122 according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical  
 4123 representation of each argument MUST be converted to a value expressed in fractional seconds  
 4124 [XF] Section 10.3.2.
- 4125 • urn:oasis:names:tc:xacml:1.0:function:anyURI-equal  
 4126 This function SHALL take two arguments of data-type  
 4127 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an  
 4128 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL convert the arguments to  
 4129 strings with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI and return "True" if and  
 4130 only if the values of the two arguments are equal on a codepoint-by-codepoint basis.
- 4131 • urn:oasis:names:tc:xacml:1.0:function:x500Name-equal  
 4132 This function SHALL take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name"  
 4133 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if  
 4134 and only if each Relative Distinguished Name (RDN) in the two arguments matches. Otherwise,  
 4135 it SHALL return "False". Two RDNs shall be said to match if and only if the result of the following  
 4136 operations is "True" .
- 4137 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory Access  
 4138 Protocol (v3): UTF-8 String Representation of Distinguished Names".
  - 4139 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute  
 4140 ValuePairs in that RDN in ascending order when compared as octet strings (described in  
 4141 ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
  - 4142 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key  
 4143 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section 4.1.2.4  
 4144 "Issuer".
- 4145 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal  
 4146 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-  
 4147 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It  
 4148 SHALL return "True" if and only if the two arguments are equal. Otherwise, it SHALL return  
 4149 "False". An RFC822 name consists of a local-part followed by "@" followed by a domain-part.  
 4150 The local-part is case-sensitive, while the domain-part (which is usually a DNS host name) is not  
 4151 case-sensitive. Perform the following operations:
- 4152 1. Normalize the domain-part of each argument to lower case
  - 4153 2. Compare the expressions by applying the function  
 4154 "urn:oasis:names:tc:xacml:1.0:function:string-equal" to the normalized arguments.

- 4155 • urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal
  - 4156 This function SHALL take two arguments of data-type
  - 4157 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an
  - 4158 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet sequences
  - 4159 represented by the value of both arguments have equal length and are equal in a conjunctive,
  - 4160 point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function.
  - 4161 Otherwise, it SHALL return "False". The conversion from the string representation to an octet
  - 4162 sequence SHALL be as specified in [XS] Section 3.2.15.
- 4163 • urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal
  - 4164 This function SHALL take two arguments of data-type
  - 4165 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an
  - 4166 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet sequences
  - 4167 represented by the value of both arguments have equal length and are equal in a conjunctive,
  - 4168 point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function.
  - 4169 Otherwise, it SHALL return "False". The conversion from the string representation to an octet
  - 4170 sequence SHALL be as specified in [XS] Section 3.2.16.

### 4171 A.3.2 Arithmetic functions

4172 All of the following functions SHALL take two arguments of the specified data-type, integer, or double,  
 4173 and SHALL return an element of integer or double data-type, respectively. However, the "add" and  
 4174 "multiply" functions MAY take more than two arguments. Each function evaluation operating on doubles  
 4175 SHALL proceed as specified by their logical counterparts in IEEE 754 [IEEE754]. For all of these  
 4176 functions, if any argument is "Indeterminate", then the function SHALL evaluate to "Indeterminate". In the  
 4177 case of the divide functions, if the divisor is zero, then the function SHALL evaluate to "Indeterminate".

- 4178 • urn:oasis:names:tc:xacml:1.0:function:integer-add
  - 4179 This function MUST accept two or more arguments.
- 4180 • urn:oasis:names:tc:xacml:1.0:function:double-add
  - 4181 This function MUST accept two or more arguments.
- 4182 • urn:oasis:names:tc:xacml:1.0:function:integer-subtract
  - 4183 The result is the second argument subtracted from the first argument.
- 4184 • urn:oasis:names:tc:xacml:1.0:function:double-subtract
  - 4185 The result is the second argument subtracted from the first argument.
- 4186 • urn:oasis:names:tc:xacml:1.0:function:integer-multiply
  - 4187 This function MUST accept two or more arguments.
- 4188 • urn:oasis:names:tc:xacml:1.0:function:double-multiply
  - 4189 This function MUST accept two or more arguments.
- 4190 • urn:oasis:names:tc:xacml:1.0:function:integer-divide
  - 4191 The result is the first argument divided by the second argument.
- 4192 • urn:oasis:names:tc:xacml:1.0:function:double-divide
  - 4193 The result is the first argument divided by the second argument.
- 4194 • urn:oasis:names:tc:xacml:1.0:function:integer-mod
  - 4195 The result is remainder of the first argument divided by the second argument.

4196 The following functions SHALL take a single argument of the specified data-type. The round and floor  
 4197 functions SHALL take a single argument of data-type "http://www.w3.org/2001/XMLSchema#double" and  
 4198 return a value of the data-type "http://www.w3.org/2001/XMLSchema#double".

- 4199 • urn:oasis:names:tc:xacml:1.0:function:integer-abs

- 4200 • urn:oasis:names:tc:xacml:1.0:function:double-abs
- 4201 • urn:oasis:names:tc:xacml:1.0:function:round
- 4202 • urn:oasis:names:tc:xacml:1.0:function:floor

### 4203 **A.3.3 String conversion functions**

4204 The following functions convert between values of the data-type  
4205 “http://www.w3.org/2001/XMLSchema#string” primitive types.

- 4206 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-space
  - 4207 This function SHALL take one argument of data-type
  - 4208 “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by stripping off all
  - 4209 leading and trailing white space characters. The whitespace characters are defined in the
  - 4210 metasympol S (Production 3) of [XML].
- 4211 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case
  - 4212 This function SHALL take one argument of data-type
  - 4213 “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by converting each
  - 4214 upper case character to its lower case equivalent. Case mapping shall be done as specified for
  - 4215 the fn:lower-case function in [XF] with no tailoring for particular languages or environments.

### 4216 **A.3.4 Numeric data-type conversion functions**

4217 The following functions convert between the data-type “http://www.w3.org/2001/XMLSchema#integer”  
4218 and” http://www.w3.org/2001/XMLSchema#double” primitive types.

- 4219 • urn:oasis:names:tc:xacml:1.0:function:double-to-integer
  - 4220 This function SHALL take one argument of data-type
  - 4221 “http://www.w3.org/2001/XMLSchema#double” and SHALL truncate its numeric value to a whole
  - 4222 number and return an element of data-type “http://www.w3.org/2001/XMLSchema#integer”.
- 4223 • urn:oasis:names:tc:xacml:1.0:function:integer-to-double
  - 4224 This function SHALL take one argument of data-type
  - 4225 “http://www.w3.org/2001/XMLSchema#integer” and SHALL promote its value to an element of
  - 4226 data-type “http://www.w3.org/2001/XMLSchema#double” with the same numeric value. If the
  - 4227 integer argument is outside the range which can be represented by a double, the result SHALL
  - 4228 be Indeterminate, with the status code “urn:oasis:names:tc:xacml:1.0:status:processing-error”.

### 4229 **A.3.5 Logical functions**

4230 This section contains the specification for logical functions that operate on arguments of data-type  
4231 “http://www.w3.org/2001/XMLSchema#boolean”.

- 4232 • urn:oasis:names:tc:xacml:1.0:function:or
  - 4233 This function SHALL return "False" if it has no arguments and SHALL return "True" if at least one
  - 4234 of its arguments evaluates to "True". The order of evaluation SHALL be from first argument to
  - 4235 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",
  - 4236 leaving the rest of the arguments unevaluated.
- 4237 • urn:oasis:names:tc:xacml:1.0:function:and
  - 4238 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of its
  - 4239 arguments evaluates to "False". The order of evaluation SHALL be from first argument to last.
  - 4240 The evaluation SHALL stop with a result of "False" if any argument evaluates to "False", leaving
  - 4241 the rest of the arguments unevaluated.
- 4242 • urn:oasis:names:tc:xacml:1.0:function:n-of
  - 4243 The first argument to this function SHALL be of data-type
  - 4244 http://www.w3.org/2001/XMLSchema#integer. The remaining arguments SHALL be of data-type

4245 http://www.w3.org/2001/XMLSchema#boolean. The first argument specifies the minimum  
4246 number of the remaining arguments that MUST evaluate to "True" for the expression to be  
4247 considered "True". If the first argument is 0, the result SHALL be "True". If the number of  
4248 arguments after the first one is less than the value of the first argument, then the expression  
4249 SHALL result in "Indeterminate". The order of evaluation SHALL be: first evaluate the integer  
4250 value, and then evaluate each subsequent argument. The evaluation SHALL stop and return  
4251 "True" if the specified number of arguments evaluate to "True". The evaluation of arguments  
4252 SHALL stop if it is determined that evaluating the remaining arguments will not satisfy the  
4253 requirement.

4254 • urn:oasis:names:tc:xacml:1.0:function:not

4255 This function SHALL take one argument of data-type  
4256 "http://www.w3.org/2001/XMLSchema#boolean". If the argument evaluates to "True", then the  
4257 result of the expression SHALL be "False". If the argument evaluates to "False", then the result  
4258 of the expression SHALL be "True".

4259 Note: When evaluating and, or, or n-of, it MAY NOT be necessary to attempt a full evaluation of each  
4260 argument in order to determine whether the evaluation of the argument would result in "Indeterminate".  
4261 Analysis of the argument regarding the availability of its *attributes*, or other analysis regarding errors,  
4262 such as "divide-by-zero", may render the argument error free. Such arguments occurring in the  
4263 expression in a position after the evaluation is stated to stop need not be processed.

### 4264 A.3.6 Numeric comparison functions

4265 These functions form a minimal set for comparing two numbers, yielding a Boolean result. For doubles  
4266 they SHALL comply with the rules governed by IEEE 754 [IEEE754].

- 4267 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
- 4268 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
- 4269 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than
- 4270 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
- 4271 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than
- 4272 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal
- 4273 • urn:oasis:names:tc:xacml:1.0:function:double-less-than
- 4274 • urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal

### 4275 A.3.7 Date and time arithmetic functions

4276 These functions perform arithmetic operations with date and time.

4277 • urn:oasis:names:tc:xacml:3.0:function:date-time-add-day-time-duration

4278 This function SHALL take two arguments, the first SHALL be of data-type  
4279 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be of data-type  
4280 "http://www.w3.org/2001/XMLSchema#dayTimeDuration". It SHALL return a result of  
4281 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by  
4282 adding the second argument to the first argument according to the specification of adding  
4283 durations to date and time [XS] Appendix E.

4284 • urn:oasis:names:tc:xacml:3.0:function:date-time-add-year-month-duration

4285 This function SHALL take two arguments, the first SHALL be a  
4286 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a  
4287 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of  
4288 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by  
4289 adding the second argument to the first argument according to the specification of adding  
4290 durations to date and time [XS] Appendix E.

- 4291 • urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration
  - 4292 This function SHALL take two arguments, the first SHALL be a
  - 4293 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
  - 4294 "http://www.w3.org/2001/XMLSchema#dayTimeDuration". It SHALL return a result of
  - 4295 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive duration,
  - 4296 then this function SHALL return the value by adding the corresponding negative duration, as per
  - 4297 the specification [XS] Appendix E. If the second argument is a negative duration, then the result
  - 4298 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-
  - 4299 dayTimeDuration" had been applied to the corresponding positive duration.
- 4300 • urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration
  - 4301 This function SHALL take two arguments, the first SHALL be a
  - 4302 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
  - 4303 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
  - 4304 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive duration,
  - 4305 then this function SHALL return the value by adding the corresponding negative duration, as per
  - 4306 the specification [XS] Appendix E. If the second argument is a negative duration, then the result
  - 4307 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-
  - 4308 yearMonthDuration" had been applied to the corresponding positive duration.
- 4309 • urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration
  - 4310 This function SHALL take two arguments, the first SHALL be a
  - 4311 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
  - 4312 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
  - 4313 "http://www.w3.org/2001/XMLSchema#date". This function SHALL return the value by adding the
  - 4314 second argument to the first argument according to the specification of adding duration to date
  - 4315 [XS] Appendix E.
- 4316 • urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration
  - 4317 This function SHALL take two arguments, the first SHALL be a
  - 4318 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
  - 4319 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
  - 4320 "http://www.w3.org/2001/XMLSchema#date". If the second argument is a positive duration, then
  - 4321 this function SHALL return the value by adding the corresponding negative duration, as per the
  - 4322 specification [XS] Appendix E. If the second argument is a negative duration, then the result
  - 4323 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration"
  - 4324 had been applied to the corresponding positive duration.

### 4325 A.3.8 Non-numeric comparison functions

4326 These functions perform comparison operations on two arguments of non-numerical types.

- 4327 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than
  - 4328 This function SHALL take two arguments of data-type
  - 4329 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
  - 4330 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
  - 4331 argument is lexicographically strictly greater than the second argument. Otherwise, it SHALL
  - 4332 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
  - 4333 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].
- 4334 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal
  - 4335 This function SHALL take two arguments of data-type
  - 4336 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
  - 4337 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
  - 4338 argument is lexicographically greater than or equal to the second argument. Otherwise, it SHALL
  - 4339 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
  - 4340 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].

- 4341 • urn:oasis:names:tc:xacml:1.0:function:string-less-than  
4342 This function SHALL take two arguments of data-type  
4343 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4344 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only the first  
4345 argument is lexicographically strictly less than the second argument. Otherwise, it SHALL return  
4346 "False". The comparison SHALL use Unicode codepoint collation, as defined for the identifier  
4347 http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.
- 4348 • urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal  
4349 This function SHALL take two arguments of data-type  
4350 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4351 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only the first  
4352 argument is lexicographically less than or equal to the second argument. Otherwise, it SHALL  
4353 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the  
4354 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.
- 4355 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than  
4356 This function SHALL take two arguments of data-type  
4357 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4358 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4359 argument is greater than the second argument according to the order relation specified for  
4360 "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it SHALL return  
4361 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does  
4362 not. In such cases, the time-in-range function should be used.
- 4363 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal  
4364 This function SHALL take two arguments of data-type  
4365 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4366 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4367 argument is greater than or equal to the second argument according to the order relation  
4368 specified for "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it  
4369 SHALL return "False". Note: it is illegal to compare a time that includes a time-zone value with  
4370 one that does not. In such cases, the time-in-range function should be used.
- 4371 • urn:oasis:names:tc:xacml:1.0:function:time-less-than  
4372 This function SHALL take two arguments of data-type  
4373 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4374 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4375 argument is less than the second argument according to the order relation specified for  
4376 "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it SHALL return  
4377 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does  
4378 not. In such cases, the time-in-range function should be used.
- 4379 • urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal  
4380 This function SHALL take two arguments of data-type  
4381 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4382 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4383 argument is less than or equal to the second argument according to the order relation specified  
4384 for "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it SHALL return  
4385 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does  
4386 not. In such cases, the time-in-range function should be used.
- 4387 • urn:oasis:names:tc:xacml:2.0:function:time-in-range  
4388 This function SHALL take three arguments of data-type  
4389 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4390 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument falls  
4391 in the range defined inclusively by the second and third arguments. Otherwise, it SHALL return

4392 "False". Regardless of its value, the third argument SHALL be interpreted as a time that is equal  
4393 to, or later than by less than twenty-four hours, the second argument. If no time zone is provided  
4394 for the first argument, it SHALL use the default time zone at the **context handler**. If no time zone  
4395 is provided for the second or third arguments, then they SHALL use the time zone from the first  
4396 argument.

- 4397 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than

4398 This function SHALL take two arguments of data-type  
4399 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4400 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4401 argument is greater than the second argument according to the order relation specified for  
4402 "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7. Otherwise, it  
4403 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an  
4404 implicit time-zone value SHALL be assigned, as described in [XS].

- 4405 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal

4406 This function SHALL take two arguments of data-type  
4407 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4408 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4409 argument is greater than or equal to the second argument according to the order relation  
4410 specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7.  
4411 Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone  
4412 value, then an implicit time-zone value SHALL be assigned, as described in [XS].

- 4413 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than

4414 This function SHALL take two arguments of data-type  
4415 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4416 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4417 argument is less than the second argument according to the order relation specified for  
4418 "http://www.w3.org/2001/XMLSchema#dateTime" by [XS, part 2, section 3.2.7]. Otherwise, it  
4419 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an  
4420 implicit time-zone value SHALL be assigned, as described in [XS].

- 4421 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal

4422 This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#  
4423 dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL  
4424 return "True" if and only if the first argument is less than or equal to the second argument  
4425 according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" by  
4426 [XS] part 2, section 3.2.7. Otherwise, it SHALL return "False". Note: if a dateTime value does  
4427 not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described  
4428 in [XS].

- 4429 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than

4430 This function SHALL take two arguments of data-type  
4431 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4432 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4433 argument is greater than the second argument according to the order relation specified for  
4434 "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it SHALL  
4435 return "False". Note: if a date value does not include a time-zone value, then an implicit time-  
4436 zone value SHALL be assigned, as described in [XS].

- 4437 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal

4438 This function SHALL take two arguments of data-type  
4439 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4440 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4441 argument is greater than or equal to the second argument according to the order relation  
4442 specified for "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9.

4443           Otherwise, it SHALL return “False”. Note: if a date value does not include a time-zone value,  
4444           then an implicit time-zone value SHALL be assigned, as described in [XS].

4445   • urn:oasis:names:tc:xacml:1.0:function:date-less-than  
4446       This function SHALL take two arguments of data-type  
4447       “http://www.w3.org/2001/XMLSchema#date” and SHALL return an  
4448       “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the first  
4449       argument is less than the second argument according to the order relation specified for  
4450       “http://www.w3.org/2001/XMLSchema#date” by [XS] part 2, section 3.2.9. Otherwise, it SHALL  
4451       return “False”. Note: if a date value does not include a time-zone value, then an implicit time-  
4452       zone value SHALL be assigned, as described in [XS].

4453   • urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal  
4454       This function SHALL take two arguments of data-type  
4455       “http://www.w3.org/2001/XMLSchema#date” and SHALL return an  
4456       “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return “True” if and only if the first  
4457       argument is less than or equal to the second argument according to the order relation specified  
4458       for “http://www.w3.org/2001/XMLSchema#date” by [XS] part 2, section 3.2.9. Otherwise, it  
4459       SHALL return “False”. Note: if a date value does not include a time-zone value, then an implicit  
4460       time-zone value SHALL be assigned, as described in [XS].

### 4461   **A.3.9 String functions**

4462   The following functions operate on strings and convert to and from other data types.

4463   • urn:oasis:names:tc:xacml:2.0:function:string-concatenate  
4464       This function SHALL take two or more arguments of data-type  
4465       “http://www.w3.org/2001/XMLSchema#string” and SHALL return a  
4466       “http://www.w3.org/2001/XMLSchema#string”. The result SHALL be the concatenation, in order,  
4467       of the arguments.

4468   • urn:oasis:names:tc:xacml:3.0:function:boolean-from-string  
4469       This function SHALL take one argument of data-type  
4470       “http://www.w3.org/2001/XMLSchema#string”, and SHALL return an  
4471       “http://www.w3.org/2001/XMLSchema#boolean”. The result SHALL be the string converted to a  
4472       boolean. If the argument is not a valid lexical representation of a boolean, then the result SHALL  
4473       be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

4474   • urn:oasis:names:tc:xacml:3.0:function:string-from-boolean  
4475       This function SHALL take one argument of data-type  
4476       “http://www.w3.org/2001/XMLSchema#boolean”, and SHALL return an  
4477       “http://www.w3.org/2001/XMLSchema#string”. The result SHALL be the boolean converted to a  
4478       string in the canonical form specified in [XS].

4479   • urn:oasis:names:tc:xacml:3.0:function:integer-from-string  
4480       This function SHALL take one argument of data-type  
4481       “http://www.w3.org/2001/XMLSchema#string”, and SHALL return an  
4482       “http://www.w3.org/2001/XMLSchema#integer”. The result SHALL be the string converted to an  
4483       integer. If the argument is not a valid lexical representation of an integer, then the result SHALL  
4484       be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.

4485   • urn:oasis:names:tc:xacml:3.0:function:string-from-integer  
4486       This function SHALL take one argument of data-type  
4487       “http://www.w3.org/2001/XMLSchema#integer”, and SHALL return an  
4488       “http://www.w3.org/2001/XMLSchema#string”. The result SHALL be the integer converted to a  
4489       string in the canonical form specified in [XS].

4490   • urn:oasis:names:tc:xacml:3.0:function:double-from-string

- 4491 This function SHALL take one argument of data-type  
4492 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4493 "http://www.w3.org/2001/XMLSchema#double". The result SHALL be the string converted to a  
4494 double. If the argument is not a valid lexical representation of a double, then the result SHALL be  
4495 Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
- 4496 • urn:oasis:names:tc:xacml:3.0:function:string-from-double  
4497 This function SHALL take one argument of data-type  
4498 "http://www.w3.org/2001/XMLSchema#double", and SHALL return an  
4499 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the double converted to a  
4500 string in the canonical form specified in **[XS]**.
  - 4501 • urn:oasis:names:tc:xacml:3.0:function:time-from-string  
4502 This function SHALL take one argument of data-type  
4503 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4504 "http://www.w3.org/2001/XMLSchema#time". The result SHALL be the string converted to a time.  
4505 If the argument is not a valid lexical representation of a time, then the result SHALL be  
4506 Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
  - 4507 • urn:oasis:names:tc:xacml:3.0:function:string-from-time  
4508 This function SHALL take one argument of data-type  
4509 "http://www.w3.org/2001/XMLSchema#time", and SHALL return an  
4510 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the time converted to a  
4511 string in the canonical form specified in **[XS]**.
  - 4512 • urn:oasis:names:tc:xacml:3.0:function:date-from-string  
4513 This function SHALL take one argument of data-type  
4514 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4515 "http://www.w3.org/2001/XMLSchema#date". The result SHALL be the string converted to a  
4516 date. If the argument is not a valid lexical representation of a date, then the result SHALL be  
4517 Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
  - 4518 • urn:oasis:names:tc:xacml:3.0:function:string-from-date  
4519 This function SHALL take one argument of data-type  
4520 "http://www.w3.org/2001/XMLSchema#date", and SHALL return an  
4521 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the date converted to a  
4522 string in the canonical form specified in **[XS]**.
  - 4523 • urn:oasis:names:tc:xacml:3.0:function:dateTime-from-string  
4524 This function SHALL take one argument of data-type  
4525 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4526 "http://www.w3.org/2001/XMLSchema#dateTime". The result SHALL be the string converted to a  
4527 dateTime. If the argument is not a valid lexical representation of a dateTime, then the result  
4528 SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
  - 4529 urn:oasis:names:tc:xacml:3.0:function:string-from-dateTime  
4530 This function SHALL take one argument of data-type  
4531 "http://www.w3.org/2001/XMLSchema#dateTime", and SHALL return an  
4532 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the dateTime converted to a  
4533 string in the canonical form specified in **[XS]**.
  - 4534 • urn:oasis:names:tc:xacml:3.0:function:anyURI-from-string  
4535 This function SHALL take one argument of data-type  
4536 "http://www.w3.org/2001/XMLSchema#string", and SHALL return a  
4537 "http://www.w3.org/2001/XMLSchema#anyURI". The result SHALL be the URI constructed by  
4538 converting the argument to an URI. If the argument is not a valid lexical representation of a URI,  
4539 then the result SHALL be Indeterminate with status code  
4540 urn:oasis:names:tc:xacml:1.0:status:syntax-error.

- 4541 • urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI  
 4542 This function SHALL take one argument of data-type  
 4543 "http://www.w3.org/2001/XMLSchema#anyURI", and SHALL return an  
 4544 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the URI converted to a  
 4545 string in the form it was originally represented in XML form.
- 4546 • urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-from-string  
 4547 This function SHALL take one argument of data-type  
 4548 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
 4549 "http://www.w3.org/2001/XMLSchema#dayTimeDuration ". The result SHALL be the string  
 4550 converted to a dayTimeDuration. If the argument is not a valid lexical representation of a  
 4551 dayTimeDuration, then the result SHALL be Indeterminate with status code  
 4552 urn:oasis:names:tc:xacml:1.0:status:syntax-error.
- 4553 • urn:oasis:names:tc:xacml:3.0:function:string-from-dayTimeDuration  
 4554 This function SHALL take one argument of data-type  
 4555 "http://www.w3.org/2001/XMLSchema#dayTimeDuration ", and SHALL return an  
 4556 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the dayTimeDuration  
 4557 converted to a string in the canonical form specified in **[XPathFunc]**.
- 4558 • urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-from-string  
 4559 This function SHALL take one argument of data-type  
 4560 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
 4561 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". The result SHALL be the string  
 4562 converted to a yearMonthDuration. If the argument is not a valid lexical representation of a  
 4563 yearMonthDuration, then the result SHALL be Indeterminate with status code  
 4564 urn:oasis:names:tc:xacml:1.0:status:syntax-error.
- 4565 • urn:oasis:names:tc:xacml:3.0:function:string-from-yearMonthDuration  
 4566 This function SHALL take one argument of data-type  
 4567 "http://www.w3.org/2001/XMLSchema#yearMonthDuration", and SHALL return an  
 4568 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the yearMonthDuration  
 4569 converted to a string in the canonical form specified in **[XPathFunc]**.
- 4570 • urn:oasis:names:tc:xacml:3.0:function:x500Name-from-string  
 4571 This function SHALL take one argument of data-type  
 4572 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
 4573 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". The result SHALL be the string converted  
 4574 to an x500Name. If the argument is not a valid lexical representation of a X500Name, then the  
 4575 result SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
- 4576 • urn:oasis:names:tc:xacml:3.0:function:string-from-x500Name  
 4577 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:1.0:data-  
 4578 type:x500Name", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result  
 4579 SHALL be the x500Name converted to a string in the form it was originally represented in XML  
 4580 form..
- 4581 • urn:oasis:names:tc:xacml:3.0:function:rfc822Name-from-string  
 4582 This function SHALL take one argument of data-type  
 4583 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
 4584 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". The result SHALL be the string converted  
 4585 to an rfc822Name. If the argument is not a valid lexical representation of an rfc822Name, then the  
 4586 result SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
- 4587 • urn:oasis:names:tc:xacml:3.0:function:string-from-rfc822Name  
 4588 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:1.0:data-  
 4589 type:rfc822Name", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The

- 4590 result SHALL be the rfc822Name converted to a string in the form it was originally represented in  
4591 XML form.
- 4592 • urn:oasis:names:tc:xacml:3.0:function:ipAddress-from-string
    - 4593 This function SHALL take one argument of data-type
    - 4594 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
    - 4595 "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". The result SHALL be the string converted to
    - 4596 an ipAddress. If the argument is not a valid lexical representation of an ipAddress, then the result
    - 4597 SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
  - 4598 • urn:oasis:names:tc:xacml:3.0:function:string-from-ipAddress
    - 4599 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-
    - 4600 type:ipAddress", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result
    - 4601 SHALL be the ipAddress converted to a string in the form it was originally represented in XML
    - 4602 form.
  - 4603 • urn:oasis:names:tc:xacml:3.0:function:dnsName-from-string
    - 4604 This function SHALL take one argument of data-type
    - 4605 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
    - 4606 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName". The result SHALL be the string converted to
    - 4607 a dnsName. If the argument is not a valid lexical representation of a dnsName, then the result
    - 4608 SHALL be Indeterminate with status code urn:oasis:names:tc:xacml:1.0:status:syntax-error.
  - 4609 • urn:oasis:names:tc:xacml:3.0:function:string-from-dnsName
    - 4610 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-
    - 4611 type:dnsName", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result
    - 4612 SHALL be the dnsName converted to a string in the form it was originally represented in XML
    - 4613 form.
  - 4614 • urn:oasis:names:tc:xacml:3.0:function:string-starts-with
    - 4615 This function SHALL take two arguments of data-type
    - 4616 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
    - 4617 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
    - 4618 begins with the first string, and false otherwise. Equality testing SHALL be done as defined for
    - 4619 urn:oasis:names:tc:xacml:1.0:function:string-equal.
  - 4620 • urn:oasis:names:tc:xacml:3.0:function:anyURI-starts-with
    - 4621 This function SHALL take a first argument of data-
    - 4622 type"http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
    - 4623 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
    - 4624 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
    - 4625 to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI begins with the string,
    - 4626 and false otherwise. Equality testing SHALL be done as defined for
    - 4627 urn:oasis:names:tc:xacml:1.0:function:string-equal.
  - 4628 • urn:oasis:names:tc:xacml:3.0:function:string-ends-with
    - 4629 This function SHALL take two arguments of data-type
    - 4630 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
    - 4631 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
    - 4632 ends with the first string, and false otherwise. Equality testing SHALL be done as defined for
    - 4633 urn:oasis:names:tc:xacml:1.0:function:string-equal.
  - 4634 • urn:oasis:names:tc:xacml:3.0:function:anyURI-ends-with
    - 4635 This function SHALL take a first argument of data-type
    - 4636 "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
    - 4637 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
    - 4638 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
    - 4639 to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI ends with the string,

- 4640 and false otherwise. Equality testing SHALL be done as defined for  
4641 urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4642 • urn:oasis:names:tc:xacml:3.0:function:string-contains  
4643 This function SHALL take two arguments of data-type  
4644 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a  
4645 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string  
4646 contains the first string, and false otherwise. Equality testing SHALL be done as defined for  
4647 urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4648 • urn:oasis:names:tc:xacml:3.0:function:anyURI-contains  
4649 This function SHALL take a first argument of data-type  
4650 "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type  
4651 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a  
4652 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted  
4653 to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI contains the string, and  
4654 false otherwise. Equality testing SHALL be done as defined for  
4655 urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4656 • urn:oasis:names:tc:xacml:3.0:function:string-substring  
4657 This function SHALL take a first argument of data-type  
4658 "http://www.w3.org/2001/XMLSchema#string" and a second and a third argument of type  
4659 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return a  
4660 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first  
4661 argument beginning at the position given by the second argument and ending at the position  
4662 before the position given by the third argument. The first character of the string has position zero.  
4663 The negative integer value -1 given for the third arguments indicates the end of the string. If the  
4664 second or third arguments are out of bounds, then the function MUST evaluate to Indeterminate  
4665 with a status code of urn:oasis:names:tc:xacml:1.0:status:processing-error.
- 4666 • urn:oasis:names:tc:xacml:3.0:function:anyURI-substring  
4667 This function SHALL take a first argument of data-type  
4668 "http://www.w3.org/2001/XMLSchema#anyURI" and a second and a third argument of type  
4669 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return a  
4670 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first  
4671 argument converted to a string with urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI  
4672 beginning at the position given by the second argument and ending at the position before the  
4673 position given by the third argument. The first character of the URI converted to a string has  
4674 position zero. The negative integer value -1 given for the third arguments indicates the end of the  
4675 string. If the second or third arguments are out of bounds, then the function MUST evaluate to  
4676 Indeterminate with a status code of  
4677 urn:oasis:names:tc:xacml:1.0:status:processing-error. If the resulting substring  
4678 is not syntactically a valid URI, then the function MUST evaluate to Indeterminate with a status  
4679 code of urn:oasis:names:tc:xacml:1.0:status:processing-error.

4680

### 4681 A.3.10 Bag functions

4682 These functions operate on a **bag** of 'type' values, where type is one of the primitive data-types, and x.x  
4683 is a version of XACML where the function has been defined. Some additional conditions defined for  
4684 each function below SHALL cause the expression to evaluate to "Indeterminate".

- 4685 • urn:oasis:names:tc:xacml:x.x:function:type-one-and-only  
4686 This function SHALL take a **bag** of 'type' values as an argument and SHALL return a value of  
4687 'type'. It SHALL return the only value in the **bag**. If the **bag** does not have one and only one  
4688 value, then the expression SHALL evaluate to "Indeterminate".

- 4689 • urn:oasis:names:tc:xacml:x.x:function:type-bag-size
- 4690     This function SHALL take a **bag** of 'type' values as an argument and SHALL return an
- 4691     "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.
- 4692 • urn:oasis:names:tc:xacml:x.x:function:type-is-in
- 4693     This function SHALL take an argument of 'type' as the first argument and a **bag** of 'type' values
- 4694     as the second argument and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".
- 4695     The function SHALL evaluate to "True" if and only if the first argument matches by the
- 4696     "urn:oasis:names:tc:xacml:x.x:function:type-equal" any value in the **bag**. Otherwise, it SHALL
- 4697     return "False".
- 4698 • urn:oasis:names:tc:xacml:x.x:function:type-bag
- 4699     This function SHALL take any number of arguments of 'type' and return a **bag** of 'type' values
- 4700     containing the values of the arguments. An application of this function to zero arguments SHALL
- 4701     produce an empty **bag** of the specified data-type.

### 4702 **A.3.11 Set functions**

4703 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**.

- 4704 • urn:oasis:names:tc:xacml:x.x:function:type-intersection
- 4705     This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
- 4706     **bag** of 'type' values such that it contains only elements that are common between the two **bags**,
- 4707     which is determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal". No duplicates, as
- 4708     determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal", SHALL exist in the result.
- 4709 • urn:oasis:names:tc:xacml:x.x:function:type-at-least-one-member-of
- 4710     This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
- 4711     "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if and
- 4712     only if at least one element of the first argument is contained in the second argument as
- 4713     determined by "urn:oasis:names:tc:xacml:x.x:function:type-is-in".
- 4714 • urn:oasis:names:tc:xacml:x.x:function:type-union
- 4715     This function SHALL take two or more arguments that are both a **bag** of 'type' values. The
- 4716     expression SHALL return a **bag** of 'type' such that it contains all elements of all the argument
- 4717     **bags**. No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
- 4718     SHALL exist in the result.
- 4719 • urn:oasis:names:tc:xacml:x.x:function:type-subset
- 4720     This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
- 4721     "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
- 4722     argument is a subset of the second argument. Each argument SHALL be considered to have had
- 4723     its duplicates removed, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
- 4724     before the subset calculation.
- 4725 • urn:oasis:names:tc:xacml:x.x:function:type-set-equals
- 4726     This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
- 4727     "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return the result of applying
- 4728     "urn:oasis:names:tc:xacml:1.0:function:and" to the application of
- 4729     "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the first and second arguments and the
- 4730     application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the second and first
- 4731     arguments.

### 4732 **A.3.12 Higher-order bag functions**

4733 This section describes functions in XACML that perform operations on **bags** such that functions may be

4734 applied to the **bags** in general.

4735 • urn:oasis:names:tc:xacml:3.0:function:any-of

4736 This function applies a Boolean function between specific primitive values and a **bag** of values,  
4737 and SHALL return "True" if and only if the **predicate** is "True" for at least one element of the **bag**.

4738 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4739 be an <Function> element that names a Boolean function that takes n arguments of primitive  
4740 types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-  
4741 types and one SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated as  
4742 if the function named in the <Function> argument were applied to the n-1 non-bag arguments  
4743 and each element of the bag argument and the results are combined with  
4744 "urn:oasis:names:tc:xacml:1.0:function:or".

4745 For example, the following expression SHALL return "True":

```
4746 <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">  
4747   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4748   <AttributeValue  
4749     DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4750   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4751     <AttributeValue  
4752       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4753     <AttributeValue  
4754       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4755     <AttributeValue  
4756       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4757     <AttributeValue  
4758       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4759   </Apply>  
4760 </Apply>
```

4761 This expression is "True" because the first argument is equal to at least one of the elements of  
4762 the **bag**, according to the function.

4763 • urn:oasis:names:tc:xacml:3.0:function:all-of

4764 This function applies a Boolean function between a specific primitive value and a **bag** of values,  
4765 and returns "True" if and only if the **predicate** is "True" for every element of the **bag**.

4766 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4767 be a <Function> element that names a Boolean function that takes n arguments of primitive  
4768 types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-  
4769 types and one SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated as  
4770 if the function named in the <Function> argument were applied to the n-1 non-bag arguments  
4771 and each element of the bag argument and the results are combined with  
4772 "urn:oasis:names:tc:xacml:1.0:function:and".

4773 For example, the following expression SHALL evaluate to "True":

```
4774 <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:all-of">  
4775   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-  
4776     greater-than"/>  
4777   <AttributeValue  
4778     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>  
4779   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4780     <AttributeValue  
4781       DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>  
4782     <AttributeValue  
4783       DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>  
4784     <AttributeValue  
4785       DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>  
4786     <AttributeValue  
4787       DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>  
4788   </Apply>  
4789 </Apply>
```

4790 This expression is "True" because the first argument (10) is greater than all of the elements of the  
4791 **bag** (9,3,4 and 2).

4792 • urn:oasis:names:tc:xacml:3.0:function:any-of-any

4793 This function applies a Boolean function on each tuple from the cross product on all bags  
4794 arguments, and returns "True" if and only if the **predicate** is "True" for at least one inside-function  
4795 call.

4796 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4797 be an <Function> element that names a Boolean function that takes n arguments. The  
4798 remaining arguments are either primitive data types or bags of primitive types. The expression  
4799 SHALL be evaluated as if the function named in the <Function> argument was applied between  
4800 every tuple of the cross product on all bags and the primitive values, and the results were  
4801 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the result of  
4802 the expression SHALL be "True" if and only if the applied **predicate** is "True" for at least one  
4803 function call on the tuples from the **bags** and primitive values.

4804 For example, the following expression SHALL evaluate to "True":

```
4805 <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of-any">  
4806   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4807   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4808     <AttributeValue  
4809     DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4810     <AttributeValue  
4811     DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>  
4812   </Apply>  
4813   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4814     <AttributeValue  
4815     DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4816     <AttributeValue  
4817     DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4818     <AttributeValue  
4819     DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4820     <AttributeValue  
4821     DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4822   </Apply>  
4823 </Apply>
```

4824 This expression is "True" because at least one of the elements of the first **bag**, namely "Ringo", is  
4825 equal to at least one of the elements of the second **bag**.

4826 • urn:oasis:names:tc:xacml:1.0:function:all-of-any

4827 This function applies a Boolean function between the elements of two **bags**. The expression  
4828 SHALL be "True" if and only if the supplied **predicate** is "True" between each element of the first  
4829 **bag** and any element of the second **bag**.

4830 This function SHALL take three arguments. The first argument SHALL be an <Function>  
4831 element that names a Boolean function that takes two arguments of primitive types. The second  
4832 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a  
4833 primitive data-type. The expression SHALL be evaluated as if the  
4834 "urn:oasis:names:tc:xacml:3.0:function:any-of" function had been applied to each value of the first  
4835 **bag** and the whole of the second **bag** using the supplied xacml:Function, and the results were  
4836 then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4837 For example, the following expression SHALL evaluate to "True":

```
4838 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">  
4839   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-  
4840   greater-than"/>  
4841   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4842     <AttributeValue  
4843     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
```

```

4844         <AttributeValue
4845         DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4846         </Apply>
4847         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4848         <AttributeValue
4849         DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4850         <AttributeValue
4851         DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4852         <AttributeValue
4853         DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4854         <AttributeValue
4855         DataType="http://www.w3.org/2001/XMLSchema#integer">19</AttributeValue>
4856         </Apply>
4857     </Apply>

```

4858 This expression is "True" because each of the elements of the first **bag** is greater than at least  
4859 one of the elements of the second **bag**.

4860 • urn:oasis:names:tc:xacml:1.0:function:any-of-all

4861 This function applies a Boolean function between the elements of two **bags**. The expression  
4862 SHALL be "True" if and only if the supplied **predicate** is "True" between each element of the  
4863 second **bag** and any element of the first **bag**.

4864 This function SHALL take three arguments. The first argument SHALL be an <Function>  
4865 element that names a Boolean function that takes two arguments of primitive types. The second  
4866 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a  
4867 primitive data-type. The expression SHALL be evaluated as if the  
4868 "urn:oasis:names:tc:xacml:3.0:function:any-of" function had been applied to each value of the  
4869 second **bag** and the whole of the first **bag** using the supplied xacml:Function, and the results  
4870 were then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4871 For example, the following expression SHALL evaluate to "True":

```

4872 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4873     <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-
4874     greater-than"/>
4875     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4876     <AttributeValue
4877     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4878     <AttributeValue
4879     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4880     </Apply>
4881     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4882     <AttributeValue
4883     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4884     <AttributeValue
4885     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4886     <AttributeValue
4887     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4888     <AttributeValue
4889     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4890     </Apply>
4891 </Apply>

```

4892 This expression is "True" because, for all of the values in the second **bag**, there is a value in the  
4893 first **bag** that is greater.

4894 • urn:oasis:names:tc:xacml:1.0:function:all-of-all

4895 This function applies a Boolean function between the elements of two **bags**. The expression  
4896 SHALL be "True" if and only if the supplied **predicate** is "True" between each and every element  
4897 of the first **bag** collectively against all the elements of the second **bag**.

4898 This function SHALL take three arguments. The first argument SHALL be an <Function>  
4899 element that names a Boolean function that takes two arguments of primitive types. The second

4900 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a  
4901 primitive data-type. The expression is evaluated as if the function named in the <Function>  
4902 element were applied between every element of the second argument and every element of the  
4903 third argument and the results were combined using "urn:oasis:names:tc:xacml:1.0:function:and".  
4904 The semantics are that the result of the expression is "True" if and only if the applied **predicate** is  
4905 "True" for all elements of the first **bag** compared to all the elements of the second **bag**.

4906 For example, the following expression SHALL evaluate to "True":

```
4907 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">  
4908   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-  
4909   greater-than"/>  
4910   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4911     <AttributeValue  
4912     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>  
4913     <AttributeValue  
4914     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>  
4915   </Apply>  
4916   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4917     <AttributeValue  
4918     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>  
4919     <AttributeValue  
4920     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>  
4921     <AttributeValue  
4922     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>  
4923     <AttributeValue  
4924     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>  
4925   </Apply>  
4926 </Apply>
```

4927 This expression is "True" because all elements of the first **bag**, "5" and "6", are each greater than  
4928 all of the integer values "1", "2", "3", "4" of the second **bag**.

4929 • urn:oasis:names:tc:xacml:3.0:function:map

4930 This function converts a **bag** of values to another **bag** of values.

4931 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4932 be a <Function> element naming a function that takes a n arguments of a primitive data-type  
4933 and returns a value of a primitive data-type Under the remaining n arguments, n-1 parameters  
4934 SHALL be values of primitive data-types and one SHALL be a **bag** of a primitive data-type. The  
4935 expression SHALL be evaluated as if the function named in the <Function> argument were  
4936 applied to the n-1 non-bag arguments and each element of the bag argument and resulting in a  
4937 **bag** of the converted value. The result SHALL be a **bag** of the primitive data-type that is returned  
4938 by the function named in the <xacml:Function> element.

4939 For example, the following expression,

```
4940 <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:map">  
4941   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-  
4942   normalize-to-lower-case">  
4943   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4944     <AttributeValue  
4945     DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>  
4946     <AttributeValue  
4947     DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>  
4948   </Apply>  
4949 </Apply>
```

4950 evaluates to a **bag** containing "hello" and "world!".

### 4951 A.3.13 Regular-expression-based functions

4952 These functions operate on various types using regular expressions and evaluate to  
4953 "http://www.w3.org/2001/XMLSchema#boolean".

- 4954 • urn:oasis:names:tc:xacml:1.0:function:string-regexp-match
- 4955 This function decides a regular expression match. It SHALL take two arguments of  
 4956 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
 4957 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
 4958 expression and the second argument SHALL be a general string. The function specification  
 4959 SHALL be that of the "xf:matches" function with the arguments reversed [XF] Section 7.6.2.
- 4960 • urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match
- 4961 This function decides a regular expression match. It SHALL take two arguments; the first is of  
 4962 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
 4963 "http://www.w3.org/2001/XMLSchema#anyURI". It SHALL return an  
 4964 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
 4965 expression and the second argument SHALL be a URI. The function SHALL convert the second  
 4966 argument to type "http://www.w3.org/2001/XMLSchema#string" with  
 4967 urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI, then apply  
 4968 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4969 • urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match
- 4970 This function decides a regular expression match. It SHALL take two arguments; the first is of  
 4971 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
 4972 "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". It SHALL return an  
 4973 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
 4974 expression and the second argument SHALL be an IPv4 or IPv6 address. The function SHALL  
 4975 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string" with  
 4976 urn:oasis:names:tc:xacml:3.0:function:string-from-ipAddress, then apply  
 4977 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4978 • urn:oasis:names:tc:xacml:2.0:function:dnsName-regexp-match
- 4979 This function decides a regular expression match. It SHALL take two arguments; the first is of  
 4980 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
 4981 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName". It SHALL return an  
 4982 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
 4983 expression and the second argument SHALL be a DNS name. The function SHALL convert the  
 4984 second argument to type "http://www.w3.org/2001/XMLSchema#string" with  
 4985 urn:oasis:names:tc:xacml:3.0:function:string-from-dnsName, then apply  
 4986 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4987 • urn:oasis:names:tc:xacml:2.0:function:rfc822Name-regexp-match
- 4988 This function decides a regular expression match. It SHALL take two arguments; the first is of  
 4989 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
 4990 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". It SHALL return an  
 4991 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
 4992 expression and the second argument SHALL be an RFC 822 name. The function SHALL convert  
 4993 the second argument to type "http://www.w3.org/2001/XMLSchema#string" with  
 4994 urn:oasis:names:tc:xacml:3.0:function:string-from-rfc822Name, then apply  
 4995 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4996 • urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match
- 4997 This function decides a regular expression match. It SHALL take two arguments; the first is of  
 4998 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
 4999 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". It SHALL return an  
 5000 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
 5001 expression and the second argument SHALL be an X.500 directory name. The function SHALL  
 5002 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string" with  
 5003 urn:oasis:names:tc:xacml:3.0:function:string-from-x500Name, then apply  
 5004 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

### 5005 **A.3.14 Special match functions**

5006 These functions operate on various types and evaluate to  
5007 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching algorithm.

- 5008 • urn:oasis:names:tc:xacml:1.0:function:x500Name-match

5009 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name"  
5010 and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It shall return "True" if and  
5011 only if the first argument matches some terminal sequence of RDNs from the second argument  
5012 when compared using x500Name-equal.

5013 As an example (non-normative), if the first argument is "O=Medico Corp,C=US" and the second  
5014 argument is "cn=John Smith,o=Medico Corp, c=US", then the function will return "True".

- 5015 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match

5016 This function SHALL take two arguments, the first is of data-type  
5017 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type  
5018 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an  
5019 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if the  
5020 first argument matches the second argument according to the following specification.

5021 An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The local-  
5022 part is case-sensitive, while the domain-part (which is usually a DNS name) is not case-sensitive.

5023 The second argument contains a complete rfc822Name. The first argument is a complete or  
5024 partial rfc822Name used to select appropriate values in the second argument as follows.

5025 In order to match a particular address in the second argument, the first argument must specify the  
5026 complete mail address to be matched. For example, if the first argument is  
5027 "Anderson@sun.com", this matches a value in the second argument of "Anderson@sun.com"  
5028 and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com", "anderson@sun.com" or  
5029 "Anderson@east.sun.com".

5030 In order to match any address at a particular domain in the second argument, the first argument  
5031 must specify only a domain name (usually a DNS name). For example, if the first argument is  
5032 "sun.com", this matches a value in the second argument of "Anderson@sun.com" or  
5033 "Baxter@SUN.COM", but not "Anderson@east.sun.com".

5034 In order to match any address in a particular domain in the second argument, the first argument  
5035 must specify the desired domain-part with a leading ".". For example, if the first argument is  
5036 ".east.sun.com", this matches a value in the second argument of "Anderson@east.sun.com" and  
5037 "anne.anderson@ISRG.EAST.SUN.COM" but not "Anderson@sun.com".

### 5038 **A.3.15 XPath-based functions**

5039 This section specifies functions that take XPath expressions for arguments. An XPath expression  
5040 evaluates to a node-set, which is a set of XML nodes that match the expression. A node or node-set is  
5041 not in the formal data-type system of XACML. All comparison or other operations on node-sets are  
5042 performed in isolation of the particular function specified. The context nodes and namespace mappings  
5043 of the XPath expressions are defined by the XPath data-type, see section B.3. The following functions  
5044 are defined:

- 5045 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-count

5046 This function SHALL take an "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" as an  
5047 argument and evaluates to an "http://www.w3.org/2001/XMLSchema#integer". The value  
5048 returned from the function SHALL be the count of the nodes within the node-set that match the  
5049 given XPath expression. If the <Content> element of the category to which the XPath  
5050 expression applies to is not present in the request, this function SHALL return a value of zero.

- 5051 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal

5052 This function SHALL take two “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression”  
5053 arguments and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. The function  
5054 SHALL return "True" if any of the XML nodes in the node-set matched by the first argument  
5055 equals any of the XML nodes in the node-set matched by the second argument. Two nodes are  
5056 considered equal if they have the same identity. If the <Content> element of the category to  
5057 which either XPath expression applies to is not present in the request, this function SHALL return  
5058 a value of “False”.

5059 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

5060 This function SHALL take two “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression”  
5061 arguments and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function  
5062 SHALL evaluate to "True" if one of the following two conditions is satisfied: (1) Any of the XML  
5063 nodes in the node-set matched by the first argument is equal to any of the XML nodes in the  
5064 node-set matched by the second argument; (2) any node below any of the XML nodes in the  
5065 node-set matched by the first argument is equal to any of the XML nodes in the node-set  
5066 matched by the second argument. Two nodes are considered equal if they have the same  
5067 identity. If the <Content> element of the category to which either XPath expression applies to is  
5068 not present in the request, this function SHALL return a value of “False”.

5069 NOTE: The first **condition** is equivalent to "xpath-node-equal", and guarantees that "xpath-node-equal" is  
5070 a special case of "xpath-node-match".

### 5071 A.3.16 Other functions

5072 • urn:oasis:names:tc:xacml:3.0:function:access-permitted

5073 This function SHALL take an “http://www.w3.org/2001/XMLSchema#anyURI” and an  
5074 “http://www.w3.org/2001/XMLSchema#string” as arguments. The first argument SHALL be  
5075 interpreted as an **attribute** category. The second argument SHALL be interpreted as the XML  
5076 content of an <Attributes> element with *Category* equal to the first argument. The function  
5077 evaluates to an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return  
5078 "True" if and only if the **policy** evaluation described below returns the value of "Permit".

5079 The following evaluation is described as if the **context** is actually instantiated, but it is only  
5080 required that an equivalent result be obtained.

5081 The function SHALL construct a new **context**, by copying all the information from the current  
5082 **context**, omitting any <Attributes> element with *Category* equal to the first argument. The  
5083 second function argument SHALL be added to the **context** as the content of an <Attributes>  
5084 element with *Category* equal to the first argument.

5085 The function SHALL invoke a complete **policy** evaluation using the newly constructed **context**.  
5086 This evaluation SHALL be completely isolated from the evaluation which invoked the function, but  
5087 shall use all current **policies** and combining algorithms, including any per request **policies**.

5088 The **PDP** SHALL detect any loop which may occur if successive evaluations invoke this function  
5089 by counting the number of total invocations of any instance of this function during any single initial  
5090 invocation of the **PDP**. If the total number of invocations exceeds the bound for such invocations,  
5091 the initial invocation of this function evaluates to Indeterminate with a  
5092 “urn:oasis:names:tc:xacml:1.0:status:processing-error” status code. Also, see the security  
5093 considerations in section 9.1.8.

### 5094 A.3.17 Extension functions and primitive types

5095 Functions and primitive types are specified by string identifiers allowing for the introduction of functions in  
5096 addition to those specified by XACML. This approach allows one to extend the XACML module with  
5097 special functions and special primitive data-types.

5098 In order to preserve the integrity of the XACML evaluation strategy, the result of an extension function  
5099 SHALL depend only on the values of its arguments. Global and hidden parameters SHALL NOT affect

5100 the evaluation of an expression. Functions SHALL NOT have side effects, as evaluation order cannot be  
5101 guaranteed in a standard way.

## 5102 **A.4 Functions, data types, attributes and algorithms planned for** 5103 **deprecation**

5104 The following functions, data types and algorithms have been defined by previous versions of XACML  
5105 and newer and better alternatives are defined in XACML 3.0. Their use is discouraged for new use and  
5106 they are candidates for deprecation in future versions of XACML.

5107 The following xpath based functions have been replaced with equivalent functions which use the new  
5108 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression datatype instead of strings.

- 5109 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-count
  - 5110 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-count
- 5111 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal
  - 5112 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal
- 5113 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-match
  - 5114 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

5115 The following URI and string concatenation function has been replaced with a string to URI conversion  
5116 function, which allows the use of the general string functions with URI through string conversion.

- 5117 • urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate
  - 5118 • Replaced by urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI

5119 The following identifiers have been replaced with official identifiers defined by W3C.

- 5120 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
  - 5121 • Replaced with <http://www.w3.org/2001/XMLSchema#dayTimeDuration>
- 5122 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
  - 5123 • Replaced with <http://www.w3.org/2001/XMLSchema#yearMonthDuration>

5124 The following functions have been replaced with functions which use the updated dayTimeDuration and  
5125 yearMonthDuration data types.

- 5126 • urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal
  - 5127 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal
- 5128 • urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal
  - 5129 • Replaced with urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal
- 5130 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration
  - 5131 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration
- 5132 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration
  - 5133 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration
- 5134 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration
  - 5135 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration
- 5136 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration
  - 5137 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration
- 5138 • urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration
  - 5139 • Replaced with urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration
- 5140 • urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration
  - 5141 • Replaced with urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration

- 5142 The following attribute identifiers have been replaced with new identifiers
- 5143 • urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address
  - 5144 • Replaced with urn:oasis:names:tc:xacml:3.0:subject:authn-locality:ip-
  - 5145 address
  - 5146 • urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name
  - 5147 • Replaced with urn:oasis:names:tc:xacml:3.0:subject:authn-
  - 5148 locality:dns-name
  - 5149
- 5150 The following combining algorithms have been replaced with new variants which allow for better handling
- 5151 of “Indeterminate” results.
- 5152 • urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides
  - 5153 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides
  - 5154 • urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides
  - 5155 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides
  - 5156 • urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides
  - 5157 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides
  - 5158 • urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
  - 5159 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides
  - 5160 • urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides
  - 5161 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides
  - 5162 • urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides
  - 5163 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-overrides
  - 5164 • urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides
  - 5165 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-overrides
  - 5166 • urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides
  - 5167 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-overrides

---

## 5168 Appendix B. XACML identifiers (normative)

5169 This section defines standard identifiers for commonly used entities.

### 5170 B.1 XACML namespaces

5171 XACML is defined using this identifier.

5172 `urn:oasis:names:tc:xacml:3.0:core:schema`

### 5173 B.2 Attribute categories

5174 The following **attribute** category identifiers MUST be used when an XACML 2.0 or earlier **policy** or  
5175 request is translated into XACML 3.0.

5176 **Attributes** previously placed in the **Resource**, **Action**, and **Environment** sections of a request are  
5177 placed in an **attribute** category with the following identifiers respectively. It is RECOMMENDED that they  
5178 are used to list **attributes of resources**, **actions**, and the **environment** respectively when authoring  
5179 XACML 3.0 **policies** or requests.

5180 `urn:oasis:names:tc:xacml:3.0:attribute-category:resource`

5181 `urn:oasis:names:tc:xacml:3.0:attribute-category:action`

5182 `urn:oasis:names:tc:xacml:3.0:attribute-category:environment`

5183 **Attributes** previously placed in the **Subject** section of a request are placed in an **attribute** category  
5184 which is identical of the **subject** category in XACML 2.0, as defined below. It is RECOMMENDED that  
5185 they are used to list **attributes of subjects** when authoring XACML 3.0 **policies** or requests.

5186 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity in a  
5187 request chain. If **subject** category is not specified in XACML 2.0, this is the default translation value.

5188 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

5189 This identifier indicates the system entity that will receive the results of the request (used when it is  
5190 distinct from the access-**subject**).

5191 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

5192 This identifier indicates a system entity through which the **access** request was passed.

5193 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

5194 This identifier indicates a system entity associated with a local or remote codebase that generated the  
5195 request. Corresponding **subject attributes** might include the URL from which it was loaded and/or the  
5196 identity of the code-signer.

5197 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

5198 This identifier indicates a system entity associated with the computer that initiated the **access** request.  
5199 An example would be an IPsec identity.

5200 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

### 5201 B.3 Data-types

5202 The following identifiers indicate data-types that are defined in Section A.2.

5203 `urn:oasis:names:tc:xacml:1.0:data-type:x500Name`.

5204 `urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`

5205 `urn:oasis:names:tc:xacml:2.0:data-type:ipAddress`

5206 `urn:oasis:names:tc:xacml:2.0:data-type:dnsName`

5207 `urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression`

- 5208 The following data-type identifiers are defined by XML Schema [XS].
- 5209 <http://www.w3.org/2001/XMLSchema#string>
- 5210 <http://www.w3.org/2001/XMLSchema#boolean>
- 5211 <http://www.w3.org/2001/XMLSchema#integer>
- 5212 <http://www.w3.org/2001/XMLSchema#double>
- 5213 <http://www.w3.org/2001/XMLSchema#time>
- 5214 <http://www.w3.org/2001/XMLSchema#date>
- 5215 <http://www.w3.org/2001/XMLSchema#dateTime>
- 5216 <http://www.w3.org/2001/XMLSchema#anyURI>
- 5217 <http://www.w3.org/2001/XMLSchema#hexBinary>
- 5218 <http://www.w3.org/2001/XMLSchema#base64Binary>
- 5219 The following data-type identifiers correspond to the `dayTimeDuration` and `yearMonthDuration` data-types defined in [XF] Sections 10.3.2 and 10.3.1, respectively.
- 5220
- 5221 <http://www.w3.org/2001/XMLSchema#dayTimeDuration>
- 5222 <http://www.w3.org/2001/XMLSchema#yearMonthDuration>

## 5223 B.4 Subject attributes

- 5224 These identifiers indicate **attributes** of a **subject**. When used, it is RECOMMENDED that they appear within an `<Attributes>` element of the request **context** with a **subject** category (see section B.2).
- 5225
- 5226 At most one of each of these **attributes** is associated with each **subject**. Each **attribute** associated with authentication included within a single `<Attributes>` element relates to the same authentication event.
- 5227
- 5228 This identifier indicates the name of the **subject**.
- 5229 `urn:oasis:names:tc:xacml:1.0:subject:subject-id`
- 5230 This identifier indicates the security domain of the subject. It identifies the administrator and **policy** that manages the name-space in which the **subject** id is administered.
- 5231
- 5232 `urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier`
- 5233 This identifier indicates a public key used to confirm the **subject's** identity.
- 5234 `urn:oasis:names:tc:xacml:1.0:subject:key-info`
- 5235 This identifier indicates the time at which the **subject** was authenticated.
- 5236 `urn:oasis:names:tc:xacml:1.0:subject:authentication-time`
- 5237 This identifier indicates the method used to authenticate the **subject**.
- 5238 `urn:oasis:names:tc:xacml:1.0:subject:authentication-method`
- 5239 This identifier indicates the time at which the **subject** initiated the **access** request, according to the **PEP**.
- 5240 `urn:oasis:names:tc:xacml:1.0:subject:request-time`
- 5241 This identifier indicates the time at which the **subject's** current session began, according to the **PEP**.
- 5242 `urn:oasis:names:tc:xacml:1.0:subject:session-start-time`
- 5243 The following identifiers indicate the location where authentication credentials were activated.
- 5244 This identifier indicates that the location is expressed as an IP address.
- 5245 `urn:oasis:names:tc:xacml:3.0:subject:authn-locality:ip-address`
- 5246 The corresponding **attribute** SHALL be of data-type "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress".
- 5247 This identifier indicates that the location is expressed as a DNS name.
- 5248 `urn:oasis:names:tc:xacml:3.0:subject:authn-locality:dns-name`
- 5249 The corresponding **attribute** SHALL be of data-type "urn:oasis:names:tc:xacml:2.0:data-type:dnsName".

5250 Where a suitable **attribute** is already defined in LDAP [LDAP-1], [LDAP-2], the XACML identifier SHALL  
5251 be formed by adding the **attribute** name to the URI of the LDAP specification. For example, the **attribute**  
5252 name for the userPassword defined in the RFC 2256 SHALL be:  
5253 `http://www.ietf.org/rfc/rfc2256.txt#userPassword`

## 5254 B.5 Resource attributes

5255 These identifiers indicate **attributes** of the **resource**. When used, it is RECOMMENDED they appear  
5256 within the <Attributes> element of the request **context** with Category  
5257 `urn:oasis:names:tc:xacml:3.0:attribute-category:resource`.

5258 This **attribute** identifies the **resource** to which **access** is requested.

5259 `urn:oasis:names:tc:xacml:1.0:resource:resource-id`

5260 This **attribute** identifies the namespace of the top element(s) of the contents of the <Content> element.  
5261 In the case where the **resource** content is supplied in the request **context** and the **resource**  
5262 namespaces are defined in the **resource**, the **PEP** MAY provide this **attribute** in the request to indicate  
5263 the namespaces of the **resource** content. In this case there SHALL be one value of this **attribute** for  
5264 each unique namespace of the top level elements in the <Content> element. The type of the  
5265 corresponding **attribute** SHALL be “`http://www.w3.org/2001/XMLSchema#anyURI`”.

5266 `urn:oasis:names:tc:xacml:2.0:resource:target-namespace`

## 5267 B.6 Action attributes

5268 These identifiers indicate **attributes** of the **action** being requested. When used, it is RECOMMENDED  
5269 they appear within the <Attributes> element of the request **context** with Category  
5270 `urn:oasis:names:tc:xacml:3.0:attribute-category:action`.

5271 This **attribute** identifies the **action** for which **access** is requested.

5272 `urn:oasis:names:tc:xacml:1.0:action:action-id`

5273 Where the **action** is implicit, the value of the action-id **attribute** SHALL be

5274 `urn:oasis:names:tc:xacml:1.0:action:implied-action`

5275 This **attribute** identifies the namespace in which the action-id **attribute** is defined.

5276 `urn:oasis:names:tc:xacml:1.0:action:action-namespace`

## 5277 B.7 Environment attributes

5278 These identifiers indicate **attributes** of the **environment** within which the **decision request** is to be  
5279 evaluated. When used in the **decision request**, it is RECOMMENDED they appear in the  
5280 <Attributes> element of the request **context** with Category `urn:oasis:names:tc:xacml:3.0:attribute-`  
5281 `category:environment`.

5282 This identifier indicates the current time at the **context handler**. In practice it is the time at which the  
5283 request **context** was created. For this reason, if these identifiers appear in multiple places within a  
5284 <Policy> or <PolicySet>, then the same value SHALL be assigned to each occurrence in the  
5285 evaluation procedure, regardless of how much time elapses between the processing of the occurrences.

5286 `urn:oasis:names:tc:xacml:1.0:environment:current-time`

5287 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#time`”.

5288 `urn:oasis:names:tc:xacml:1.0:environment:current-date`

5289 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#date`”.

5290 `urn:oasis:names:tc:xacml:1.0:environment:current-dateTime`

5291 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#dateTime`”.

## 5292 **B.8 Status codes**

5293 The following status code values are defined.

5294 This identifier indicates success.

5295 urn:oasis:names:tc:xacml:1.0:status:ok

5296 This identifier indicates that all the **attributes** necessary to make a **policy decision** were not available  
5297 (see Section 5.58).

5298 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

5299 This identifier indicates that some **attribute** value contained a syntax error, such as a letter in a numeric  
5300 field.

5301 urn:oasis:names:tc:xacml:1.0:status:syntax-error

5302 This identifier indicates that an error occurred during **policy** evaluation. An example would be division by  
5303 zero.

5304 urn:oasis:names:tc:xacml:1.0:status:processing-error

## 5305 **B.9 Combining algorithms**

5306 The deny-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId  
5307 attribute:

5308 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides

5309 The deny-overrides **policy-combining algorithm** has the following value for the  
5310 policyCombiningAlgId attribute:

5311 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides

5312 The permit-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId  
5313 attribute:

5314 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides

5315 The permit-overrides **policy-combining algorithm** has the following value for the  
5316 policyCombiningAlgId attribute:

5317 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides

5318 The first-applicable **rule-combining algorithm** has the following value for the ruleCombiningAlgId  
5319 attribute:

5320 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable

5321 The first-applicable **policy-combining algorithm** has the following value for the  
5322 policyCombiningAlgId attribute:

5323 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable

5324 The only-one-applicable-policy **policy-combining algorithm** has the following value for the  
5325 policyCombiningAlgId attribute:

5326 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

5327 The ordered-deny-overrides **rule-combining algorithm** has the following value for the  
5328 ruleCombiningAlgId attribute:

5329 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides

5330 The ordered-deny-overrides **policy-combining algorithm** has the following value for the  
5331 policyCombiningAlgId attribute:

5332 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-  
5333 overrides

5334 The ordered-permit-overrides **rule-combining algorithm** has the following value for the  
5335 ruleCombiningAlgId attribute:

5336 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-  
5337 overrides

5338 The ordered-permit-overrides **policy-combining algorithm** has the following value for the  
5339 policyCombiningAlgId attribute:

5340 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-  
5341 overrides

5342 The deny-unless-permit **rule-combining algorithm** has the following value for the  
5343 policyCombiningAlgId attribute:

5344 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit

5345 The permit-unless-deny **rule-combining algorithm** has the following value for the  
5346 policyCombiningAlgId attribute:

5347 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny

5348 The deny-unless-permit **policy-combining algorithm** has the following value for the  
5349 policyCombiningAlgId attribute:

5350 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit

5351 The permit-unless-deny **policy-combining algorithm** has the following value for the  
5352 policyCombiningAlgId attribute:

5353 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny

5354 The legacy deny-overrides **rule-combining algorithm** has the following value for the  
5355 ruleCombiningAlgId attribute:

5356 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5357 The legacy deny-overrides **policy-combining algorithm** has the following value for the  
5358 policyCombiningAlgId attribute:

5359 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

5360 The legacy permit-overrides **rule-combining algorithm** has the following value for the  
5361 ruleCombiningAlgId attribute:

5362 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

5363 The legacy permit-overrides **policy-combining algorithm** has the following value for the  
5364 policyCombiningAlgId attribute:

5365 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

5366 The legacy ordered-deny-overrides **rule-combining algorithm** has the following value for the  
5367 ruleCombiningAlgId attribute:

5368 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides

5369 The legacy ordered-deny-overrides **policy-combining algorithm** has the following value for the  
5370 policyCombiningAlgId attribute:

5371 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-  
5372 overrides

5373 The legacy ordered-permit-overrides **rule-combining algorithm** has the following value for the  
5374 ruleCombiningAlgId attribute:

5375 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-  
5376 overrides

5377 The legacy ordered-permit-overrides **policy-combining algorithm** has the following value for the  
5378 policyCombiningAlgId attribute:

5379 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-  
5380 overrides

5381

5382

## Appendix C. Combining algorithms (normative)

5383 This section contains a description of the **rule-** and **policy-combining algorithms** specified by XACML.  
5384 Pseudo code is normative, descriptions in English are non-normative.

5385 The legacy **combining algorithms** are defined in previous versions of XACML, and are retained for  
5386 compatibility reasons. It is RECOMMENDED that the new **combining algorithms** are used instead of the  
5387 legacy **combining algorithms** for new use.

5388 Note that in each case an implementation is conformant as long as it produces the same result as is  
5389 specified here, regardless of how and in what order the implementation behaves internally.

### 5390 C.1 Extended Indeterminate values

5391 Some combining algorithms are defined in terms of an extended set of "Indeterminate" values. See  
5392 section 7.10 for the definition of the Extended Indeterminate values. For these algorithms, the **PDP** MUST  
5393 keep track of the extended set of "Indeterminate" values during **rule** and **policy** combining.

5394 The output of a combining algorithm which does not track the extended set of "Indeterminate" values  
5395 MUST be treated as "Indeterminate{DP}" for the value "Indeterminate" by a combining algorithm which  
5396 tracks the extended set of "Indeterminate" values.

5397 A combining algorithm which does not track the extended set of "Indeterminate" values MUST treat the  
5398 output of a combining algorithm which tracks the extended set of "Indeterminate" values as an  
5399 "Indeterminate" for any of the possible values of the extended set of "Indeterminate".

### 5400 C.2 Deny-overrides

5401 This section defines the "Deny-overrides" **rule-combining algorithm** of a **policy** and **policy-combining**  
5402 **algorithm** of a **policy set**.

5403 This **combining algorithm** makes use of the extended "Indeterminate".

5404 The **rule combining algorithm** defined here has the following identifier:

5405 `urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides`

5406 The **policy combining algorithm** defined here has the following identifier:

5407 `urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides`

5408 The following is a non-normative informative description of this **combining algorithm**.

5409 The deny overrides **combining algorithm** is intended for those cases where a deny  
5410 decision should have priority over a permit decision. This algorithm has the following  
5411 behavior.

- 5412 1. If any decision is "Deny", the result is "Deny".
- 5413 2. Otherwise, if any decision is "Indeterminate{DP}", the result is "Indeterminate{DP}".
- 5414 3. Otherwise, if any decision is "Indeterminate{D}" and another decision is "Indeterminate{P}" or  
5415 Permit, the result is "Indeterminate{DP}".
- 5416 4. Otherwise, if any decision is "Indeterminate{D}", the result is "Indeterminate{D}".
- 5417 5. Otherwise, if any decision is "Permit", the result is "Permit".
- 5418 6. Otherwise, if any decision is "Indeterminate{P}", the result is "Indeterminate{P}".
- 5419 7. Otherwise, the result is "NotApplicable".

5420 The following pseudo-code represents the normative specification of this **combining algorithm**. The  
5421 algorithm is presented here in a form where the input to it is an array with children (the **policies**, **policy**  
5422 **sets** or **rules**) of the **policy** or **policy set**. The children may be processed in any order, so the set of  
5423 obligations or advice provided by this algorithm is not deterministic.

```

5424 Decision denyOverridesCombiningAlgorithm(Node[] children)
5425 {
5426     Boolean atLeastOneErrorD = false;
5427     Boolean atLeastOneErrorP = false;
5428     Boolean atLeastOneErrorDP = false;
5429     Boolean atLeastOnePermit = false;
5430     for( i=0 ; i < lengthOf(children) ; i++ )
5431     {
5432         Decision decision = children[i].evaluate();
5433         if (decision == Deny)
5434         {
5435             return Deny;
5436         }
5437         if (decision == Permit)
5438         {
5439             atLeastOnePermit = true;
5440             continue;
5441         }
5442         if (decision == NotApplicable)
5443         {
5444             continue;
5445         }
5446         if (decision == Indeterminate{D})
5447         {
5448             atLeastOneErrorD = true;
5449             continue;
5450         }
5451         if (decision == Indeterminate{P})
5452         {
5453             atLeastOneErrorP = true;
5454             continue;
5455         }
5456         if (decision == Indeterminate{DP})
5457         {
5458             atLeastOneErrorDP = true;
5459             continue;
5460         }
5461     }
5462     if (atLeastOneErrorDP)
5463     {
5464         return Indeterminate{DP};
5465     }
5466     if (atLeastOneErrorD && (atLeastOneErrorP || atLeastOnePermit))
5467     {
5468         return Indeterminate{DP};
5469     }
5470     if (atLeastOneErrorD)
5471     {
5472         return Indeterminate{D};
5473     }
5474     if (atLeastOnePermit)
5475     {
5476         return Permit;
5477     }
5478     if (atLeastOneErrorP)
5479     {
5480         return Indeterminate{P};
5481     }
5482     return NotApplicable;
5483 }

```

5484 **Obligations** and **advice** shall be combined as described in Section 7.18.

### 5485 C.3 Ordered-deny-overrides

5486 The following specification defines the "Ordered-deny-overrides" **rule-combining algorithm** of a **policy**.

5487 The behavior of this algorithm is identical to that of the "Deny-overrides" **rule-combining**  
5488 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5489 match the order as listed in the **policy**.

5490 The **rule combining algorithm** defined here has the following identifier:

5491 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides

5492 The following specification defines the "Ordered-deny-overrides" **policy-combining algorithm** of a  
5493 **policy set**.

5494 The behavior of this algorithm is identical to that of the "Deny-overrides" **policy-combining**  
5495 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
5496 match the order as listed in the **policy set**.

5497 The **policy combining algorithm** defined here has the following identifier:

5498 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-  
5499 overrides

### 5500 C.4 Permit-overrides

5501 This section defines the "Permit-overrides" **rule-combining algorithm** of a **policy** and **policy-combining**  
5502 **algorithm** of a **policy set**.

5503 This **combining algorithm** makes use of the extended "Indeterminate".

5504 The **rule combining algorithm** defined here has the following identifier:

5505 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides

5506 The **policy combining algorithm** defined here has the following identifier:

5507 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides

5508 The following is a non-normative informative description of this combining algorithm.

5509 The permit overrides **combining algorithm** is intended for those cases where a permit  
5510 decision should have priority over a deny decision. This algorithm has the following  
5511 behavior.

- 5512 1. If any decision is "Permit", the result is "Permit".
- 5513 2. Otherwise, if any decision is "Indeterminate{DP}", the result is "Indeterminate{DP}".
- 5514 3. Otherwise, if any decision is "Indeterminate{P}" and another decision is  
5515 "Indeterminate{D} or Deny, the result is "Indeterminate{DP}".
- 5516 4. Otherwise, if any decision is "Indeterminate{P}", the result is "Indeterminate{P}".
- 5517 5. Otherwise, if any decision is "Deny", the result is "Deny".
- 5518 6. Otherwise, if any decision is "Indeterminate{D}", the result is "Indeterminate{D}".
- 5519 7. Otherwise, the result is "NotApplicable".

5520 The following pseudo-code represents the normative specification of this **combining algorithm**. The  
5521 algorithm is presented here in a form where the input to it is an array with all children (the **policies**, **policy**  
5522 **sets** or **rules**) of the **policy** or **policy set**. The children may be processed in any order, so the set of  
5523 obligations or advice provided by this algorithm is not deterministic.

```
5524 Decision permitOverridesCombiningAlgorithm(Node[] children)  
5525 {  
5526     Boolean atLeastOneErrorD = false;  
5527     Boolean atLeastOneErrorP = false;  
5528     Boolean atLeastOneErrorDP = false;  
5529     Boolean atLeastOneDeny = false;
```

```

5530     for( i=0 ; i < lengthOf(children) ; i++ )
5531     {
5532         Decision decision = children[i].evaluate();
5533         if (decision == Deny)
5534         {
5535             atLeastOneDeny = true;
5536             continue;
5537         }
5538         if (decision == Permit)
5539         {
5540             return Permit;
5541         }
5542         if (decision == NotApplicable)
5543         {
5544             continue;
5545         }
5546         if (decision == Indeterminate{D})
5547         {
5548             atLeastOneErrorD = true;
5549             continue;
5550         }
5551         if (decision == Indeterminate{P})
5552         {
5553             atLeastOneErrorP = true;
5554             continue;
5555         }
5556         if (decision == Indeterminate{DP})
5557         {
5558             atLeastOneErrorDP = true;
5559             continue;
5560         }
5561     }
5562     if (atLeastOneErrorDP)
5563     {
5564         return Indeterminate{DP};
5565     }
5566     if (atLeastOneErrorP && (atLeastOneErrorD || atLeastOneDeny))
5567     {
5568         return Indeterminate{DP};
5569     }
5570     if (atLeastOneErrorP)
5571     {
5572         return Indeterminate{P};
5573     }
5574     if (atLeastOneDeny)
5575     {
5576         return Deny;
5577     }
5578     if (atLeastOneErrorD)
5579     {
5580         return Indeterminate{D};
5581     }
5582     return NotApplicable;
5583 }

```

5584 **Obligations** and **advice** shall be combined as described in Section 7.18.

## 5585 C.5 Ordered-permit-overrides

5586 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a **policy**.

5587 The behavior of this algorithm is identical to that of the "Permit-overrides" **rule-combining**  
5588 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5589 match the order as listed in the **policy**.

5590 The **rule combining algorithm** defined here has the following identifier:  
5591 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-  
5592 overrides  
5593 The following specification defines the "Ordered-permit-overrides" **policy-combining algorithm** of a  
5594 **policy set**.  
5595 The behavior of this algorithm is identical to that of the "Permit-overrides" **policy-combining**  
5596 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
5597 match the order as listed in the **policy set**.  
5598 The **policy combining algorithm** defined here has the following identifier:  
5599 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-  
5600 overrides

## 5601 C.6 Deny-unless-permit

5602 This section defines the "Deny-unless-permit" **rule-combining algorithm** of a **policy** or **policy-**  
5603 **combining algorithm** of a **policy set**.

5604 The **rule combining algorithm** defined here has the following identifier:  
5605 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit

5606 The **policy combining algorithm** defined here has the following identifier:  
5607 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit

5608 The following is a non-normative informative description of this **combining algorithm**.

5609 The "Deny-unless-permit" **combining algorithm** is intended for those cases where a  
5610 permit decision should have priority over a deny decision, and an "Indeterminate" or  
5611 "NotApplicable" must never be the result. It is particularly useful at the top level in a  
5612 **policy** structure to ensure that a **PDP** will always return a definite "Permit" or "Deny"  
5613 result. This algorithm has the following behavior.

- 5614 1. If any decision is "Permit", the result is "Permit".
- 5615 2. Otherwise, the result is "Deny".

5616 The following pseudo-code represents the normative specification of this **combining algorithm**. The  
5617 algorithm is presented here in a form where the input to it is an array with all the children (the **policies**,  
5618 **policy sets** or **rules**) of the **policy** or **policy set**. The children may be processed in any order, so the set  
5619 of obligations or advice provided by this algorithm is not deterministic.

```
5620 Decision denyUnlessPermitCombiningAlgorithm(Node[] children)  
5621 {  
5622   for( i=0 ; i < lengthOf(children) ; i++ )  
5623   {  
5624     if (children[i].evaluate() == Permit)  
5625     {  
5626       return Permit;  
5627     }  
5628   }  
5629   return Deny;  
5630 }
```

5631 **Obligations** and **advice** shall be combined as described in Section 7.18.

## 5632 C.7 Permit-unless-deny

5633 This section defines the "Permit-unless-deny" **rule-combining algorithm** of a **policy** or **policy-**  
5634 **combining algorithm** of a **policy set**.

5635 The **rule combining algorithm** defined here has the following identifier:

5636 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny

5637 The **policy combining algorithm** defined here has the following identifier:  
5638 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny  
5639 The following is a non-normative informative description of this **combining algorithm**.

5640 The "Permit-unless-deny" **combining algorithm** is intended for those cases where a  
5641 deny decision should have priority over a permit decision, and an "Indeterminate" or  
5642 "NotApplicable" must never be the result. It is particularly useful at the top level in a  
5643 **policy** structure to ensure that a **PDP** will always return a definite "Permit" or "Deny"  
5644 result. This algorithm has the following behavior.

- 5645 1. If any decision is "Deny", the result is "Deny".
- 5646 2. Otherwise, the result is "Permit".

5647 The following pseudo-code represents the normative specification of this **combining algorithm**. The  
5648 algorithm is presented here in a form where the input to it is an array with all the children (the **policies**,  
5649 **policy sets** or **rules**) of the **policy** or **policy set**. The children may be processed in any order, so the set  
5650 of obligations or advice provided by this algorithm is not deterministic.

```
5651 Decision permitUnlessDenyCombiningAlgorithm(Node[] children)
5652 {
5653   for( i=0 ; i < lengthOf(children) ; i++ )
5654   {
5655     if (children[i].evaluate() == Deny)
5656     {
5657       return Deny;
5658     }
5659   }
5660   return Permit;
5661 }
```

5662 **Obligations** and **advice** shall be combined as described in Section 7.18.

## 5663 C.8 First-applicable

5664 This section defines the "First-applicable" **rule-combining algorithm** of a **policy** and **policy-combining**  
5665 **algorithm** of a **policy set**.

5666 The **rule combining algorithm** defined here has the following identifier:

5667 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable

5668 The following is a non-normative informative description of the "First-Applicable" **rule-combining**  
5669 **algorithm** of a **policy**.

5670 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a particular  
5671 **rule**, if the **target** matches and the **condition** evaluates to "True", then the evaluation of the  
5672 **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the result of the evaluation  
5673 of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected in the evaluation, if the  
5674 **target** evaluates to "False" or the **condition** evaluates to "False", then the next **rule** in the order  
5675 SHALL be evaluated. If no further **rule** in the order exists, then the **policy** SHALL evaluate to  
5676 "NotApplicable".

5677 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation SHALL  
5678 halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error status.

5679 The following pseudo-code represents the normative specification of this **rule-combining algorithm**.

```
5680 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule[] rules)
5681 {
5682   for( i = 0 ; i < lengthOf(rules) ; i++ )
5683   {
5684     Decision decision = evaluate(rules[i]);
5685     if (decision == Deny)
5686     {
```

```

5687         return Deny;
5688     }
5689     if (decision == Permit)
5690     {
5691         return Permit;
5692     }
5693     if (decision == NotApplicable)
5694     {
5695         continue;
5696     }
5697     if (decision == Indeterminate)
5698     {
5699         return Indeterminate;
5700     }
5701 }
5702 return NotApplicable;
5703 }

```

5704 The **policy combining algorithm** defined here has the following identifier:

5705 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable

5706 The following is a non-normative informative description of the "First-applicable" **policy-combining**  
5707 **algorithm of a policy set**.

5708 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular **policy**, if  
5709 the **target** evaluates to "True" and the **policy** evaluates to a determinate value of "Permit" or  
5710 "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to the **effect** value of  
5711 that **policy**. For a particular **policy**, if the **target** evaluate to "False", or the **policy** evaluates to  
5712 "NotApplicable", then the next **policy** in the order SHALL be evaluated. If no further **policy** exists  
5713 in the order, then the **policy set** SHALL evaluate to "NotApplicable".

5714 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**, the  
5715 reference to the **policy** is considered invalid, or the **policy** itself evaluates to "Indeterminate",  
5716 then the evaluation of the **policy-combining algorithm** shall halt, and the **policy set** shall  
5717 evaluate to "Indeterminate" with an appropriate error status.

5718 The following pseudo-code represents the normative specification of this policy-combination algorithm.

```

5719 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy[] policies)
5720 {
5721     for( i = 0 ; i < lengthOf(policies) ; i++ )
5722     {
5723         Decision decision = evaluate(policies[i]);
5724         if(decision == Deny)
5725         {
5726             return Deny;
5727         }
5728         if(decision == Permit)
5729         {
5730             return Permit;
5731         }
5732         if (decision == NotApplicable)
5733         {
5734             continue;
5735         }
5736         if (decision == Indeterminate)
5737         {
5738             return Indeterminate;
5739         }
5740     }
5741     return NotApplicable;
5742 }

```

5743 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.18.

## 5744 C.9 Only-one-applicable

5745 This section defines the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

5746 The **policy combining algorithm** defined here has the following identifier:

5747 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

5748 The following is a non-normative informative description of the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

5750 In the entire set of **policies** in the **policy set**, if no **policy** is considered applicable by virtue of its  
5751 **target**, then the result of the policy-combination algorithm SHALL be "NotApplicable". If more  
5752 than one **policy** is considered applicable by virtue of its **target**, then the result of the policy-  
5753 combination algorithm SHALL be "Indeterminate".

5754 If only one **policy** is considered applicable by evaluation of its **target**, then the result of the  
5755 **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

5756 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered  
5757 invalid or the **policy** evaluation results in "Indeterminate, then the **policy set** SHALL evaluate to  
5758 "Indeterminate", with the appropriate error status.

5759 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```
5760 Decision onlyOneApplicablePolicyPolicyCombiningAlogrithm(Policy[] policies)
5761 {
5762     Boolean          atLeastOne      = false;
5763     Policy           selectedPolicy = null;
5764     ApplicableResult appResult;
5765
5766     for ( i = 0; i < lengthOf(policies) ; i++ )
5767     {
5768         appResult = isApplicable(policies[I]);
5769
5770         if ( appResult == Indeterminate )
5771         {
5772             return Indeterminate;
5773         }
5774         if( appResult == Applicable )
5775         {
5776             if ( atLeastOne )
5777             {
5778                 return Indeterminate;
5779             }
5780             else
5781             {
5782                 atLeastOne      = true;
5783                 selectedPolicy = policies[i];
5784             }
5785         }
5786         if ( appResult == NotApplicable )
5787         {
5788             continue;
5789         }
5790     }
5791     if ( atLeastOne )
5792     {
5793         return evaluate(selectedPolicy);
5794     }
5795     else
5796     {
5797         return NotApplicable;
5798     }
5799 }
```

5800 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.18.

## 5801 C.10 Legacy Deny-overrides

5802 This section defines the legacy “Deny-overrides” *rule-combining algorithm* of a *policy* and *policy-*  
5803 *combining algorithm* of a *policy set*.

5804

5805 The *rule combining algorithm* defined here has the following identifier:

5806 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5807 The following is a non-normative informative description of this combining algorithm.

5808 The “Deny-overrides” rule combining algorithm is intended for those cases where a deny  
5809 decision should have priority over a permit decision. This algorithm has the following  
5810 behavior.

- 5811 1. If any rule evaluates to "Deny", the result is "Deny".
- 5812 2. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is  
5813 "Indeterminate".
- 5814 3. Otherwise, if any rule evaluates to "Permit", the result is "Permit".
- 5815 4. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate", the result is  
5816 "Indeterminate".
- 5817 5. Otherwise, the result is "NotApplicable".

5818 The following pseudo-code represents the normative specification of this *rule-combining algorithm*.

```
5819 Decision denyOverridesRuleCombiningAlgorithm(Rule[] rules)
5820 {
5821     Boolean atLeastOneError = false;
5822     Boolean potentialDeny = false;
5823     Boolean atLeastOnePermit = false;
5824     for( i=0 ; i < lengthOf(rules) ; i++ )
5825     {
5826         Decision decision = evaluate(rules[i]);
5827         if (decision == Deny)
5828         {
5829             return Deny;
5830         }
5831         if (decision == Permit)
5832         {
5833             atLeastOnePermit = true;
5834             continue;
5835         }
5836         if (decision == NotApplicable)
5837         {
5838             continue;
5839         }
5840         if (decision == Indeterminate)
5841         {
5842             atLeastOneError = true;
5843
5844             if (effect(rules[i]) == Deny)
5845             {
5846                 potentialDeny = true;
5847             }
5848             continue;
5849         }
5850     }
5851     if (potentialDeny)
5852     {
5853         return Indeterminate;
5854     }
5855     if (atLeastOnePermit)
5856     {
```

```

5857     return Permit;
5858   }
5859   if (atLeastOneError)
5860   {
5861     return Indeterminate;
5862   }
5863   return NotApplicable;
5864 }

```

5865 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.18.

5866 The **policy combining algorithm** defined here has the following identifier:

5867 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

5868 The following is a non-normative informative description of this combining algorithm.

5869 The "Deny-overrides" policy combining algorithm is intended for those cases where a  
5870 deny decision should have priority over a permit decision. This algorithm has the  
5871 following behavior.

- 5872 1. If any policy evaluates to "Deny", the result is "Deny".
- 5873 2. Otherwise, if any policy evaluates to "Indeterminate", the result is "Deny".
- 5874 3. Otherwise, if any policy evaluates to "Permit", the result is "Permit".
- 5875 4. Otherwise, the result is "NotApplicable".

5876 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```

5877 Decision denyOverridesPolicyCombiningAlgorithm(Policy[] policies)
5878 {
5879   Boolean atLeastOnePermit = false;
5880   for( i=0 ; i < lengthOf(policies) ; i++ )
5881   {
5882     Decision decision = evaluate(policies[i]);
5883     if (decision == Deny)
5884     {
5885       return Deny;
5886     }
5887     if (decision == Permit)
5888     {
5889       atLeastOnePermit = true;
5890       continue;
5891     }
5892     if (decision == NotApplicable)
5893     {
5894       continue;
5895     }
5896     if (decision == Indeterminate)
5897     {
5898       return Deny;
5899     }
5900   }
5901   if (atLeastOnePermit)
5902   {
5903     return Permit;
5904   }
5905   return NotApplicable;
5906 }

```

5907 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.18.

## 5908 C.11 Legacy Ordered-deny-overrides

5909 The following specification defines the legacy "Ordered-deny-overrides" **rule-combining algorithm** of a  
5910 **policy**.

5911 The behavior of this algorithm is identical to that of the “Deny-overrides” **rule-combining**  
5912 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5913 match the order as listed in the **policy**.

5914 The **rule combining algorithm** defined here has the following identifier:

5915 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides

5916 The following specification defines the legacy “Ordered-deny-overrides” **policy-combining algorithm** of  
5917 a **policy set**.

5918 The behavior of this algorithm is identical to that of the “Deny-overrides” **policy-combining**  
5919 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
5920 match the order as listed in the **policy set**.

5921 The **rule combining algorithm** defined here has the following identifier:

5922 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-  
5923 overrides

## 5924 C.12 Legacy Permit-overrides

5925 This section defines the legacy “Permit-overrides” **rule-combining algorithm** of a **policy** and **policy-**  
5926 **combining algorithm** of a **policy set**.

5927 The **rule combining algorithm** defined here has the following identifier:

5928 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

5929 The following is a non-normative informative description of this combining algorithm.

5930 The “Permit-overrides” rule combining algorithm is intended for those cases where a  
5931 permit decision should have priority over a deny decision. This algorithm has the  
5932 following behavior.

- 5933 1. If any rule evaluates to "Permit", the result is "Permit".
- 5934 2. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate" the result is  
5935 "Indeterminate".
- 5936 3. Otherwise, if any rule evaluates to "Deny", the result is "Deny".
- 5937 4. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is  
5938 "Indeterminate".
- 5939 5. Otherwise, the result is "NotApplicable".

5940 The following pseudo-code represents the normative specification of this **rule-combining algorithm**.

```
5941 Decision permitOverridesRuleCombiningAlgorithm(Rule[] rules)
5942 {
5943     Boolean atLeastOneError = false;
5944     Boolean potentialPermit = false;
5945     Boolean atLeastOneDeny = false;
5946     for( i=0 ; i < lengthOf(rules) ; i++ )
5947     {
5948         Decision decision = evaluate(rules[i]);
5949         if (decision == Deny)
5950         {
5951             atLeastOneDeny = true;
5952             continue;
5953         }
5954         if (decision == Permit)
5955         {
5956             return Permit;
5957         }
5958         if (decision == NotApplicable)
5959         {
5960             continue;
5961         }
5962     }
5963 }
```

```

5962         if (decision == Indeterminate)
5963         {
5964             atLeastOneError = true;
5965
5966             if (effect(rules[i]) == Permit)
5967             {
5968                 potentialPermit = true;
5969             }
5970             continue;
5971         }
5972     }
5973     if (potentialPermit)
5974     {
5975         return Indeterminate;
5976     }
5977     if (atLeastOneDeny)
5978     {
5979         return Deny;
5980     }
5981     if (atLeastOneError)
5982     {
5983         return Indeterminate;
5984     }
5985     return NotApplicable;
5986 }

```

5987 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.18.

5988 The **policy combining algorithm** defined here has the following identifier:

5989 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

5990 The following is a non-normative informative description of this combining algorithm.

5991 The "Permit-overrides" policy combining algorithm is intended for those cases where a  
5992 permit decision should have priority over a deny decision. This algorithm has the  
5993 following behavior.

- 5994 1. If any policy evaluates to "Permit", the result is "Permit".
- 5995 2. Otherwise, if any policy evaluates to "Deny", the result is "Deny".
- 5996 3. Otherwise, if any policy evaluates to "Indeterminate", the result is "Indeterminate".
- 5997 4. Otherwise, the result is "NotApplicable".

5998 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```

5999 Decision permitOverridesPolicyCombiningAlgorithm(Policy[] policies)
6000 {
6001     Boolean atLeastOneError = false;
6002     Boolean atLeastOneDeny = false;
6003     for( i=0 ; i < lengthOf(policies) ; i++ )
6004     {
6005         Decision decision = evaluate(policies[i]);
6006         if (decision == Deny)
6007         {
6008             atLeastOneDeny = true;
6009             continue;
6010         }
6011         if (decision == Permit)
6012         {
6013             return Permit;
6014         }
6015         if (decision == NotApplicable)
6016         {
6017             continue;
6018         }
6019         if (decision == Indeterminate)

```

```

6020     {
6021         atLeastOneError = true;
6022         continue;
6023     }
6024 }
6025 if (atLeastOneDeny)
6026 {
6027     return Deny;
6028 }
6029 if (atLeastOneError)
6030 {
6031     return Indeterminate;
6032 }
6033 return NotApplicable;
6034 }

```

6035 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.18.

### 6036 C.13 Legacy Ordered-permit-overrides

6037 The following specification defines the legacy "Ordered-permit-overrides" **rule-combining algorithm** of a  
6038 **policy**.

6039 The behavior of this algorithm is identical to that of the "Permit-overrides" **rule-combining**  
6040 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
6041 match the order as listed in the **policy**.

6042 The **rule combining algorithm** defined here has the following identifier:

6043 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-  
6044 overrides

6045 The following specification defines the legacy "Ordered-permit-overrides" **policy-combining algorithm** of  
6046 a **policy set**.

6047 The behavior of this algorithm is identical to that of the "Permit-overrides" **policy-combining**  
6048 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
6049 match the order as listed in the **policy set**.

6050 The **policy combining algorithm** defined here has the following identifier:

6051 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-  
6052 overrides

6053

---

## 6054 **Appendix D. Acknowledgements**

6055 The following individuals have participated in the creation of this specification and are gratefully  
6056 acknowledged:

6057  
6058 Anil Saldhana  
6059 Anil Tappetla  
6060 Anne Anderson  
6061 Anthony Nadalin  
6062 Bill Parducci  
6063 Craig Forster  
6064 David Chadwick  
6065 David Staggs  
6066 Dilli Arumugam  
6067 Duane DeCouteau  
6068 Erik Rissanen  
6069 Gareth Richards  
6070 Hal Lockhart  
6071 Jan Herrmann  
6072 John Tolbert  
6073 Ludwig Seitz  
6074 Michiharu Kudo  
6075 Naomaru Itoi  
6076 Paul Tyson  
6077 Prateek Mishra  
6078 Rich Levinson  
6079 Ronald Jacobson  
6080 Seth Proctor  
6081 Sridhar Muppidi  
6082 Tim Moses  
6083 Vernon Murdoch

6084

## Appendix E. Revision History

6085

6086

Revision	Date	Editor	Changes Made
WD 05	10 Oct 2007	Erik Rissanen	Convert to new OASIS template. Fixed typos and errors.
WD 06	18 May 2008	Erik Rissanen	<p>Added missing MaxDelegationDepth in schema fragments.</p> <p>Added missing urn:oasis:names:tc:xacml:1.0:resource:xpath identifier.</p> <p>Corrected typos on xpaths in the example policies.</p> <p>Removed use of xpointer in the examples.</p> <p>Made the &lt;Content&gt; element the context node of all xpath expressions and introduced categorization of XPath expressions so they point to a specific &lt;Content&gt; element.</p> <p>Added &lt;Content&gt; element to the policy issuer.</p> <p>Added description of the &lt;PolicyIssuer&gt; element.</p> <p>Updated the schema figure in the introduction to reflect the new AllOf/AnyOf schema.</p> <p>Remove duplicate &lt;CombinerParameters&gt; element in the &lt;Policy&gt; element in the schema.</p> <p>Removed default attributes in the schema. (Version in &lt;Policy(Set)&gt; and MustBePresent in &lt;AttributeDesignator&gt; in &lt;AttributeSelector&gt;)</p> <p>Removed references in section 7.3 to the &lt;Condition&gt; element having a FunctionId attribute.</p> <p>Fixed typos in data type URIs in section A.3.7.</p>
WD 07	3 Nov 2008	Erik Rissanen	<p>Fixed "...:data-types:..." typo in conformace section.</p> <p>Removed XML default attribute for IncludeInResult for element &lt;Attribute&gt;. Also added this attribute in the associated schema file.</p> <p>Removed description of non-existing XML attribute "ResourceId" from the element &lt;Result&gt;.</p> <p>Moved the urn:oasis:names:tc:xacml:3.0:function:access-permitted function into here from the delegation profile.</p>

			<p>Updated the daytime and yearmonth duration data types to the W3C defined identifiers.</p> <p>Added &lt;Description&gt; to &lt;Apply&gt;.</p> <p>Added XPath versioning to the request.</p> <p>Added security considerations about denial service and the access-permitted function.</p> <p>Changed &lt;Target&gt; matching so NoMatch has priority over Indeterminate.</p> <p>Fixed multiple typos in identifiers.</p> <p>Lower case incorrect use of "MAY".</p> <p>Misc minor typos.</p> <p>Removed whitespace in example attributes.</p> <p>Removed an incorrect sentence about higher order functions in the definition of the &lt;Function&gt; element.</p> <p>Clarified evaluation of empty or missing targets.</p> <p>Use Unicode codepoint collation for string comparisons.</p> <p>Support multiple arguments in multiply functions.</p> <p>Define Indeterminate result for overflow in integer to double conversion.</p> <p>Simplified descriptions of deny/permit overrides algorithms.</p> <p>Add ipAddress and dnsName into conformance section.</p> <p>Don't refer to IEEE 754 for integer arithmetic.</p> <p>Rephrase indeterminate result for arithmetic functions.</p> <p>Fix typos in examples.</p> <p>Clarify Match evaluation and drop list of example functions which can be used in a Match.</p> <p>Added behavior for circular policy/variable references.</p> <p>Fix obligation enforcement so it refers to PEP bias.</p> <p>Added Version xml attribute to the example policies.</p> <p>Remove requirement for PDP to check the target-namespace resource attribute.</p> <p>Added policy identifier list to the response/request.</p> <p>Added statements about Unicode normalization.</p> <p>Clarified definitions of string functions.</p>
--	--	--	---

			<p>Added new string functions.</p> <p>Added section on Unicode security issues.</p>
WD 08	5 Feb 2009	Erik Rissanen	<p>Updated Unicode normalization section according to suggestion from W3C working group.</p> <p>Set union functions now may take more than two arguments.</p> <p>Made obligation parameters into runtime expressions.</p> <p>Added new combining algorithms</p> <p>Added security consideration about policy id collisions.</p> <p>Added the &lt;Advice&gt; feature</p> <p>Made obligations mandatory (per the 19<sup>th</sup> Dec 2008 decision of the TC)</p> <p>Made obligations/advice available in rules</p> <p>Changed wording about deprecation</p>
WD 09			<p>Clarified wording about normative/informative in the combining algorithms section.</p> <p>Fixed duplicate variable in comb.algs and cleaned up variable names.</p> <p>Updated the schema to support the new multiple request scheme.</p>
WD 10	19 Mar 2009	Erik Rissanen	<p>Fixed schema for &lt;Request&gt;</p> <p>Fixed typos.</p> <p>Added optional Category to AttributeAssignments in obligations/advice.</p>
WD 11		Erik Rissanen	<p>Cleanups courtesy of John Tolbert.</p> <p>Added Issuer XML attribute to &lt;AttributeAssignment&gt;</p> <p>Fix the XPath expressions in the example policies and requests</p> <p>Fix inconsistencies in the conformance tables.</p> <p>Editorial cleanups.</p>
WD 12	16 Nov 2009	Erik Rissanen	<p>(Now working draft after public review of CD 1)</p> <p>Fix typos</p> <p>Allow element selection in attribute selector.</p> <p>Improve consistency in the use of the terms obligation, advice, and advice/obligation expressions and where they can appear.</p> <p>Fixed inconsistency in PEP bias between sections 5.1 and 7.2.</p> <p>Clarified text in overview of combining algorithms.</p> <p>Relaxed restriction on matching in xpath-node-</p>

			<p>match function.</p> <p>Remove note about XPath expert review.</p> <p>Removed obsolete resource:xpath identifier.</p> <p>Updated reference to XML spec.</p> <p>Defined error behavior for string-substring and uri-substring functions.</p> <p>Reversed the order of the arguments for the following functions: string-starts-with, uri-starts-with, string-ends-with, uri-ends-with, string-contains and uri-contains</p> <p>Renamed functions:</p> <ul style="list-style-type: none"> <li>• uri-starts-with to anyURI-starts-with</li> <li>• uri-ends-with to anyURI-ends-with</li> <li>• uri-contains to anyURI-contains</li> <li>• uri-substring to anyURI-substring</li> </ul> <p>Removed redundant occurrence indicators from RequestType.</p> <p>Don't use "...:os" namespace in examples since this is still just "...wd-12".</p> <p>Added missing MustBePresent and Version XML attributes in example policies.</p> <p>Added missing ReturnPolicyIdList and IncludeInResult XML attributes in example requests.</p> <p>Clarified error behavior in obligation/advice expressions.</p> <p>Allow bags in attribute assignment expressions.</p> <p>Use the new daytimeduration and yearmonthduration identifiers consistently.</p>
WD 13	14 Dec 2009	Erik Rissanen	<p>Fix small inconsistency in number of arguments to the multiply function.</p> <p>Generalize higher order bag functions.</p> <p>Add ContextSelectorId to attribute selector.</p> <p>Use &lt;Policy(Set)IdReference&gt; in &lt;PolicyIdList&gt;.</p> <p>Fix typos and formatting issues.</p> <p>Make the conformance section clearly reference the functional requirements in the spec.</p> <p>Conformance tests are no longer hosted by Sun.</p>
WD 14	17 Dec 2009	Erik Rissanen	Update acknowledgments
WD 15		Erik Rissanen	<p>Replace DecisionCombiningAlgorithm with a simple Boolean for CombinedDecision.</p> <p>Restrict &lt;Content&gt; to a single child element</p>

			and update the <AttributeSelector> and XPathExpression data type accordingly.
WD 16	12 Jan 2010	Erik Rissanen	Updated cross references Fix typos and minor inconsistencies. Simplify schema of <PolicyIdentifierList> Refactor some of the text to make it easier to understand. Update acknowledgments
WD 17	8 Mar 2010	Erik Rissanen	Updated cross references. Fixed OASIS style issues.
WD 18	23 Jun 2010	Erik Rissanen	Fixed typos in examples. Fixed typos in schema fragments.
WD 19	14 April 2011	Erik Rissanen	Updated function identifiers for new duration functions. Listed old identifiers as planned for deprecation. Added example for the X500Name-match function. Removed the (broken) Haskell definitions of the higher order functions. Clarified behavior of extended indeterminate in context of legacy combining algorithms or an Indeterminate target. Removed <Condition> from the expression substitution group. Specified argument order for subtract, divide and mod functions. Specified datatype to string conversion form to those functions which depend on it. Specified Indeterminate value for functions which convert strings to another datatype if the string is not a valid lexicographical representation of the datatype. Removed higher order functions for ip address and dns name.
WD 20	24 May 2011	Erik Rissanen	Fixed typo between “first” and “second” arguments in rfc822Name-match function. Removed duplicate word “string” in a couple of places. Improved and reorganized the text about extended indeterminate processing and Rule/Policy/PolicySet evaluation. Explicitly stated that an implementation is conformant regardless of its internal workings as long as the external result is the same as in this specification. Changed requirement on Indeterminate behavior at the top of section A.3 which

			conflicted with Boolean function definitions.
WD 21	28 Jun 2011	Erik Rissanen	<p>Redefined combining algorithms so they explicitly evaluate their children in the pseudocode.</p> <p>Changed wording in 7.12 and 7.13 to clarify that the combining algorithm applies to the children only, not the target.</p> <p>Removed wording in attribute category definitions about the attribute categories appearing multiple times since bags of bags are not supported,</p> <p>Fixed many small typos.</p> <p>Clarified wording about combiner parameters.</p>
WD 22	28 Jun 2011	Erik Rissanen	Fix typos in combining algorithm pseudo code.
WD 23	19 Mar 2012	Erik Rissanen	<p>Reformat references to OASIS specs.</p> <p>Define how XACML identifiers are matched.</p> <p>Do not highlight “actions” with the glossary term meaning in section 2.12.</p> <p>Fix minor typos.</p> <p>Make a reference to the full list of combining algorithms from the introduction.</p> <p>Clarified behavior of the context handler.</p> <p>Renamed higher order functions which were generalized in an earlier working draft.</p> <p>Add missing line in schema fragment for &lt;AttributeDesignator&gt;</p> <p>Removed reference to reuse of rules in section 2.2. There is no mechanism in XACML itself to re-use rules, though of course a tool could create copies as a form of “re-use”.</p>

6087