



eXtensible Access Control Markup Language (XACML) Version 3.0

Committee Specification 01

10 August 2010

Specification URIs:

This Version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.html>
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.doc> (Authoritative)
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf>

Previous Version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-04-en.html>
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-04-en.doc> (Authoritative)
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-04-en.pdf>

Latest Version:

<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.doc> (Authoritative)
<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>
Hal Lockhart, Oracle <hal.lockhart@oracle.com>

Editor:

Erik Rissanen, Axiomatics AB <erik@axiomatics.com>

Related work:

This specification replaces or supercedes:

- [eXtensible Access Control Markup Language \(XACML\) Version 2.0](#)

Declared XML Namespace(s):

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 eXtensible Access Control Markup Language (XACML) TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved 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>.

The non-normative errata page for this specification is located at <http://www.oasis-open.org/committees/xacml/>.

Notices

Copyright © OASIS® 2010. 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 names "OASIS" and "XACML" are trademarks 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>	58
5.31 Element <AttributeValue>	59
5.32 Element <Obligations>	60
5.33 Element <AssociatedAdvice>	60
5.34 Element <Obligation>	60
5.35 Element <Advice>	61
5.36 Element <AttributeAssignment>	61
5.37 Element <ObligationExpressions>	62
5.38 Element <AdviceExpressions>	62
5.39 Element <ObligationExpression>	62
5.40 Element <AdviceExpression>	63
5.41 Element <AttributeAssignmentExpression>	64
5.42 Element <Request>	64
5.43 Element <RequestDefaults>	65
5.44 Element <Attributes>	66
5.45 Element <Content>	66
5.46 Element <Attribute>	66
5.47 Element <Response>	67
5.48 Element <Result>	67

5.49	Element <PolicyIdentifierList>.....	68
5.50	Element <MultiRequests>.....	69
5.51	Element <RequestReference>.....	69
5.52	Element <AttributesReference>.....	69
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	73
7	Functional requirements	75
7.1	Unicode issues	75
7.1.1	Normalization	75
7.1.2	Version of Unicode	75
7.2	Policy enforcement point	75
7.2.1	Base PEP	75
7.2.2	Deny-biased PEP	75
7.2.3	Permit-biased PEP	76
7.3	Attribute evaluation	76
7.3.1	Structured attributes	76
7.3.2	Attribute bags	76
7.3.3	Multivalued attributes.....	77
7.3.4	Attribute Matching.....	77
7.3.5	Attribute Retrieval.....	77
7.3.6	Environment Attributes	77
7.3.7	AttributeSelector evaluation.....	77
7.4	Expression evaluation.....	79
7.5	Arithmetic evaluation	79
7.6	Match evaluation.....	79
7.7	Target evaluation	80
7.8	VariableReference Evaluation	81
7.9	Condition evaluation	81
7.10	Rule evaluation	81
7.11	Policy evaluation.....	82
7.12	Policy Set evaluation	83
7.13	PolicySetIdReference and PolicyIdReference evaluation	84
7.14	Hierarchical resources	84
7.15	Authorization decision.....	84
7.16	Obligations and advice	84
7.17	Exception handling	85
7.17.1	Unsupported functionality	85
7.17.2	Syntax and type errors	85
7.17.3	Missing attributes.....	85
8	XACML extensibility points (non-normative).....	86

8.1 Extensible XML attribute types	86
8.2 Structured attributes	86
9 Security and privacy considerations (non-normative)	87
9.1 Threat model.....	87
9.1.1 Unauthorized disclosure	87
9.1.2 Message replay	87
9.1.3 Message insertion	87
9.1.4 Message deletion	88
9.1.5 Message modification.....	88
9.1.6 NotApplicable results.....	88
9.1.7 Negative rules	88
9.1.8 Denial of service.....	89
9.2 Safeguards	89
9.2.1 Authentication.....	89
9.2.2 Policy administration.....	89
9.2.3 Confidentiality	90
9.2.4 Policy integrity	90
9.2.5 Policy identifiers.....	90
9.2.6 Trust model	91
9.2.7 Privacy.....	91
9.3 Unicode security issues	92
10 Conformance.....	93
10.1 Introduction.....	93
10.2 Conformance tables.....	93
10.2.1 Schema elements.....	93
10.2.2 Identifier Prefixes.....	94
10.2.3 Algorithms	94
10.2.4 Status Codes.....	95
10.2.5 Attributes	95
10.2.6 Identifiers.....	95
10.2.7 Data-types	96
10.2.8 Functions.....	96
10.2.9 Identifiers planned for future deprecation	101
A. Data-types and functions (normative).....	102
A.1 Introduction.....	102
A.2 Data-types	102
A.3 Functions.....	104
A.3.1 Equality predicates.....	104
A.3.2 Arithmetic functions.....	106
A.3.3 String conversion functions	106
A.3.4 Numeric data-type conversion functions	107
A.3.5 Logical functions	107
A.3.6 Numeric comparison functions.....	108
A.3.7 Date and time arithmetic functions.....	108
A.3.8 Non-numeric comparison functions.....	109

A.3.9 String functions	112
A.3.10 Bag functions	116
A.3.11 Set functions	116
A.3.12 Higher-order bag functions.....	117
A.3.13 Regular-expression-based functions.....	122
A.3.14 Special match functions	123
A.3.15 XPath-based functions	124
A.3.16 Other functions.....	124
A.3.17 Extension functions and primitive types	125
A.4 Functions, data types and algorithms planned for deprecation	125
B. XACML identifiers (normative).....	127
B.1 XACML namespaces.....	127
B.2 Attribute categories	127
B.3 Data-types.....	127
B.4 Subject attributes.....	128
B.5 Resource attributes	129
B.6 Action attributes.....	129
B.7 Environment attributes	129
B.8 Status codes.....	130
B.9 Combining algorithms.....	130
C. Combining algorithms (normative).....	132
C.1 Extended Indeterminate value.....	132
C.2 Deny-overrides.....	132
C.3 Ordered-deny-overrides	134
C.4 Permit-overrides.....	134
C.5 Ordered-permit-overrides.....	135
C.6 Deny-unless-permit	136
C.7 Permit-unless-deny	136
C.8 First-applicable.....	137
C.9 Only-one-applicable	139
C.10 Legacy Deny-overrides	140
C.11 Legacy Ordered-deny-overrides.....	141
C.12 Legacy Permit-overrides	142
C.13 Legacy Ordered-permit-overrides	144
D. Acknowledgements	145
E. Revision History	146

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 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*

Disjunctive sequence

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

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

84 Rule-combining algorithm

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

86 Subject

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

88 Target

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

91 Type Unification

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

99 1.1.2 Related terms

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

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

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

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

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

106 1.2 Terminology

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

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

112

113 Listings of XACML schema appear like this.

114

115 Example code listings appear like this.

116

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

- 120 • The prefix `xacml:` stands for the XACML 3.0 namespace.
- 121 • The prefix `ds:` stands for the W3C XML Signature namespace [DS].
- 122 • The prefix `xs:` stands for the W3C XML Schema namespace [XS].
- 123 • The prefix `xf:` stands for the XQuery 1.0 and XPath 2.0 Function and Operators specification
124 namespace [XF].

125 • The prefix xml: stands for the XML namespace <http://www.w3.org/XML/1998/namespace>.
126 This specification uses the following typographical conventions in text: <XACMLElement>,
127 <ns:ForeignElement>, Attribute, Datatype, OtherCode. Terms in **bold-face italic** are intended
128 to have the meaning defined in the Glossary.

129 1.3 Schema organization and namespaces

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

132 1.4 Normative References

- 133 **[CMF]** Martin J. Dürst et al, eds., *Character Model for the World Wide Web 1.0: Fundamentals*, W3C Recommendation 15 February 2005,
134 <http://www.w3.org/TR/2005/REC-charmod-20050215/>
135
- 136 **[DS]** D. Eastlake et al., *XML-Signature Syntax and Processing*,
137 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 138 **[exc-c14n]** J. Boyer et al, eds., *Exclusive XML Canonicalization, Version 1.0*, W3C
139 Recommendation 18 July 2002, <http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>
140
- 141 **[Hancock]** Hancock, *Polymorphic Type Checking*, in Simon L. Peyton Jones,
142 *Implementation of Functional Programming Languages*, Section 8,
143 Prentice-Hall International, 1987.
- 144 **[Haskell]** Haskell, a purely functional language. Available at <http://www.haskell.org/>
145 **[Hier]** OASIS Committee Specification 01, XACML v3.0 Hierarchical Resource Profile
146 Version 1.0, August 2010, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cs-01-en.doc>
147
- 148 **[IEEE754]** IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8,
149 IEEE Product No. SH10116-TBR.
- 150 **[ISO10181-3]** ISO/IEC 10181-3:1996 Information technology – Open Systems Interconnection -
151 - Security frameworks for open systems: Access control framework.
- 152 **[Kudo00]** Kudo M and Hada S, *XML document security based on provisional authorization*,
153 Proceedings of the Seventh ACM Conference on Computer and Communications
154 Security, Nov 2000, Athens, Greece, pp 87-96.
- 155 **[LDAP-1]** RFC2256, *A summary of the X500(96) User Schema for use with LDAPv3*,
156 Section 5, M Wahl, December 1997, <http://www.ietf.org/rfc/rfc2256.txt>
- 157 **[LDAP-2]** RFC2798, *Definition of the inetOrgPerson*, M. Smith, April 2000
158 <http://www.ietf.org/rfc/rfc2798.txt>
- 159 **[MathML]** *Mathematical Markup Language (MathML)*, Version 2.0, W3C Recommendation,
160 21 October 2003. Available at: <http://www.w3.org/TR/2003/REC-MathML2-20031021/>
161
- 162 **[Multi]** OASIS Committee Specification 01, XACML v3.0 Multiple Decision Profile
163 Version 1.0, August 2010, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cs-01-en.doc>
164
- 165 **[Perritt93]** Perritt, H. Knowbots, *Permissions Headers and Contract Law*, Conference on
166 Technological Strategies for Protecting Intellectual Property in the Networked
167 Multimedia Environment, April 1993. Available at:
168 <http://www.ifla.org/documents/infopol/copyright/perh2.txt>
- 169 **[RBAC]** David Ferraiolo and Richard Kuhn, *Role-Based Access Controls*, 15th National
170 Computer Security Conference, 1992.
- 171 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
172 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

173	[RFC2396]	Berners-Lee T, Fielding R, Masinter L, <i>Uniform Resource Identifiers (URI): Generic Syntax</i> . Available at: http://www.ietf.org/rfc/rfc2396.txt
174		
175	[RFC2732]	Hinden R, Carpenter B, Masinter L, <i>Format for Literal IPv6 Addresses in URL's</i> . Available at: http://www.ietf.org/rfc/rfc2732.txt
176		
177	[RFC3198]	IETF RFC 3198: <i>Terminology for Policy-Based Management</i> , November 2001. http://www.ietf.org/rfc/rfc3198.txt
178		
179	[UAX15]	Mark Davis, Martin Dürst, <i>Unicode Standard Annex #15: Unicode Normalization Forms, Unicode 5.1</i> , available from http://unicode.org/reports/tr15/
180		
181	[UTR36]	Davis, Mark, Suignard, Michel, <i>Unicode Technocal Report #36: Unicode Security Considerations</i> . Available at http://www.unicode.org/reports/tr36/
182		
183	[XACMLAdmin]	OASIS Committee Draft 03, <i>XACML v3.0 Administration and Delegation Profile Version 1.0</i> . 11 March 2010. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-administration-v1-spec-cd-03-en.doc
184		
185		
186	[XACMLv1.0]	OASIS Standard, <i>Extensible access control markup language (XACML) Version 1.0</i> . 18 February 2003. http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf
187		
188		
189	[XACMLv1.1]	OASIS Committee Specification, <i>Extensible access control markup language (XACML) Version 1.1</i> . 7 August 2003. http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf
190		
191		
192		
193	[XACML v3.0]	OASIS Committee Specification 01, <i>eXtensible access control markup language (XACML) Version 3.0</i> . August 2010. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.doc
194		
195		
196		
197	[XF]	<i>XQuery 1.0 and XPath 2.0 Functions and Operators</i> , W3C Recommendation 23 January 2007. Available at: http://www.w3.org/TR/2007/REC-xpath-functions-20070123/
198		
199		
200	[XML]	Bray, Tim, et.al. eds, <i>Extensible Markup Language (XML) 1.0 (Fifth Edition)</i> , W3C Recommendation 26 November 2008, available at http://www.w3.org/TR/2008/REC-xml-20081126/
201		
202		
203	[XMLid]	Marsh, Jonathan, et.al. eds, <i>xml:id Version 1.0</i> . W3C Recommendation 9 September 2005. Available at: http://www.w3.org/TR/2005/REC-xml-id-20050909/
204		
205		
206	[XS]	<i>XML Schema, parts 1 and 2</i> . Available at: http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/
207		
208	[XPath]	<i>XML Path Language (XPath), Version 1.0</i> , W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xpath
209		
210	[XSLT]	<i>XSL Transformations (XSLT) Version 1.0</i> , W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xslt
211		

212 1.5 Non-Normative References

213	[CM]	<i>Character model model for the World Wide Web 1.0: Normalization</i> , W3C Working Draft, 27 October 2005, http://www.w3.org/TR/2005/WD-charmod-norm-20051027/ , World Wide Web Consortium.
214		
215		
216	[Hinton94]	Hinton, H, M, Lee, E, S, <i>The Compatibility of Policies</i> , Proceedings 2nd ACM Conference on Computer and Communications Security, Nov 1994, Fairfax, Virginia, USA.
217		
218		
219	[Sloman94]	Sloman, M. <i>Policy Driven Management for Distributed Systems</i> . Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
220		

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, and be re-used in multiple **policies**.

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**. Standard combining algorithms are defined for:

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

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

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

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

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

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

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

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

319 2.4 Multiple subjects

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

326 2.5 Policies based on subject and resource attributes

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

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

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

343 2.6 Multi-valued attributes

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

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

353 2.7 Policies based on resource contents

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

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

360 <Content> element of the **resource** to identify data in the information **resource** to be used in the **policy**
361 evaluation.

362 In cases where the information **resource** is not an XML document, specified **attributes** of the **resource**
363 can be referenced, as described in Section 2.5.

364 2.8 Operators

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

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

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

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

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

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

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

393 2.9 Policy distribution

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

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

405 2.10 Policy indexing

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

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

410 Two approaches are supported:

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

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

419 2.11 Abstraction layer

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

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

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

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

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

441 2.12 Actions performed in conjunction with enforcement

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

451 2.13 Supplemental information about a decision

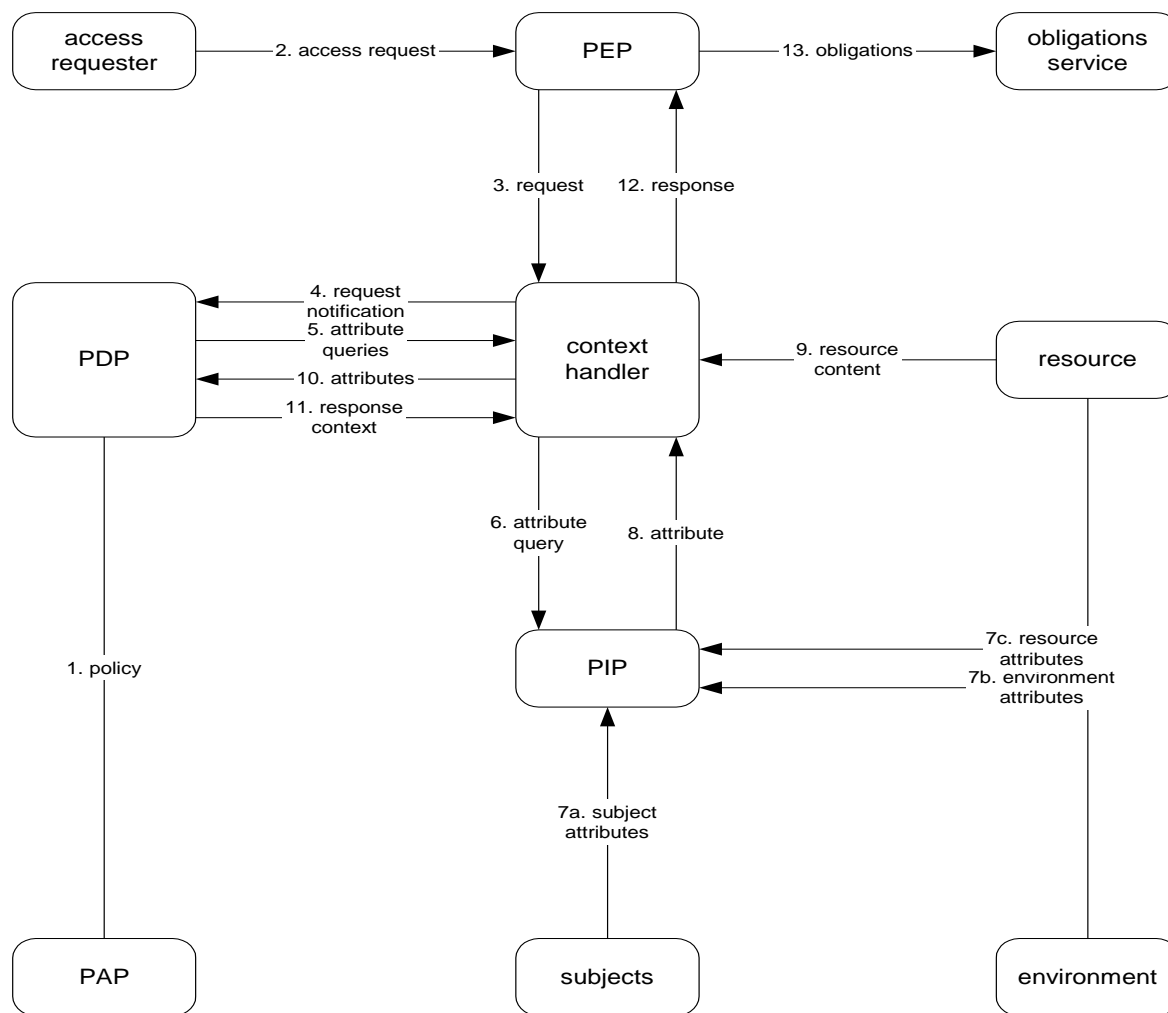
452 In some applications it is helpful to specify supplemental information about a decision. XACML provides
453 facilities to specify supplemental information about a decision with the <Advice> element. Such **advice**
454 may be safely ignored by the **PEP**.

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.



459
460 Figure 1 - Data-flow diagram

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

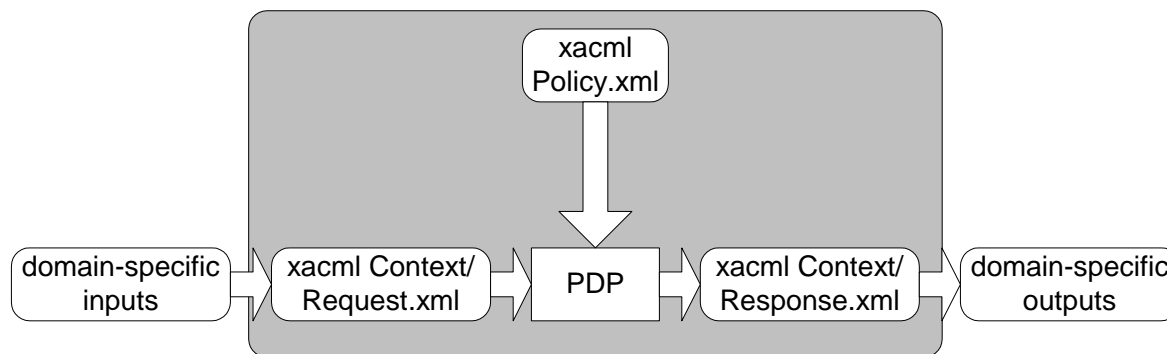
467 The model operates by the following steps.

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

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

490 3.2 XACML context

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



502
503 *Figure 2 - XACML context*

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

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

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

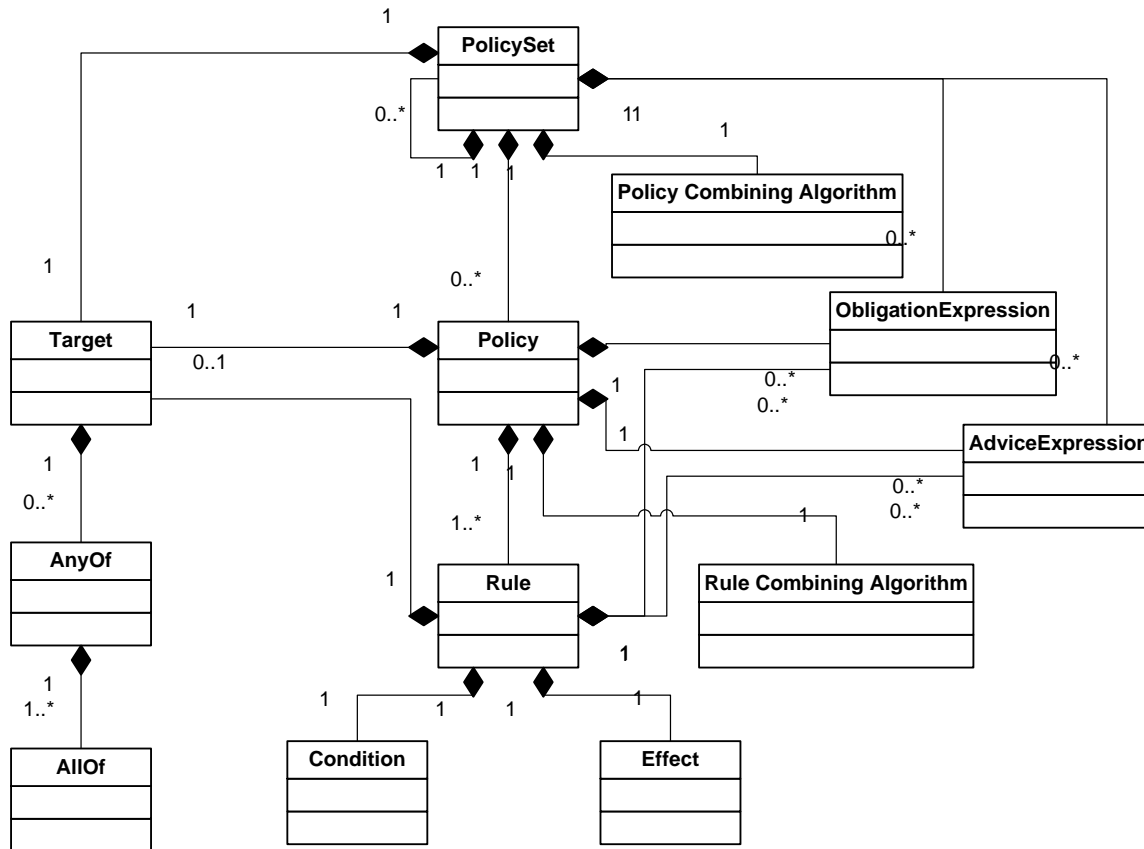
509 **3.3 Policy language model**

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

- 511 • **Rule**;
- 512 • **Policy**; and
- 513 • **Policy set**.

514 These are described in the following sub-sections.

515



516

517 *Figure 3 - Policy language model*

518 **3.3.1 Rule**

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

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

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

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

527 These are discussed in the following sub-sections.

528 3.3.1.1 Rule target

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

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

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

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

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

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

554 3.3.1.2 Effect

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

557 3.3.1.3 Condition

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

560 3.3.1.4 Obligation expressions

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

562 When a **PDP** evaluates a **rule** containing **obligation** expressions, it evaluates the **obligation**
563 expressions into **obligations** and returns certain of those **obligations** to the **PEP** in the response
564 **context**. Section 7.16 explains which **obligations** are to be returned.

565 3.3.1.5 Advice

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

567 When a **PDP** evaluates a **rule** containing **advice** expressions, it evaluates the **advice** expressions into
568 **advice** and returns certain of those **advice** to the **PEP** in the response **context**. Section 7.16 explains
569 which **advice** are to be returned. In contrast to **obligations**, **advice** may be safely ignored by the **PEP**.

570 3.3.2 Policy

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

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

579 3.3.2.1 Policy target

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

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

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

599 3.3.2.2 Rule-combining algorithm

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

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

605 3.3.2.3 Obligation expressions

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

607 When a **PDP** evaluates a **policy** containing **obligation** expressions, it evaluates the **obligation**
608 expressions into **obligations** and returns certain of those **obligations** to the **PEP** in the response
609 **context**. Section 7.16 explains which **obligations** are to be returned.

610 3.3.2.4 Advice

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

612 When a **PDP** evaluates a **policy** containing **advice** expressions, it evaluates the **advice** expressions into
613 **advice** and returns certain of those **advice** to the **PEP** in the response **context**. Section 7.16 explains
614 which **advice** are to be returned. In contrast to **obligations**, **advice** may be safely ignored by the **PEP**.

615 3.3.3 Policy set

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

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

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

624 3.3.3.1 Policy-combining algorithm

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

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

631 3.3.3.2 Obligation expressions

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

634 When a **PDP** evaluates a **policy set** containing **obligations** expressions, it evaluates the **obligation**
635 expressions into **obligations** and returns certain of those **obligations** to the **PEP** in its response **context**.
636 Section 7.16 explains which **obligations** are to be returned.

637 3.3.3.3 Advice expressions

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

639 When a **PDP** evaluates a **policy set** containing **advice** expressions, it evaluates the **advice** expressions
640 into **advice** and returns certain of those **advice** to the **PEP** in the response **context**. Section 7.16
641 explains which **advice** are to be returned. In contrast to **obligations**, **advice** may be safely ignored by
642 the **PEP**.

643 4 Examples (non-normative)

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

648 4.1 Example one

649 4.1.1 Example policy

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

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

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

```
656 [a1] <?xml version="1.0" encoding="UTF-8"?>
657 [a2] <Policy
658 [a3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
659 [a4]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
660 [a5]   xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
661 [a6]   http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
662 [a7]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy1"
663 [a8]   Version="1.0"
664 [a9]   RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
665 [a10]  <Description>
666 [a11]    Medi Corp access control policy
667 [a12]  </Description>
668 [a13]  <Target/>
669 [a14]  <Rule
670 [a15]    RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule1"
671 [a16]    Effect="Permit">
672 [a17]    <Description>
673 [a18]      Any subject with an e-mail name in the med.example.com domain
674 [a19]      can perform any action on any resource.
675 [a20]    </Description>
676 [a21]    <Target>
677 [a22]      <AnyOf>
678 [a23]        <AllOf>
679 [a24]          <Match
680 [a25]            MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
681 [a26]            <AttributeValue
682 [a27]              DataType="http://www.w3.org/2001/XMLSchema#string"
683 [a28]              >med.example.com</AttributeValue>
684 [a29]            <AttributeDesignator
685 [a30]              MustBePresent="false"
686 [a31]              Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
687 [a32] subject"
688 [a33]              AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
689 [a34]              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
690 [a35]            </Match>
691 [a36]          </AllOf>
692 [a37]        </AnyOf>
693 [a38]      </Target>
694 [a39]    </Rule>
695 [a40]  </Policy>
```

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

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

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

742 4.1.2 Example request context

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

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

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

```

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

```

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

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

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

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

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

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

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

806 4.1.3 Example response context

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

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

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

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

824 [c6] closes the response *context*.

825 4.2 Example two

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

829 4.2.1 Example medical record instance

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

```
832 [d1] <?xml version="1.0" encoding="UTF-8"?>  
833 [d2] <record xmlns="urn:example:med:schemas:record"  
834 [d3]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
835 [d4]   <patient>  
836 [d5]     <patientName>  
837 [d6]       <first>Bartholomew</first>  
838 [d7]       <last>Simpson</last>  
839 [d8]     </patientName>  
840 [d9]     <patientContact>  
841 [d10]       <street>27 Shelbyville Road</street>  
842 [d11]       <city>Springfield</city>  
843 [d12]       <state>MA</state>  
844 [d13]       <zip>12345</zip>  
845 [d14]       <phone>555.123.4567</phone>  
846 [d15]       <fax/>  
847 [d16]       <email/>  
848 [d17]     </patientContact>  
849 [d18]     <patientDoB>1992-03-21</patientDoB>  
850 [d19]     <patientGender>male</patientGender>  
851 [d20]     <patient-number>555555</patient-number>  
852 [d21]   </patient>  
853 [d22]   <parentGuardian>  
854 [d23]     <parentGuardianId>HS001</parentGuardianId>  
855 [d24]     <parentGuardianName>  
856 [d25]       <first>Homer</first>  
857 [d26]       <last>Simpson</last>  
858 [d27]     </parentGuardianName>  
859 [d28]     <parentGuardianContact>  
860 [d29]       <street>27 Shelbyville Road</street>  
861 [d30]       <city>Springfield</city>  
862 [d31]       <state>MA</state>  
863 [d32]       <zip>12345</zip>  
864 [d33]       <phone>555.123.4567</phone>  
865 [d34]     <fax/>
```

```

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

```

914 4.2.2 Example request context

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

```

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

```

```

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

```

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

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

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

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

1000 [e21] - [e26] **Subject** physician-id **attribute**.

1001 [e28] - [e56] **Resource attributes** are placed in the urn:oasis:names:tc:xacml:3.0:attribute-
 1002 category:resource **attribute** category of the <Request> element. Each **attribute** consists of **attribute**
 1003 meta-data and an **attribute** value.

1004 [e30] - [e42] **Resource** content. The XML **resource** instance, **access** to all or part of which may be
 1005 requested, is placed here.

1006 [e43] - [e49] The identifier of the **Resource** instance for which **access** is requested, which is an XPath
 1007 expression into the <Content> element that selects the data to be accessed.

1008 [e57] - [e64] **Action attributes** are placed in the urn:oasis:names:tc:xacml:3.0:attribute-category:action
 1009 **attribute** category of the <Request> element.

1010 [e59] - [e63] **Action** identifier.

1011 4.2.3 Example plain-language rules

1012 The following plain-language **rules** are to be enforced:

- 1013 Rule 1: A person, identified by his or her patient number, may read any record for which he or she is
 1014 the designated patient.
- 1015 Rule 2: A person may read any record for which he or she is the designated parent or guardian, and
 1016 for which the patient is under 16 years of age.
- 1017 Rule 3: A physician may write to any medical element for which he or she is the designated primary
 1018 care physician, provided an email is sent to the patient.
- 1019 Rule 4: An administrator shall not be permitted to read or write to medical elements of a patient
 1020 record.

1021 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

1022 4.2.4 Example XACML rule instances

1023 4.2.4.1 Rule 1

1024 **Rule 1** illustrates a simple **rule** with a single <Condition> element. It also illustrates the use of the
 1025 <VariableDefinition> element to define a function that may be used throughout the **policy**. The
 1026 following XACML <Rule> instance expresses **Rule 1**:

```

1027 [f1] <?xml version="1.0" encoding="UTF-8"?>
1028 [f2] <Policy
1029 [f3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1030 [f4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1031 [f5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1032 [f6]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1033 [f7]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:1"
1034 [f8]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1035 algorithm:deny-overrides"
1036 [f9]   Version="1.0">
1037 [f10]   <PolicyDefaults>
1038 [f11]     <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1039 [f12]   </PolicyDefaults>
1040 [f13]   <Target/>
1041 [f14]   <VariableDefinition VariableId="17590034">
1042 [f15]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1043 [f16]       <Apply
1044 [f17]         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1045 [f18]           <AttributeDesignator
1046 [f19]             MustBePresent="false"
1047 [f20]             Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
1048 subject"
1049 [f21]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:patient-
1050 number"
1051 [f22]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1052 [f23]           </Apply>
1053 [f24]         <Apply
1054 [f25]           FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">

```

```

1055 [f26]         <AttributeSelector
1056 [f27]             MustBePresent="false"
1057 [f28]             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1058 [f29]             Path="md:record/md:patient/md:patient-number/text()"
1059 [f30]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1060 [f31]         </Apply>
1061 [f32]     </Apply>
1062 [f33] </VariableDefinition>
1063 [f34] <Rule
1064 [f35]     RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"
1065 [f36]     Effect="Permit">
1066 [f37]     <Description>
1067 [f38]         A person may read any medical record in the
1068 [f39]         http://www.med.example.com/schemas/record.xsd namespace
1069 [f40]         for which he or she is the designated patient
1070 [f41]     </Description>
1071 [f42]     <Target>
1072 [f43]         <AnyOf>
1073 [f44]             <AllOf>
1074 [f45]                 <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1075 [f46]                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1076 [f47]                         >urn:example:med:schemas:record</AttributeValue>
1077 [f48]                     <AttributeDesignator
1078 [f49]                         MustBePresent="false"
1079 [f50]                         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1080 [f51]                         AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1081 [f52]                         DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1082 [f53]                 </Match>
1083 [f54]                 <Match
1084 [f55]                     MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1085 [f56]                         <AttributeValue
1086 [f57]                             DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1087 [f58]                             XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1088 [f59]                             >md:record</AttributeValue>
1089 [f60]                         <AttributeDesignator
1090 [f61]                             MustBePresent="false"
1091 [f62]                             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1092 [f63]                             AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1093 [f64]                             DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1094 [f65]                         </Match>
1095 [f66]                     </AllOf>
1096 [f67]                 </AnyOf>
1097 [f68]             <AnyOf>
1098 [f69]                 <AllOf>
1099 [f70]                     <Match
1100 [f71]                         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1101 [f72]                             <AttributeValue
1102 [f73]                                 DataType="http://www.w3.org/2001/XMLSchema#string"
1103 [f74]                                 >read</AttributeValue>
1104 [f75]                             <AttributeDesignator
1105 [f76]                                 MustBePresent="false"
1106 [f77]                                 Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1107 [f78]                                 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1108 [f79]                                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1109 [f80]                             </Match>
1110 [f81]                     </AllOf>
1111 [f82]                 </AnyOf>
1112 [f83]             </Target>
1113 [f84]         <Condition>
1114 [f85]             <VariableReference VariableId="17590034"/>
1115 [f86]         </Condition>
1116 [f87]     </Rule>
1117 [f88] </Policy>

```

1118 [f3] - [f6] XML namespace declarations.

1119 [f11] XPath expressions in the *policy* are to be interpreted according to the 1.0 version of the XPath specification.

1121 [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*.

1123 [f15] The `FunctionId` attribute names the function to be used for comparison. In this case, comparison
1124 is done with the “urn:oasis:names:tc:xacml:1.0:function:string-equal” function; this function takes two
1125 arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1126 [f17] The first argument of the variable definition is a function specified by the `FunctionId` attribute.
1127 Since urn:oasis:names:tc:xacml:1.0:function:string-equal takes arguments of type
1128 “http://www.w3.org/2001/XMLSchema#string” and `AttributeDesignator` selects a **bag** of type
1129 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1130 only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly one
1131 value.

1132 [f18] The `AttributeDesignator` selects a **bag** of values for the patient-number **subject attribute** in
1133 the request **context**.

1134 [f25] The second argument of the variable definition is a function specified by the `FunctionId` attribute.
1135 Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes arguments of type
1136 “http://www.w3.org/2001/XMLSchema#string” and the `AttributeSelector` selects a **bag** of type
1137 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-and-
1138 only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly one
1139 value.

1140 [f26] The `<AttributeSelector>` element selects a **bag** of values from the **resource** content using a
1141 free-form XPath expression. In this case, it selects the value of the patient-number in the **resource**.
1142 Note that the namespace prefixes in the XPath expression are resolved with the standard XML
1143 namespace declarations.

1144 [f35] **Rule** identifier.

1145 [f36] **Rule effect** declaration. When a **rule** evaluates to ‘True’ it emits the value of the `Effect` attribute.
1146 This value is then combined with the `Effect` values of other **rules** according to the **rule-combining**
1147 **algorithm**.

1148 [f37] - [f41] Free form description of the **rule**.

1149 [f42] - [f83] A **rule target** defines a set of **decision requests** that the **rule** is intended to evaluate.

1150 [f43] - [f67] The `<AnyOf>` element contains a **disjunctive sequence** of `<AllOf>` elements. In this
1151 example, there is just one.

1152 [f44] - [f66] The `<AllOf>` element encloses the **conjunctive sequence** of `Match` elements. In this
1153 example, there are two.

1154 [f45] - [f53] The first `<Match>` element compares its first and second child elements according to the
1155 matching function. A match is positive if the value of the first argument matches any of the values
1156 selected by the second argument. This match compares the **target** namespace of the requested
1157 document with the value of “urn:example:med:schemas:record”.

1158 [f45] The `MatchId` attribute names the matching function.

1159 [f46] - [f47] Literal **attribute** value to match.

1160 [f48] - [f52] The `<AttributeDesignator>` element selects the **target** namespace from the **resource**
1161 contained in the request **context**. The **attribute** name is specified by the `AttributeId`.

1162 [f54] - [f65] The second `<Match>` element. This match compares the results of two XPath expressions
1163 applied to the `<Content>` element of the **resource** category. The second XPath expression is the
1164 location path to the requested XML element and the first XPath expression is the literal value “md:record”.
1165 The “xpath-node-match” function evaluates to “True” if the requested XML element is below the
1166 “md:record” element.

1167 [f68] - [f82] The `<AnyOf>` element contains a **disjunctive sequence** of `<AllOf>` elements. In this case,
1168 there is just one `<AllOf>` element.

1169 [f69] - [f81] The `<AllOf>` element contains a **conjunctive sequence** of `<Match>` elements. In this case,
1170 there is just one `<Match>` element.

1171 [f70] - [f80] The <Match> element compares its first and second child elements according to the matching
1172 function. The match is positive if the value of the first argument matches any of the values selected by
1173 the second argument. In this case, the value of the action-id **action attribute** in the request **context** is
1174 compared with the literal value "read".

1175 [f84] - [f86] The <Condition> element. A **condition** must evaluate to "True" for the **rule** to be
1176 applicable. This **condition** contains a reference to a variable definition defined elsewhere in the **policy**.

1177 4.2.4.2 Rule 2

1178 **Rule 2** illustrates the use of a mathematical function, i.e. the <Apply> element with functionId
1179 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate the date of the patient's
1180 sixteenth birthday. It also illustrates the use of **predicate** expressions, with the functionId
1181 "urn:oasis:names:tc:xacml:1.0:function:and". This example has one function embedded in the
1182 <Condition> element and another one referenced in a <VariableDefinition> element.

```
1183 [g1] <?xml version="1.0" encoding="UTF-8"?>
1184 [g2] <Policy
1185 [g3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1186 [g4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1187 [g5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1188 [g6]   xmlns:xf="http://www.w3.org/2005/xpath-functions"
1189 [g7]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1190 [g8]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"
1191 [g9]   Version="1.0"
1192 [g10]  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1193      algorithm:deny-overrides">
1194 [g11]  <PolicyDefaults>
1195 [g12]    <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1196 [g13]  </PolicyDefaults>
1197 [g14]  <Target/>
1198 [g15]  <VariableDefinition VariableId="17590035">
1199 [g16]    <Apply
1200 [g17]      FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal">
1201 [g18]      <Apply
1202 [g19]        FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
1203 [g20]          <AttributeDesignator
1204 [g21]            MustBePresent="false"
1205 [g22]            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
1206 [g23]            AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
1207 [g24]            DataType="http://www.w3.org/2001/XMLSchema#date"/>
1208 [g25]          </Apply>
1209 [g26]        <Apply
1210 [g27]          FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
1211 [g28]            <Apply
1212 [g29]              FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
1213 [g30]                <AttributeSelector
1214 [g31]                  MustBePresent="false"
1215 [g32]                  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1216 [g33]                  Path="md:record/md:patient/md:patientDoB/text () "
1217 [g34]                  DataType="http://www.w3.org/2001/XMLSchema#date"/>
1218 [g35]                </Apply>
1219 [g36]              <AttributeValue
1220 [g37]                DataType="http://www.w3.org/2001/XMLSchema#yearMonthDuration"
1221 [g38]                >P16Y</AttributeValue>
1222 [g39]            </Apply>
1223 [g40]          </Apply>
1224 [g41]        </VariableDefinition>
1225 [g42]      <Rule
1226 [g43]        RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
1227 [g44]        Effect="Permit">
1228 [g45]        <Description>
1229 [g46]          A person may read any medical record in the
1230 [g47]          http://www.med.example.com/records.xsd namespace
1231 [g48]          for which he or she is the designated parent or guardian,
1232 [g49]          and for which the patient is under 16 years of age
1233 [g50]        </Description>
1234 [g51]      <Target>
1235 [g52]        <AnyOf>
1236 [g53]      <AllOf>
```

```

1237 [g54] <Match
1238 [g55]   MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1239 [g56]   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1240 [g57]     >urn:example:med:schemas:record</AttributeValue>
1241 [g58]   <AttributeDesignator
1242 [g59]     MustBePresent="false"
1243 [g60]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1244 [g61]   AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1245 [g62]   DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1246 [g63] </Match>
1247 [g64] <Match
1248 [g65]   MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1249 [g66]   <AttributeValue
1250 [g67]     DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1251 [g68]   XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1252 [g69]     >md:record</AttributeValue>
1253 [g70]   <AttributeDesignator
1254 [g71]     MustBePresent="false"
1255 [g72]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1256 [g73]   AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1257 [g74]   DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1258 [g75] </Match>
1259 [g76] </AllOf>
1260 [g77] </AnyOf>
1261 [g78] <AnyOf>
1262 [g79]   <AllOf>
1263 [g80]     <Match
1264 [g81]       MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1265 [g82]       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1266 [g83]         >read</AttributeValue>
1267 [g84]       <AttributeDesignator
1268 [g85]         MustBePresent="false"
1269 [g86]         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1270 [g87]       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1271 [g88]       DataType="http://www.w3.org/2001/XMLSchema#string"/>
1272 [g89]     </Match>
1273 [g90]   </AllOf>
1274 [g91] </AnyOf>
1275 [g92] </Target>
1276 [g93] <Condition>
1277 [g94]   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1278 [g95]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1279 [g96]       <Apply
1280 [g97]         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1281 [g98]           <AttributeDesignator
1282 [g99]             MustBePresent="false"
1283 [g100]           Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1284 [g101]           AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:parent-
1285 guardian-id"
1286 [g102]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1287 [g103]         </Apply>
1288 [g104]       <Apply
1289 [g105]         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1290 [g106]           <AttributeSelector
1291 [g107]             MustBePresent="false"
1292 [g108]           Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1293 [g109]           Path="md:record/md:parentGuardian/md:parentGuardianId/text()"
1294 [g110]           DataType="http://www.w3.org/2001/XMLSchema#string"/>
1295 [g111]         </Apply>
1296 [g112]       </Apply>
1297 [g113]     <VariableReference VariableId="17590035"/>
1298 [g114]   </Apply>
1299 [g115] </Condition>
1300 [g116] </Rule>
1301 [g117] </Policy>

```

1302 [g15] - [g41] The <VariableDefinition> element contains part of the **condition** (i.e. is the patient
1303 under 16 years of age?). The patient is under 16 years of age if the current date is less than the date
1304 computed by adding 16 to the patient's date of birth.

1305 [g16] - [g40] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to compare the two date
1306 arguments.

1307 [g18] - [g25] The first date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-one-and-only” to
 1308 ensure that the **bag** of values selected by its argument contains exactly one value of type
 1309 “http://www.w3.org/2001/XMLSchema#date”.

1310 [g20] The current date is evaluated by selecting the “urn:oasis:names:tc:xacml:1.0:environment:current-
 1311 date” **environment attribute**.

1312 [g26] - [g39] The second date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-add-
 1313 yearMonthDuration” to compute the date of the patient’s sixteenth birthday by adding 16 years to the
 1314 patient’s date of birth. The first of its arguments is of type “http://www.w3.org/2001/XMLSchema#date”
 1315 and the second is of type “http://www.w3.org/TR/2007/REC-xpath-functions-20070123/#dt-
 1316 yearMonthDuration”.

1317 [g30] The <AttributeSelector> element selects the patient’s date of birth by taking the XPath
 1318 expression over the **resource** content.

1319 [g36] - [g38] Year Month Duration of 16 years.

1320 [g51] - [g92] **Rule** declaration and **rule target**. See **Rule 1** in Section 4.2.4.1 for the detailed explanation
 1321 of these elements.

1322 [g93] - [g115] The <Condition> element. The **condition** must evaluate to “True” for the **rule** to be
 1323 applicable. This **condition** evaluates the truth of the statement: the requestor is the designated parent or
 1324 guardian and the patient is under 16 years of age. It contains one embedded <Apply> element and one
 1325 referenced <VariableDefinition> element.

1326 [g94] The **condition** uses the “urn:oasis:names:tc:xacml:1.0:function:and” function. This is a Boolean
 1327 function that takes one or more Boolean arguments (2 in this case) and performs the logical “AND”
 1328 operation to compute the truth value of the expression.

1329 [g95] - [g112] The first part of the **condition** is evaluated (i.e. is the requestor the designated parent or
 1330 guardian?). The function is “urn:oasis:names:tc:xacml:1.0:function:string-equal” and it takes two
 1331 arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1332 [g96] designates the first argument. Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes
 1333 arguments of type “http://www.w3.org/2001/XMLSchema#string”,
 1334 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that the **subject attribute**
 1335 “urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id” in the request **context** contains
 1336 exactly one value.

1337 [g98] designates the first argument. The value of the **subject attribute**
 1338 “urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id” is selected from the request **context**
 1339 using the <AttributeDesignator> element.

1340 [g104] As above, the “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that
 1341 the **bag** of values selected by its argument contains exactly one value of type
 1342 “http://www.w3.org/2001/XMLSchema#string”.

1343 [g106] The second argument selects the value of the <md:parentGuardianId> element from the
 1344 **resource** content using the <AttributeSelector> element. This element contains a free-form XPath
 1345 expression, pointing into the <Content> element of the resource category. Note that all namespace
 1346 prefixes in the XPath expression are resolved with standard namespace declarations. The
 1347 AttributeSelector evaluates to the **bag** of values of type
 1348 “http://www.w3.org/2001/XMLSchema#string”.

1349 [g113] references the <VariableDefinition> element, where the second part of the **condition** is
 1350 defined.

1351 4.2.4.3 Rule 3

1352 **Rule 3** illustrates the use of an **obligation** expression.

```

1353 [h1] <?xml version="1.0" encoding="UTF-8"?>
1354 [h2] <Policy
1355 [h3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1356 [h4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1357 [h5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

```

```

1358 [h6] xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
1359 http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
1360 [h7] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1361 [h8] PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:3"
1362 [h9] Version="1.0"
1363 [h10] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1364 algorithm:deny-overrides">
1365 [h11] <Description>
1366 [h12] Policy for any medical record in the
1367 [h13] http://www.med.example.com/schemas/record.xsd namespace
1368 [h14] </Description>
1369 [h15] <PolicyDefaults>
1370 [h16] <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1371 [h17] </PolicyDefaults>
1372 [h18] <Target>
1373 [h19] <AnyOf>
1374 [h20] <AllOf>
1375 [h21] <Match
1376 [h22] MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1377 [h23] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1378 [h24] >urn:example:med:schemas:record</AttributeValue>
1379 [h25] <AttributeDesignator
1380 [h26] MustBePresent="false"
1381 [h27] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1382 [h28] AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1383 [h29] DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1384 [h30] </Match>
1385 [h31] </AllOf>
1386 [h32] </AnyOf>
1387 [h33] </Target>
1388 [h34] <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:3"
1389 [h35] Effect="Permit">
1390 [h36] <Description>
1391 [h37] A physician may write any medical element in a record
1392 [h38] for which he or she is the designated primary care
1393 [h39] physician, provided an email is sent to the patient
1394 [h40] </Description>
1395 [h41] <Target>
1396 [h42] <AnyOf>
1397 [h43] <AllOf>
1398 [h44] <Match
1399 [h45] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1400 [h46] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1401 [h47] >physician</AttributeValue>
1402 [h48] <AttributeDesignator
1403 [h49] MustBePresent="false"
1404 [h50] Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1405 [h51] AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
1406 [h52] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1407 [h53] </Match>
1408 [h54] </AllOf>
1409 [h55] </AnyOf>
1410 [h56] <AnyOf>
1411 [h57] <AllOf>
1412 [h58] <Match
1413 [h59] MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1414 [h60] <AttributeValue
1415 [h61] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1416 [h62] XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1417 [h63] >md:record/md:medical</AttributeValue>
1418 [h64] <AttributeDesignator
1419 [h65] MustBePresent="false"
1420 [h66] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1421 [h67] AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1422 [h68] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1423 [h69] </Match>
1424 [h70] </AllOf>
1425 [h71] </AnyOf>
1426 [h72] <AnyOf>
1427 [h73] <AllOf>
1428 [h74] <Match
1429 [h75] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1430 [h76] <AttributeValue

```

```

1431 [h77]         DataType="http://www.w3.org/2001/XMLSchema#string"
1432 [h78]         >write</AttributeValue>
1433 [h79]         <AttributeDesignator
1434 [h80]             MustBePresent="false"
1435 [h81]             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1436 [h82]             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1437 [h83]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1438 [h84]         </Match>
1439 [h85]     </AllOf>
1440 [h86] </AnyOf>
1441 [h87] </Target>
1442 [h88] <Condition>
1443 [h89]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1444 [h90]         <Apply
1445 [h91]             FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1446 [h92]             <AttributeDesignator
1447 [h93]                 MustBePresent="false"
1448 [h94]                 Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1449 [h95]                 AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id"
1450 [h96]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1451 [h97]             </Apply>
1452 [h98]         <Apply
1453 [h99]             FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1454 [h100]            <AttributeSelector
1455 [h101]                MustBePresent="false"
1456 [h102]                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1457 [h103]                Path="md:record/md:primaryCarePhysician/md:registrationID/text()"
1458 [h104]                DataType="http://www.w3.org/2001/XMLSchema#string"/>
1459 [h105]            </Apply>
1460 [h106]        </Apply>
1461 [h107]    </Condition>
1462 [h108] </Rule>
1463 [h109] <ObligationExpressions>
1464 [h110]     <ObligationExpression
1465 [h111]         ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
1466 [h112]         FulfillOn="Permit">
1467 [h113]         <AttributeAssignmentExpression
1468 [h114]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:mailto">
1469 [h115]             <AttributeSelector
1470 [h116]                 MustBePresent="true"
1471 [h117]                 Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1472 [h118]                 Path="md:record/md:patient/md:patientContact/md:email"
1473 [h119]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1474 [h120]             </AttributeAssignmentExpression>
1475 [h121]             <AttributeAssignmentExpression
1476 [h122]                 AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
1477 [h123]                 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1478 [h124]                     >Your medical record has been accessed by:</AttributeValue>
1479 [h125]                 </AttributeAssignmentExpression>
1480 [h126]             </AttributeAssignmentExpression>
1481 [h127]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
1482 [h128]             <AttributeDesignator
1483 [h129]                 MustBePresent="false"
1484 [h130]                 Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1485 [h131]                 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1486 [h132]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1487 [h133]             </AttributeAssignmentExpression>
1488 [h134]         </ObligationExpression>
1489 [h135]     </ObligationExpressions>
1490 [h136] </Policy>

```

1491 [h2] - [h10] The <Policy> element includes standard namespace declarations as well as **policy** specific
1492 parameters, such as PolicyId and RuleCombiningAlgId.

1493 [h8] **Policy** identifier. This parameter allows the **policy** to be referenced by a **policy set**.

1494 [h10] The **Rule-combining algorithm** identifies the algorithm for combining the outcomes of **rule**
1495 evaluation.

1496 [h11] - [h14] Free-form description of the **policy**.

1497 [h18] - [h33] **Policy target**. The **policy target** defines a set of applicable **decision requests**. The
1498 structure of the <Target> element in the <Policy> is identical to the structure of the <Target>

1499 element in the <Rule>. In this case, the **policy target** is the set of all XML **resources** that conform to
1500 the namespace “urn:example:med:schemas:record”.

1501 [h34] - [h108] The only <Rule> element included in this <Policy>. Two parameters are specified in the
1502 **rule** header: RuleId and Effect.

1503 [h41] - [h87] The **rule target** further constrains the **policy target**.

1504 [h44] - [h53] The <Match> element targets the **rule** at **subjects** whose
1505 “urn:oasis:names:tc:xacml:3.0:example:attribute:role” **subject attribute** is equal to “physician”.

1506 [h58] - [h69] The <Match> element targets the **rule** at **resources** that match the XPath expression
1507 “md:record/md:medical”.

1508 [h74] - [h84] The <Match> element targets the **rule** at **actions** whose
1509 “urn:oasis:names:tc:xacml:1.0:action:action-id” **action attribute** is equal to “write”.

1510 [h88] - [h107] The <Condition> element. For the **rule** to be applicable to the **decision request**, the
1511 **condition** must evaluate to “True”. This **condition** compares the value of the
1512 “urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id” **subject attribute** with the value of the
1513 <registrationId> element in the medical record that is being accessed.

1514 [h109] - [h134] The <ObligationExpressions> element. **Obligations** are a set of operations that
1515 must be performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be
1516 associated with a “Permit” or “Deny” **authorization decision**. The element contains a single **obligation**
1517 expression, which will be evaluated into an obligation when the policy is evaluated.

1518 [h110] - [h133] The <ObligationExpression> element consists of the ObligationId attribute, the
1519 **authorization decision** value for which it must be fulfilled, and a set of **attribute** assignments.

1520 [h110] The ObligationId attribute identifies the **obligation**. In this case, the **PEP** is required to send
1521 email.

1522 [h111] The FulfillOn attribute defines the **authorization decision** value for which the **obligation**
1523 derived from the **obligation** expression must be fulfilled. In this case, the **obligation** must be fulfilled
1524 when **access** is permitted.

1525 [h112] - [h119] The first parameter indicates where the **PEP** will find the email address in the **resource**.
1526 The **PDP** will evaluate the <AttributeSelector> and return the result to the **PEP** inside the resulting
1527 **obligation**.

1528 [h120] - [h123] The second parameter contains literal text for the email body.

1529 [h125] - [h132] The third parameter indicates where the **PEP** will find further text for the email body in the
1530 **resource**. The **PDP** will evaluate the <AttributeDesignator> and return the result to the **PEP** inside
1531 the resulting **obligation**.

1532 4.2.4.4 Rule 4

1533 **Rule 4** illustrates the use of the “Deny” **Effect** value, and a <Rule> with no <Condition> element.

```

1534 [i1] <?xml version="1.0" encoding="UTF-8"?>
1535 [i2] <Policy
1536 [i3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1537 [i4]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1538 [i5]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1539 [i6]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:4"
1540 [i7]   Version="1.0"
1541 [i8]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1542   algorithm:deny-overrides">
1543 [i9]   <PolicyDefaults>
1544 [i10]     <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1545 [i11]   </PolicyDefaults>
1546 [i12]   <Target/>
1547 [i13]   <Rule
1548 [i14]     RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:4"
1549 [i15]     Effect="Deny">
1550 [i16]   <Description>
1551 [i17]     An Administrator shall not be permitted to read or write

```

```

1552 [i18] medical elements of a patient record in the
1553 [i19] http://www.med.example.com/records.xsd namespace.
1554 [i20] </Description>
1555 [i21] <Target>
1556 [i22] <AnyOf>
1557 [i23] <AllOf>
1558 [i24] <Match
1559 [i25] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1560 [i26] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1561 [i27] >administrator</AttributeValue>
1562 [i28] <AttributeDesignator
1563 [i29] MustBePresent="false"
1564 [i30] Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1565 [i31] AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
1566 [i32] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1567 [i33] </Match>
1568 [i34] </AllOf>
1569 [i35] </AnyOf>
1570 [i36] <AnyOf>
1571 [i37] <AllOf>
1572 [i38] <Match
1573 [i39] MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1574 [i40] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1575 [i41] >urn:example:med:schemas:record</AttributeValue>
1576 [i42] <AttributeDesignator
1577 [i43] MustBePresent="false"
1578 [i44] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1579 [i45] AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1580 [i46] DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1581 [i47] </Match>
1582 [i48] <Match
1583 [i49] MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1584 [i50] <AttributeValue
1585 [i51] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1586 [i52] XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1587 [i53] >md:record/md:medical</AttributeValue>
1588 [i54] <AttributeDesignator
1589 [i55] MustBePresent="false"
1590 [i56] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1591 [i57] AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1592 [i58] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1593 [i59] </Match>
1594 [i60] </AllOf>
1595 [i61] </AnyOf>
1596 [i62] <AnyOf>
1597 [i63] <AllOf>
1598 [i64] <Match
1599 [i65] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1600 [i66] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1601 [i67] >read</AttributeValue>
1602 [i68] <AttributeDesignator
1603 [i69] MustBePresent="false"
1604 [i70] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1605 [i71] AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1606 [i72] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1607 [i73] </Match>
1608 [i74] </AllOf>
1609 [i75] <AllOf>
1610 [i76] <Match
1611 [i77] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1612 [i78] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1613 [i79] >write</AttributeValue>
1614 [i80] <AttributeDesignator
1615 [i81] MustBePresent="false"
1616 [i82] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1617 [i83] AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1618 [i84] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1619 [i85] </Match>
1620 [i86] </AllOf>
1621 [i87] </AnyOf>
1622 [i88] </Target>
1623 [i89] </Rule>
1624 [i90] </Policy>

```


1625 [i13] - [i15] The <Rule> element declaration.

1626 [i15] **Rule Effect**. Every **rule** that evaluates to “True” emits the **rule effect** as its value. This **rule**
 1627 **Effect** is “Deny” meaning that according to this **rule**, **access** must be denied when it evaluates to
 1628 “True”.

1629 [i16] - [i20] Free form description of the **rule**.

1630 [i21] - [i88] **Rule target**. The **Rule target** defines the set of **decision requests** that are applicable to the
 1631 **rule**.

1632 [i24] - [i33] The <Match> element targets the **rule** at **subjects** whose
 1633 “urn:oasis:names:tc:xacml:3.0:example:attribute:role” **subject attribute** is equal to “administrator”.

1634 [i36] - [i61] The <AnyOf> element contains one <AllOf> element, which (in turn) contains two <Match>
 1635 elements. The **target** matches if the **resource** identified by the request **context** matches both **resource**
 1636 match criteria.

1637 [i38] - [i47] The first <Match> element targets the **rule** at **resources** whose
 1638 “urn:oasis:names:tc:xacml:2.0:resource:target-namespace” **resource attribute** is equal to
 1639 “urn:example:med:schemas:record”.

1640 [i48] - [i59] The second <Match> element targets the **rule** at XML elements that match the XPath
 1641 expression “/md:record/md:medical”.

1642 [i62] - [i87] The <AnyOf> element contains two <AllOf> elements, each of which contains one <Match>
 1643 element. The **target** matches if the **action** identified in the request **context** matches either of the **action**
 1644 match criteria.

1645 [i64] - [i85] The <Match> elements **target** the **rule** at **actions** whose
 1646 “urn:oasis:names:tc:xacml:1.0:action:action-id” **action attribute** is equal to “read” or “write”.

1647 This **rule** does not have a <Condition> element.

1648 4.2.4.5 Example PolicySet

1649 This section uses the examples of the previous sections to illustrate the process of combining **policies**.
 1650 The **policy** governing read **access** to medical elements of a record is formed from each of the four **rules**
 1651 described in Section 4.2.3. In plain language, the combined **rule** is:

- 1652 • Either the requestor is the patient; or
- 1653 • the requestor is the parent or guardian and the patient is under 16; or
- 1654 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1655 • the requestor is not an administrator.

1656 The following **policy set** illustrates the combined **policies**. **Policy 3** is included by reference and **policy**
 1657 **2** is explicitly included.

```

1658 [j1]    <?xml version="1.0" encoding="UTF-8"?>
1659 [j2]    <PolicySet
1660 [j3]      xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1661 [j4]      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1662 [j5]      PolicySetId="urn:oasis:names:tc:xacml:3.0:example:policysetid:1"
1663 [j6]      Version="1.0"
1664 [j7]      PolicyCombiningAlgId=
1665 [j8]      "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
1666 [j9]    <Description>
1667 [j10]      Example policy set.
1668 [j11]    </Description>
1669 [j12]    <Target>
1670 [j13]      <AnyOf>
1671 [j14]        <AllOf>
1672 [j15]          <Match
1673 [j16]            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1674 [j17]              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1675 [j18]                >urn:example:med:schema:records</AttributeValue>
1676 [j19]            <AttributeDesignator
1677 [j20]              MustBePresent="false"
  
```

```

1678 [j21]         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1679 [j22]         AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1680 [j23]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1681 [j24]         </Match>
1682 [j25]         </AllOf>
1683 [j26]         </AnyOf>
1684 [j27]         </Target>
1685 [j28]         <PolicyIdReference>
1686 [j29]           urn:oasis:names:tc:xacml:3.0:example:policyid:3
1687 [j30]         </PolicyIdReference>
1688 [j31]         <Policy
1689 [j32]           PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"
1690 [j33]           RuleCombiningAlgId=
1691 [j34]             "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
1692 [j35]           Version="1.0">
1693 [j36]           <Target/>
1694 [j37]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"
1695 [j38]             Effect="Permit">
1696 [j39]           </Rule>
1697 [j40]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
1698 [j41]             Effect="Permit">
1699 [j42]           </Rule>
1700 [j43]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:4"
1701 [j44]             Effect="Deny">
1702 [j45]           </Rule>
1703 [j46]         </Policy>
1704 [j47]         </PolicySet>

```

1705 [j2] - [j8] The <PolicySet> element declaration. Standard XML namespace declarations are included.

1706 [j5] The PolicySetId attribute is used for identifying this **policy set** for possible inclusion in another
1707 **policy set**.

1708 [j7] - [j8] The **policy-combining algorithm** identifier. **Policies** and **policy sets** in this **policy set** are
1709 combined according to the specified **policy-combining algorithm** when the **authorization decision** is
1710 computed.

1711 [j9] - [j11] Free form description of the **policy set**.

1712 [j12] - [j27] The **policy set** <Target> element defines the set of **decision requests** that are applicable to
1713 this <PolicySet> element.

1714 [j28] - [j30] PolicyIdReference includes a **policy** by id.

1715 [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 7.12 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 7.12.

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 7.16.

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 7.16.

```
<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>
```

```
1765 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>
1766 <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
1767 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>
1768 <xs:attribute name="MaxDelegationDepth" type="xs:integer" use="optional"/>
1769 </xs:complexType>
```

1770 The <PolicySet> element is of PolicySetType complex type.

1771 The <PolicySet> element contains the following attributes and elements:

1772 PolicySetId [Required]

1773 **Policy set** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to
1774 the **PD**P have the same identifier. This MAY be achieved by following a predefined URN or URI
1775 scheme. If the **policy set** identifier is in the form of a URL, then it MAY be resolvable.

1776 Version [Required]

1777 The version number of the PolicySet.

1778 PolicyCombiningAlgId [Required]

1779 The identifier of the **policy-combining algorithm** by which the <PolicySet>,
1780 <CombinerParameters>, <PolicyCombinerParameters> and
1781 <PolicySetCombinerParameters> components MUST be combined. Standard **policy-**
1782 **combining algorithms** are listed in Appendix C. Standard **policy-combining algorithm**
1783 identifiers are listed in Section B.9.

1784 MaxDelegationDepth [Optional]

1785 If present, limits the depth of delegation which is authorized by this **policy set**. See the delegation
1786 profile [**XACMLAdmin**].

1787 <Description> [Optional]

1788 A free-form description of the **policy set**.

1789 <PolicyIssuer> [Optional]

1790 **Attributes** of the **issuer** of the **policy set**.

1791 <PolicySetDefaults> [Optional]

1792 A set of default values applicable to the **policy set**. The scope of the <PolicySetDefaults>
1793 element SHALL be the enclosing **policy set**.

1794 <Target> [Required]

1795 The <Target> element defines the applicability of a **policy set** to a set of **decision requests**.

1796 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be computed
1797 from the <Target> elements of the referenced <Policy> elements, either as an intersection or
1798 as a union.

1799 <PolicySet> [Any Number]

1800 A **policy set** that is included in this **policy set**.

1801 <Policy> [Any Number]

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

1803 <PolicySetIdReference> [Any Number]

1804 A reference to a **policy set** that MUST be included in this **policy set**. If
1805 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1806 <PolicyIdReference> [Any Number]

1807 A reference to a **policy** that MUST be included in this **policy set**. If the
1808 <PolicyIdReference> is a URL, then it MAY be resolvable.

- 1809 <ObligationExpressions> [Optional]
 1810 Contains the set of <ObligationExpression> elements. See Section 7.16 for a description of
 1811 how the set of **obligations** to be returned by the **PDP** shall be determined.
- 1812 <AdviceExpressions> [Optional]
 1813 Contains the set of <AdviceExpression> elements. See Section 7.16 for a description of how
 1814 the set of **advice** to be returned by the **PDP** shall be determined.
- 1815 <CombinerParameters> [Optional]
 1816 Contains a sequence of <CombinerParameter> elements.
- 1817 <PolicyCombinerParameters> [Optional]
 1818 Contains a sequence of <CombinerParameter> elements that are associated with a particular
 1819 <Policy> or <PolicyIdReference> element within the <PolicySet>.
- 1820 <PolicySetCombinerParameters> [Optional]
 1821 Contains a sequence of <CombinerParameter> elements that are associated with a particular
 1822 <PolicySet> or <PolicySetIdReference> element within the <PolicySet>.

1823 5.2 Element <Description>

- 1824 The <Description> element contains a free-form description of the <PolicySet>, <Policy>,
 1825 <Rule> or <Apply> element. The <Description> element is of xs:string simple type.

1826

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

1827 5.3 Element <PolicyIssuer>

- 1828 The <PolicyIssuer> element contains **attributes** describing the issuer of the **policy** or **policy set**.
 1829 The use of the **policy** issuer element is defined in a separate administration profile [**XACMLAdmin**]. A
 1830 PDP which does not implement the administration profile **MUST** report an error or return an indeterminate
 1831 result if it encounters this element.

1832

```
<xs:element name="PolicyIssuer" type="xacml:PolicyIssuerType"/>  

  1833 <xs:complexType name="PolicyIssuerType">  

  1834   <xs:sequence>  

  1835     <xs:element ref="xacml:Content" minOccurs="0"/>  

  1836     <xs:element ref="xacml:Attribute" minOccurs="0" maxOccurs="unbounded"/>  

  1837   </xs:sequence>  

  1838 </xs:complexType>
```

- 1839 The <PolicyIssuer> element is of PolicyIssuerType complex type.

- 1840 The <PolicyIssuer> element contains the following elements:

- 1841 <Content> [Optional]
 1842 Free form XML describing the issuer. See Section 5.45.
- 1843 <Attribute> [Zero to many]
 1844 An **attribute** of the issuer. See Section 5.46.

1845 5.4 Element <PolicySetDefaults>

- 1846 The <PolicySetDefaults> element **SHALL** specify default values that apply to the <PolicySet>
 1847 element.

1848

```
<xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>  

  1849 <xs:complexType name="DefaultsType">  

  1850   <xs:sequence>
```

1851
1852
1853
1854
1855

```
<xs:choice>
  <xs:element ref="xacml:XPathVersion">
</xs:choice>
</xs:sequence>
</xs:complexType>
```

1856 <PolicySetDefaults> element is of DefaultsType complex type.

1857 The <PolicySetDefaults> element contains the following elements:

1858 <XPathVersion> [Optional]

1859 Default XPath version.

1860 5.5 Element <XPathVersion>

1861 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by
1862 <AttributeSelector> elements and XPath-based functions in the **policy set** or **policy**.

1863

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

1864 The URI for the XPath 1.0 specification is "http://www.w3.org/TR/1999/REC-xpath-19991116".

1865 The URI for the XPath 2.0 specification is "http://www.w3.org/TR/2007/REC-xpath20-20070123".

1866 The <XPathVersion> element is REQUIRED if the XACML enclosing **policy set** or **policy** contains
1867 <AttributeSelector> elements or XPath-based functions.

1868 5.6 Element <Target>

1869 The <Target> element identifies the set of **decision requests** that the parent element is intended to
1870 evaluate. The <Target> element SHALL appear as a child of a <PolicySet> and <Policy> element
1871 and MAY appear as a child of a <Rule> element.

1872 The <Target> element SHALL contain a **conjunctive sequence** of <AnyOf> elements. For the parent
1873 of the <Target> element to be applicable to the **decision request**, there MUST be at least one positive
1874 match between each <AnyOf> element of the <Target> element and the corresponding section of the
1875 <Request> element.

1876
1877
1878
1879
1880
1881

```
<xs:element name="Target" type="xacml:TargetType"/>
<xs:complexType name="TargetType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="xacml:AnyOf"/>
  </xs:sequence>
</xs:complexType>
```

1882 The <Target> element is of TargetType complex type.

1883 The <Target> element contains the following elements:

1884 <AnyOf> [Zero to Many]

1885 Matching specification for **attributes** in the **context**. If this element is missing, then the **target**
1886 SHALL match all **contexts**.

1887 5.7 Element <AnyOf>

1888 The <AnyOf> element SHALL contain a **disjunctive sequence** of <AllOf> elements.

1889
1890
1891
1892
1893
1894

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

1895 The <AnyOf> element is of AnyOfType complex type.

1896 The <AnyOf> element contains the following elements:

1897 <AllOf> [One to Many, Required]

1898 See Section 5.8.

1899 5.8 Element <AllOf>

1900 The <AllOf> element SHALL contain a **conjunctive sequence** of <Match> elements.

```
1901 <xs:element name="AllOf" type="xacml:AllOfType"/>
1902 <xs:complexType name="AllOfType">
1903   <xs:sequence minOccurs="1" maxOccurs="unbounded">
1904     <xs:element ref="xacml:Match"/>
1905   </xs:sequence>
1906 </xs:complexType>
```

1907 The <AllOf> element is of AllOfType complex type.

1908 The <AllOf> element contains the following elements:

1909 <Match> [One to Many]

1910 A **conjunctive sequence** of individual matches of the **attributes** in the request **context** and the
1911 embedded **attribute** values. See Section 5.9.

1912 5.9 Element <Match>

1913 The <Match> element SHALL identify a set of entities by matching **attribute** values in an
1914 <Attributes> element of the request **context** with the embedded **attribute** value.

```
1915 <xs:element name="Match" type="xacml:MatchType"/>
1916 <xs:complexType name="MatchType">
1917   <xs:sequence>
1918     <xs:element ref="xacml:AttributeValue"/>
1919     <xs:choice>
1920       <xs:element ref="xacml:AttributeDesignator"/>
1921       <xs:element ref="xacml:AttributeSelector"/>
1922     </xs:choice>
1923   </xs:sequence>
1924   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
1925 </xs:complexType>
```

1926 The <Match> element is of MatchType complex type.

1927 The <Match> element contains the following attributes and elements:

1928 MatchId [Required]

1929 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI with legal
1930 values documented in Section 7.6.

1931 <AttributeValue> [Required]

1932 Embedded **attribute** value.

1933 <AttributeDesignator> [Required choice]

1934 MAY be used to identify one or more **attribute** values in an <Attributes> element of the
1935 request **context**.

1936 <AttributeSelector> [Required choice]

1937 MAY be used to identify one or more **attribute** values in a <Content> element of the request
1938 **context**.

1939 5.10 Element <PolicySetIdReference>

1940 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element by id.
1941 If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet> element.
1942 However, the mechanism for resolving a **policy set** reference to the corresponding **policy set** is outside
1943 the scope of this specification.

```
1944 <xs:element name="PolicySetIdReference" type="xacml:IdReferenceType"/>
1945 <xs:complexType name="IdReferenceType">
1946   <xs:simpleContent>
1947     <xs:extension base="xs:anyURI">
1948       <xs:attribute name="xacml:Version"
1949         type="xacml:VersionMatchType" use="optional"/>
1950       <xs:attribute name="xacml:EarliestVersion"
1951         type="xacml:VersionMatchType" use="optional"/>
1952       <xs:attribute name="xacml:LatestVersion"
1953         type="xacml:VersionMatchType" use="optional"/>
1954     </xs:extension>
1955   </xs:simpleContent>
1956 </xs:complexType>
```

1957 Element <PolicySetIdReference> is of `xacml:IdReferenceType` complex type.

1958 `IdReferenceType` extends the `xs:anyURI` type with the following attributes:

1959 **Version** [Optional]

1960 Specifies a matching expression for the version of the **policy set** referenced.

1961 **EarliestVersion** [Optional]

1962 Specifies a matching expression for the earliest acceptable version of the **policy set** referenced.

1963 **LatestVersion** [Optional]

1964 Specifies a matching expression for the latest acceptable version of the **policy set** referenced.

1965 The matching operation is defined in Section 5.13. Any combination of these attributes MAY be present
1966 in a <PolicySetIdReference>. The referenced **policy set** MUST match all expressions. If none of
1967 these attributes is present, then any version of the **policy set** is acceptable. In the case that more than
1968 one matching version can be obtained, then the most recent one SHOULD be used.

1969 5.11 Element <PolicyIdReference>

1970 The <PolicyIdReference> element SHALL be used to reference a <Policy> element by id. If
1971 <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy> element. However, the
1972 mechanism for resolving a **policy** reference to the corresponding **policy** is outside the scope of this
1973 specification.

```
1974 <xs:element name="PolicyIdReference" type="xacml:IdReferenceType"/>
```

1975 Element <PolicyIdReference> is of `xacml:IdReferenceType` complex type (see Section 5.10) .

1976 5.12 Simple type VersionType

1977 Elements of this type SHALL contain the version number of the **policy** or **policy set**.

```
1978 <xs:simpleType name="VersionType">
1979   <xs:restriction base="xs:string">
1980     <xs:pattern value="(\d+\.)*\d+"/>
1981   </xs:restriction>
1982 </xs:simpleType>
```

1983 The version number is expressed as a sequence of decimal numbers, each separated by a period (.).
1984 'd+' represents a sequence of one or more decimal digits.

1985 5.13 Simple type VersionMatchType

1986 Elements of this type SHALL contain a restricted regular expression matching a version number (see
1987 Section 5.12). The expression SHALL match versions of a referenced **policy** or **policy set** that are
1988 acceptable for inclusion in the referencing **policy** or **policy set**.

```
1989 <xs:simpleType name="VersionMatchType">  
1990   <xs:restriction base="xs:string">  
1991     <xs:pattern value="((\d+|\*)\.)*(\d+|\*|\+)" />  
1992   </xs:restriction>  
1993 </xs:simpleType>
```

1994 A version match is '.'-separated, like a version string. A number represents a direct numeric match. A '*'
1995 means that any single number is valid. A '+' means that any number, and any subsequent numbers, are
1996 valid. In this manner, the following four patterns would all match the version string '1.2.3': '1.2.3', '1.*.3',
1997 '1.2.*' and '1.+'

1998 5.14 Element <Policy>

1999 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2000 A <Policy> element may be evaluated, in which case the evaluation procedure defined in Section 7.11
2001 SHALL be used.

2002 The main components of this element are the <Target>, <Rule>, <CombinerParameters>,
2003 <RuleCombinerParameters>, <ObligationExpressions> and <AdviceExpressions>
2004 elements and the RuleCombiningAlgId attribute.

2005 A <Policy> element MAY contain a <PolicyIssuer> element. The interpretation of the
2006 <PolicyIssuer> element is explained in the separate administrative **policy** profile [XACMLAdmin].

2007 The <Target> element defines the applicability of the <Policy> element to a set of **decision requests**.
2008 If the <Target> element within the <Policy> element matches the request **context**, then the
2009 <Policy> element MAY be used by the **PDP** in making its **authorization decision**. See Section 7.11.

2010 The <Policy> element includes a sequence of choices between <VariableDefinition> and
2011 <Rule> elements.

2012 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the
2013 RuleCombiningAlgId attribute.

2014 The <ObligationExpressions> element contains a set of **obligation** expressions that MUST be
2015 evaluated into **obligations** by the **PDP** and the resulting **obligations** MUST be fulfilled by the **PEP** in
2016 conjunction with the **authorization decision**. If the **PEP** does not understand, or cannot fulfill, any of the
2017 **obligations**, then it MUST act according to the PEP bias. See Section 7.2 and 7.16.

2018 The <AdviceExpressions> element contains a set of **advice** expressions that MUST be evaluated into
2019 **advice** by the **PDP**. The resulting **advice** MAY be safely ignored by the **PEP** in conjunction with the
2020 **authorization decision**. See Section 7.16.

```
2021 <xs:element name="Policy" type="xacml:PolicyType"/>  
2022 <xs:complexType name="PolicyType">  
2023   <xs:sequence>  
2024     <xs:element ref="xacml:Description" minOccurs="0"/>  
2025     <xs:element ref="xacml:PolicyIssuer" minOccurs="0"/>  
2026     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>  
2027     <xs:element ref="xacml:Target"/>  
2028     <xs:choice maxOccurs="unbounded">  
2029       <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>  
2030       <xs:element ref="xacml:RuleCombinerParameters" minOccurs="0"/>  
2031       <xs:element ref="xacml:VariableDefinition"/>  
2032       <xs:element ref="xacml:Rule"/>  
2033     </xs:choice>  
2034     <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
```

```

2035     <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
2036 </xs:sequence>
2037 <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2038 <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
2039 <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2040 <xs:attribute name="MaxDelegationDepth" type="xs:integer" use="optional"/>
2041 </xs:complexType>

```

2042 The <Policy> element is of PolicyType complex type.

2043 The <Policy> element contains the following attributes and elements:

2044 PolicyId [Required]

2045 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to the
2046 **PDP** have the same identifier. This MAY be achieved by following a predefined URN or URI
2047 scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2048 Version [Required]

2049 The version number of the **Policy**.

2050 RuleCombiningAlgId [Required]

2051 The identifier of the **rule-combining algorithm** by which the <Policy>,
2052 <CombinerParameters> and <RuleCombinerParameters> components MUST be
2053 combined. Standard **rule-combining algorithms** are listed in Appendix C. Standard **rule-**
2054 **combining algorithm** identifiers are listed in Section B.9.

2055 MaxDelegationDepth [Optional]

2056 If present, limits the depth of delegation which is authorized by this **policy**. See the delegation
2057 profile [XACMLAdmin].

2058 <Description> [Optional]

2059 A free-form description of the **policy**. See Section 5.2.

2060 <PolicyIssuer> [Optional]

2061 **Attributes** of the **issuer** of the **policy**.

2062 <PolicyDefaults> [Optional]

2063 Defines a set of default values applicable to the **policy**. The scope of the <PolicyDefaults>
2064 element SHALL be the enclosing **policy**.

2065 <CombinerParameters> [Optional]

2066 A sequence of parameters to be used by the **rule-combining algorithm**.

2067 <RuleCombinerParameters> [Optional]

2068 A sequence of parameters to be used by the **rule-combining algorithm**.

2069 <Target> [Required]

2070 The <Target> element defines the applicability of a <Policy> to a set of **decision requests**.

2071 The <Target> element MAY be declared by the creator of the <Policy> element, or it MAY be
2072 computed from the <Target> elements of the referenced <Rule> elements either as an
2073 intersection or as a union.

2074 <VariableDefinition> [Any Number]

2075 Common function definitions that can be referenced from anywhere in a **rule** where an
2076 expression can be found.

2077 <Rule> [Any Number]

2078 A sequence of **rules** that MUST be combined according to the `RuleCombiningAlgId` attribute.
2079 **Rules** whose `<Target>` elements and conditions match the **decision request** MUST be
2080 considered. **Rules** whose `<Target>` elements or conditions do not match the **decision request**
2081 SHALL be ignored.

2082 `<ObligationExpressions>` [Optional]

2083 A **conjunctive sequence** of **obligation** expressions which MUST be evaluated into **obligations**
2084 by the PDP. The corresponding **obligations** MUST be fulfilled by the **PEP** in conjunction with
2085 the **authorization decision**. See Section 7.16 for a description of how the set of **obligations** to
2086 be returned by the **PDP** SHALL be determined. See section 7.2 about enforcement of
2087 **obligations**.

2088 `<AdviceExpressions>` [Optional]

2089 A **conjunctive sequence** of **advice** expressions which MUST be evaluated into **advice** by the **PDP**.
2090 The corresponding **advice** provide supplementary information to the **PEP** in conjunction with the
2091 **authorization decision**. See Section 7.16 for a description of how the set of **advice** to be
2092 returned by the **PDP** SHALL be determined.

2093 5.15 Element `<PolicyDefaults>`

2094 The `<PolicyDefaults>` element SHALL specify default values that apply to the `<Policy>` element.

```
2095 <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>  
2096 <xs:complexType name="DefaultsType">  
2097   <xs:sequence>  
2098     <xs:choice>  
2099       <xs:element ref="xacml:XPathVersion" />  
2100     </xs:choice>  
2101   </xs:sequence>  
2102 </xs:complexType>
```

2103 `<PolicyDefaults>` element is of `DefaultsType` complex type.

2104 The `<PolicyDefaults>` element contains the following elements:

2105 `<XPathVersion>` [Optional]

2106 Default XPath version.

2107 5.16 Element `<CombinerParameters>`

2108 The `<CombinerParameters>` element conveys parameters for a **policy-** or **rule-combining algorithm**.

2109 If multiple `<CombinerParameters>` elements occur within the same **policy** or **policy set**, they SHALL
2110 be considered equal to one `<CombinerParameters>` element containing the concatenation of all the
2111 sequences of `<CombinerParameters>` contained in all the aforementioned `<CombinerParameters>`
2112 elements, such that the order of occurrence of the `<CombinerParameters>` elements is preserved in the
2113 concatenation of the `<CombinerParameter>` elements.

2114 Note that none of the combining algorithms specified in XACML 3.0 is parameterized.

```
2115 <xs:element name="CombinerParameters" type="xacml:CombinerParametersType"/>  
2116 <xs:complexType name="CombinerParametersType">  
2117   <xs:sequence>  
2118     <xs:element ref="xacml:CombinerParameter" minOccurs="0"  
2119       maxOccurs="unbounded"/>  
2120   </xs:sequence>  
2121 </xs:complexType>
```

2122 The `<CombinerParameters>` element is of `CombinerParametersType` complex type.

2123 The `<CombinerParameters>` element contains the following elements:

- 2124 <CombinerParameter> [Any Number]
2125 A single parameter. See Section 5.17.
2126 Support for the <CombinerParameters> element is optional.

2127 5.17 Element <CombinerParameter>

2128 The <CombinerParameter> element conveys a single parameter for a *policy*- or *rule-combining*
2129 *algorithm*.

```
2130 <xs:element name="CombinerParameter" type="xacml:CombinerParameterType"/>  
2131 <xs:complexType name="CombinerParameterType">  
2132   <xs:sequence>  
2133     <xs:element ref="xacml:AttributeValue"/>  
2134   </xs:sequence>  
2135   <xs:attribute name="ParameterName" type="xs:string" use="required"/>  
2136 </xs:complexType>
```

2137 The <CombinerParameter> element is of CombinerParameterType complex type.

2138 The <CombinerParameter> element contains the following attributes:

2139 ParameterName [Required]

2140 The identifier of the parameter.

2141 <AttributeValue> [Required]

2142 The value of the parameter.

2143 Support for the <CombinerParameter> element is optional.

2144 5.18 Element <RuleCombinerParameters>

2145 The <RuleCombinerParameters> element conveys parameters associated with a particular *rule*
2146 within a *policy* for a *rule-combining algorithm*.

2147 Each <RuleCombinerParameters> element MUST be associated with a *rule* contained within the
2148 same *policy*. If multiple <RuleCombinerParameters> elements reference the same *rule*, they SHALL
2149 be considered equal to one <RuleCombinerParameters> element containing the concatenation of all
2150 the sequences of <CombinerParameters> contained in all the aforementioned
2151 <RuleCombinerParameters> elements, such that the order of occurrence of the
2152 <RuleCominberParameters> elements is preserved in the concatenation of the
2153 <CombinerParameter> elements.

2154 Note that none of the *rule-combining algorithms* specified in XACML 3.0 is parameterized.

```
2155 <xs:element name="RuleCombinerParameters"  
2156   type="xacml:RuleCombinerParametersType"/>  
2157 <xs:complexType name="RuleCombinerParametersType">  
2158   <xs:complexContent>  
2159     <xs:extension base="xacml:CombinerParametersType">  
2160       <xs:attribute name="RuleIdRef" type="xs:string"  
2161         use="required"/>  
2162     </xs:extension>  
2163   </xs:complexContent>  
2164 </xs:complexType>
```

2165 The <RuleCombinerParameters> element contains the following attribute:

2166 RuleIdRef [Required]

2167 The identifier of the <Rule> contained in the *policy*.

2168 Support for the <RuleCombinerParameters> element is optional, only if support for combiner
2169 parameters is not implemented.

2170 5.19 Element <PolicyCombinerParameters>

2171 The <PolicyCombinerParameters> element conveys parameters associated with a particular *policy*
2172 within a *policy set* for a *policy-combining algorithm*.

2173 Each <PolicyCombinerParameters> element MUST be associated with a *policy* contained within the
2174 same *policy set*. If multiple <PolicyCombinerParameters> elements reference the same *policy*,
2175 they SHALL be considered equal to one <PolicyCombinerParameters> element containing the
2176 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned
2177 <PolicyCombinerParameters> elements, such that the order of occurrence of the
2178 <PolicyCombinerParameters> elements is preserved in the concatenation of the
2179 <CombinerParameter> elements.

2180 Note that none of the *policy-combining algorithms* specified in XACML 3.0 is parameterized.

```
2181 <xs:element name="PolicyCombinerParameters"  
2182 type="xacml:PolicyCombinerParametersType"/>  
2183 <xs:complexType name="PolicyCombinerParametersType">  
2184 <xs:complexContent>  
2185 <xs:extension base="xacml:CombinerParametersType">  
2186 <xs:attribute name="PolicyIdRef" type="xs:anyURI"  
2187 use="required"/>  
2188 </xs:extension>  
2189 </xs:complexContent>  
2190 </xs:complexType>
```

2191 The <PolicyCombinerParameters> element is of PolicyCombinerParametersType complex
2192 type.

2193 The <PolicyCombinerParameters> element contains the following attribute:

2194 PolicyIdRef [Required]

2195 The identifier of a <Policy> or the value of a <PolicyIdReference> contained in the *policy*
2196 *set*.

2197 Support for the <PolicyCombinerParameters> element is optional, only if support for combiner
2198 parameters is not implemented.

2199 5.20 Element <PolicySetCombinerParameters>

2200 The <PolicySetCombinerParameters> element conveys parameters associated with a particular
2201 *policy set* within a *policy set* for a *policy-combining algorithm*.

2202 Each <PolicySetCombinerParameters> element MUST be associated with a *policy set* contained
2203 within the same *policy set*. If multiple <PolicySetCombinerParameters> elements reference the
2204 same *policy set*, they SHALL be considered equal to one <PolicySetCombinerParameters>
2205 element containing the concatenation of all the sequences of <CombinerParameters> contained in all
2206 the aforementioned <PolicySetCombinerParameters> elements, such that the order of occurrence
2207 of the <PolicySetCombinerParameters> elements is preserved in the concatenation of the
2208 <CombinerParameter> elements.

2209 Note that none of the *policy-combining algorithms* specified in XACML 3.0 is parameterized.

```
2210 <xs:element name="PolicySetCombinerParameters"  
2211 type="xacml:PolicySetCombinerParametersType"/>  
2212 <xs:complexType name="PolicySetCombinerParametersType">  
2213 <xs:complexContent>  
2214 <xs:extension base="xacml:CombinerParametersType">  
2215 <xs:attribute name="PolicySetIdRef" type="xs:anyURI"  
2216 use="required"/>  
2217 </xs:extension>  
2218 </xs:complexContent>  
2219 </xs:complexType>
```

2220 The <PolicySetCombinerParameters> element is of PolicySetCombinerParametersType
 2221 complex type.
 2222 The <PolicySetCombinerParameters> element contains the following attribute:
 2223 PolicySetIdRef [Required]
 2224 The identifier of a <PolicySet> or the value of a <PolicySetIdReference> contained in the
 2225 *policy set*.
 2226 Support for the <PolicySetCombinerParameters> element is optional, only if support for combiner
 2227 parameters is not implemented.

2228 5.21 Element <Rule>

2229 The <Rule> element SHALL define the individual *rules* in the *policy*. The main components of this
 2230 element are the <Target>, <Condition>, <ObligationExpressions> and
 2231 <AdviceExpressions> elements and the Effect attribute.
 2232 A <Rule> element may be evaluated, in which case the evaluation procedure defined in Section 7.10
 2233 SHALL be used.

```

2234 <xs:element name="Rule" type="xacml:RuleType"/>
2235 <xs:complexType name="RuleType">
2236   <xs:sequence>
2237     <xs:element ref="xacml:Description" minOccurs="0"/>
2238     <xs:element ref="xacml:Target" minOccurs="0"/>
2239     <xs:element ref="xacml:Condition" minOccurs="0"/>
2240     <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
2241     <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
2242   </xs:sequence>
2243   <xs:attribute name="RuleId" type="xs:string" use="required"/>
2244   <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>
2245 </xs:complexType>
  
```

2246 The <Rule> element is of RuleType complex type.
 2247 The <Rule> element contains the following attributes and elements:
 2248 RuleId [Required]
 2249 A string identifying this *rule*.
 2250 Effect [Required]
 2251 **Rule effect.** The value of this attribute is either “Permit” or “Deny”.
 2252 <Description> [Optional]
 2253 A free-form description of the *rule*.
 2254 <Target> [Optional]
 2255 Identifies the set of *decision requests* that the <Rule> element is intended to evaluate. If this
 2256 element is omitted, then the *target* for the <Rule> SHALL be defined by the <Target> element
 2257 of the enclosing <Policy> element. See Section 7.7 for details.
 2258 <Condition> [Optional]
 2259 A *predicate* that MUST be satisfied for the *rule* to be assigned its Effect value.
 2260 <ObligationExpressions> [Optional]
 2261 A *conjunctive sequence* of *obligation* expressions which MUST be evaluated into *obligations*
 2262 by the PDP. The corresponding *obligations* MUST be fulfilled by the *PEP* in conjunction with
 2263 the *authorization decision*. See Section 7.16 for a description of how the set of *obligations* to
 2264 be returned by the *PDP* SHALL be determined. See section 7.2 about enforcement of
 2265 *obligations*.

2266 <AdviceExpressions> [Optional]
2267 A **conjunctive sequence** of **advice** expressions which MUST be evaluated into **advice** by the **PDP**.
2268 The corresponding **advice** provide supplementary information to the **PEP** in conjunction with the
2269 **authorization decision**. See Section 7.16 for a description of how the set of **advice** to be
2270 returned by the **PDP** SHALL be determined.

2271 5.22 Simple type EffectType

2272 The EffectType simple type defines the values allowed for the Effect attribute of the <Rule> element
2273 and for the FulfillOn attribute of the <ObligationExpression> and <AdviceExpression>
2274 elements.

```
2275 <xs:simpleType name="EffectType">  
2276   <xs:restriction base="xs:string">  
2277     <xs:enumeration value="Permit"/>  
2278     <xs:enumeration value="Deny"/>  
2279   </xs:restriction>  
2280 </xs:simpleType>
```

2281 5.23 Element <VariableDefinition>

2282 The <VariableDefinition> element SHALL be used to define a value that can be referenced by a
2283 <VariableReference> element. The name supplied for its VariableId attribute SHALL NOT occur
2284 in the VariableId attribute of any other <VariableDefinition> element within the encompassing
2285 **policy**. The <VariableDefinition> element MAY contain undefined <VariableReference>
2286 elements, but if it does, a corresponding <VariableDefinition> element MUST be defined later in
2287 the encompassing **policy**. <VariableDefinition> elements MAY be grouped together or MAY be
2288 placed close to the reference in the encompassing **policy**. There MAY be zero or more references to
2289 each <VariableDefinition> element.

```
2290 <xs:element name="VariableDefinition" type="xacml:VariableDefinitionType"/>  
2291 <xs:complexType name="VariableDefinitionType">  
2292   <xs:sequence>  
2293     <xs:element ref="xacml:Expression"/>  
2294   </xs:sequence>  
2295   <xs:attribute name="VariableId" type="xs:string" use="required"/>  
2296 </xs:complexType>
```

2297 The <VariableDefinition> element is of VariableDefinitionType complex type. The
2298 <VariableDefinition> element has the following elements and attributes:

2299 <Expression> [Required]
2300 Any element of ExpressionType complex type.
2301 VariableId [Required]
2302 The name of the variable definition.

2303 5.24 Element <VariableReference>

2304 The <VariableReference> element is used to reference a value defined within the same
2305 encompassing <Policy> element. The <VariableReference> element SHALL refer to the
2306 <VariableDefinition> element by string equality on the value of their respective VariableId
2307 attributes. One and only one <VariableDefinition> MUST exist within the same encompassing
2308 <Policy> element to which the <VariableReference> refers. There MAY be zero or more
2309 <VariableReference> elements that refer to the same <VariableDefinition> element.

```
2310 <xs:element name="VariableReference" type="xacml:VariableReferenceType"  
2311   substitutionGroup="xacml:Expression"/>  
2312 <xs:complexType name="VariableReferenceType">
```

```

2313     <xs:complexContent>
2314         <xs:extension base="xacml:ExpressionType">
2315             <xs:attribute name="VariableId" type="xs:string"
2316                 use="required"/>
2317         </xs:extension>
2318     </xs:complexContent>
2319 </xs:complexType>

```

2320 The <VariableReference> element is of the VariableReferenceType complex type, which is of
 2321 the ExpressionType complex type and is a member of the <Expression> element substitution group.
 2322 The <VariableReference> element MAY appear any place where an <Expression> element occurs
 2323 in the schema.

2324 The <VariableReference> element has the following attribute:

2325 VariableId [Required]

2326 The name used to refer to the value defined in a <VariableDefinition> element.

2327 5.25 Element <Expression>

2328 The <Expression> element is not used directly in a **policy**. The <Expression> element signifies that
 2329 an element that extends the ExpressionType and is a member of the <Expression> element
 2330 substitution group SHALL appear in its place.

```

2331 <xs:element name="Expression" type="xacml:ExpressionType" abstract="true"/>
2332 <xs:complexType name="ExpressionType" abstract="true"/>

```

2333 The following elements are in the <Expression> element substitution group:

2334 <Apply>, <AttributeSelector>, <AttributeValue>, <Function>, <VariableReference> and
 2335 <AttributeDesignator>.

2336 5.26 Element <Condition>

2337 The <Condition> element is a Boolean function over **attributes** or functions of **attributes**.

```

2338 <xs:element name="Condition" type="xacml:ConditionType"/>
2339 <xs:complexType name="ConditionType">
2340     <xs:sequence>
2341         <xs:element ref="xacml:Expression"/>
2342     </xs:sequence>
2343 </xs:complexType>

```

2344 The <Condition> contains one <Expression> element, with the restriction that the <Expression>
 2345 return data-type MUST be "http://www.w3.org/2001/XMLSchema#boolean". Evaluation of the
 2346 <Condition> element is described in Section 7.9.

2347 5.27 Element <Apply>

2348 The <Apply> element denotes application of a function to its arguments, thus encoding a function call.

2349 The <Apply> element can be applied to any combination of the members of the <Expression>
 2350 element substitution group. See Section 5.25.

```

2351 <xs:element name="Apply" type="xacml:ApplyType"
2352     substitutionGroup="xacml:Expression"/>
2353 <xs:complexType name="ApplyType">
2354     <xs:complexContent>
2355         <xs:extension base="xacml:ExpressionType">
2356             <xs:sequence>
2357                 <xs:element ref="xacml:Description" minOccurs="0"/>
2358                 <xs:element ref="xacml:Expression" minOccurs="0"
2359                     maxOccurs="unbounded"/>

```



```

2360         </xs:sequence>
2361         <xs:attribute name="FunctionId" type="xs:anyURI"
2362             use="required"/>
2363     </xs:extension>
2364 </xs:complexContent>
2365 </xs:complexType>

```

2366 The <Apply> element is of ApplyType complex type.

2367 The <Apply> element contains the following attributes and elements:

2368 FunctionId [Required]

2369 The identifier of the function to be applied to the arguments. XACML-defined functions are
2370 described in Appendix A.3.

2371 <Description> [Optional]

2372 A free-form description of the <Apply> element.

2373 <Expression> [Optional]

2374 Arguments to the function, which may include other functions.

2375 5.28 Element <Function>

2376 The <Function> element SHALL be used to name a function as an argument to the function defined by
2377 the parent <Apply> element.

```

2378 <xs:element name="Function" type="xacml:FunctionType"
2379 substitutionGroup="xacml:Expression"/>
2380 <xs:complexType name="FunctionType">
2381 <xs:complexContent>
2382 <xs:extension base="xacml:ExpressionType">
2383 <xs:attribute name="FunctionId" type="xs:anyURI"
2384 use="required"/>
2385 </xs:extension>
2386 </xs:complexContent>
2387 </xs:complexType>

```

2388 The <Function> element is of FunctionType complex type.

2389 The <Function> element contains the following attribute:

2390 FunctionId [Required]

2391 The identifier of the function.

2392 5.29 Element <AttributeDesignator>

2393 The <AttributeDesignator> element retrieves a **bag** of values for a **named attribute** from the
2394 request **context**. A **named attribute** SHALL be considered present if there is at least one **attribute** that
2395 matches the criteria set out below.

2396 The <AttributeDesignator> element SHALL return a **bag** containing all the **attribute** values that are
2397 matched by the **named attribute**. In the event that no matching **attribute** is present in the **context**, the
2398 MustBePresent attribute governs whether this element returns an empty **bag** or "Indeterminate". See
2399 Section 7.3.5.

2400 The <AttributeDesignator> MAY appear in the <Match> element and MAY be passed to the
2401 <Apply> element as an argument.

2402 The <AttributeDesignator> element is of the AttributeDesignatorType complex type.

```

2403 <xs:complexType name="AttributeDesignatorType">
2404 <xs:complexContent>
2405 <xs:extension base="xacml:ExpressionType">

```

```

2406     <xs:attribute name="Category" type="xs:anyURI"
2407         use="required"/>
2408     <xs:attribute name="AttributeId" type="xs:anyURI"
2409         use="required"/>
2410     <xs:attribute name="DataType" type="xs:anyURI"
2411         use="required"/>
2412     <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2413     <xs:attribute name="MustBePresent" type="xs:boolean"
2414         use="required"/>
2415     </xs:extension>
2416 </xs:complexContent>
2417 </xs:complexType>

```

2418 A **named attribute** SHALL match an **attribute** if the values of their respective `Category`,
2419 `AttributeId`, `DataType` and `Issuer` attributes match. The attribute designator's `Category` MUST
2420 match, by URI equality, the `Category` of the `<Attributes>` element in which the **attribute** is present.
2421 The attribute designator's `AttributeId` MUST match, by URI equality, the `AttributeId` of the
2422 attribute. The attribute designator's `DataType` MUST match, by URI equality, the `DataType` of the same
2423 **attribute**.

2424 If the `Issuer` attribute is present in the attribute designator, then it MUST match, using the
2425 "urn:oasis:names:tc:xacml:1.0:function:string-equal" function, the `Issuer` of the same **attribute**. If the
2426 `Issuer` is not present in the attribute designator, then the matching of the **attribute** to the **named**
2427 **attribute** SHALL be governed by `AttributeId` and `DataType` attributes alone.

2428 The `<AttributeDesignatorType>` contains the following attributes:

2429 `Category` [Required]

2430 This attribute SHALL specify the `Category` with which to match the **attribute**.

2431 `AttributeId` [Required]

2432 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2433 `DataType` [Required]

2434 The **bag** returned by the `<AttributeDesignator>` element SHALL contain values of this data-
2435 type.

2436 `Issuer` [Optional]

2437 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2438 `MustBePresent` [Required]

2439 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the event
2440 the **named attribute** is absent from the request **context**. See Section 7.3.5. Also see Sections
2441 7.17.2 and 7.17.3.

2442 5.30 Element `<AttributeSelector>`

2443 The `<AttributeSelector>` element produces a **bag** of unnamed and uncategorized **attribute** values.
2444 The values shall be constructed from the node(s) selected by applying the XPath expression given by the
2445 element's `Path` attribute to the XML content indicated by the element's `Category` attribute. Support for
2446 the `<AttributeSelector>` element is OPTIONAL.

2447 See section 7.3.7 for details of `<AttributeSelector>` evaluation.

```

2448 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"
2449     substitutionGroup="xacml:Expression"/>
2450 <xs:complexType name="AttributeSelectorType">
2451     <xs:complexContent>
2452         <xs:extension base="xacml:ExpressionType">
2453             <xs:attribute name="Category" type="xs:anyURI"
2454                 use="required"/>

```

```

2455         <xs:attribute name="ContextSelectorId" type="xs:anyURI"
2456             use="optional"/>
2457         <xs:attribute name="Path" type="xs:string"
2458             use="required"/>
2459         <xs:attribute name="DataType" type="xs:anyURI"
2460             use="required"/>
2461         <xs:attribute name="MustBePresent" type="xs:boolean"
2462             use="required"/>
2463     </xs:extension>
2464 </xs:complexContent>
2465 </xs:complexType>

```

2466 The <AttributeSelector> element is of AttributeSelectorType complex type.

2467 The <AttributeSelector> element has the following attributes:

2468 Category [Required]

2469 This attribute SHALL specify the **attributes** category of the <Content> element containing the
2470 XML from which nodes will be selected. It also indicates the **attributes** category containing the
2471 applicable ContextSelectorId attribute, if the element includes a ContextSelectorId xml
2472 attribute.

2473 ContextSelectorId [Optional]

2474 This attribute refers to the **attribute** (by its AttributeId) in the request **context** in the category
2475 given by the Category attribute. The referenced **attribute** MUST have data type
2476 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression, and must select a single node in the
2477 <Content> element. The XPathCategory attribute of the referenced **attribute** MUST be equal
2478 to the Category attribute of the **attribute selector**.

2479 Path [Required]

2480 This attribute SHALL contain an XPath expression to be evaluated against the specified XML
2481 content. See Section 7.3.7 for details of the XPath evaluation during <AttributeSelector>
2482 processing.

2483 DataType [Required]

2484 The attribute specifies the datatype of the values returned from the evaluation of this
2485 <AttributeSelector> element.

2486 MustBePresent [Required]

2487 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the event
2488 the XPath expression selects no node. See Section 7.3.5. Also see Sections 7.17.2 and 7.17.3.

2489 5.31 Element <AttributeValue>

2490 The <AttributeValue> element SHALL contain a literal **attribute** value.

```

2491 <xs:element name="AttributeValue" type="xacml:AttributeValueType"
2492     substitutionGroup="xacml:Expression"/>
2493 <xs:complexType name="AttributeValueType" mixed="true">
2494     <xs:complexContent mixed="true">
2495         <xs:extension base="xacml:ExpressionType">
2496             <xs:sequence>
2497                 <xs:any namespace="##any" processContents="lax"
2498                     minOccurs="0" maxOccurs="unbounded"/>
2499             </xs:sequence>
2500             <xs:attribute name="DataType" type="xs:anyURI"
2501                 use="required"/>
2502             <xs:anyAttribute namespace="##any" processContents="lax"/>
2503         </xs:extension>
2504     </xs:complexContent>

```

2505 `</xs:complexType>`

2506 The `<AttributeValue>` element is of `AttributeValueType` complex type.

2507 The `<AttributeValue>` element has the following attributes:

2508 `DataType` [Required]

2509 The data-type of the **attribute** value.

2510 **5.32 Element `<Obligations>`**

2511 The `<Obligations>` element SHALL contain a set of `<Obligation>` elements.

```
2512 <xs:element name="Obligations" type="xacml:ObligationsType"/>
2513 <xs:complexType name="ObligationsType">
2514   <xs:sequence>
2515     <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>
2516   </xs:sequence>
2517 </xs:complexType>
```

2518 The `<Obligations>` element is of `ObligationsType` complexType.

2519 The `<Obligations>` element contains the following element:

2520 `<Obligation>` [One to Many]

2521 A sequence of **obligations**. See Section 5.34.

2522 **5.33 Element `<AssociatedAdvice>`**

2523 The `<AssociatedAdvice>` element SHALL contain a set of `<Advice>` elements.

```
2524 <xs:element name="AssociatedAdvice" type="xacml:AssociatedAdviceType"/>
2525 <xs:complexType name="AssociatedAdviceType">
2526   <xs:sequence>
2527     <xs:element ref="xacml:Advice" maxOccurs="unbounded"/>
2528   </xs:sequence>
2529 </xs:complexType>
```

2530 The `<AssociatedAdvice>` element is of `AssociatedAdviceType` complexType.

2531 The `<AssociatedAdvice>` element contains the following element:

2532 `<Advice>` [One to Many]

2533 A sequence of **advice**. See Section 5.35.

2534 **5.34 Element `<Obligation>`**

2535 The `<Obligation>` element SHALL contain an identifier for the **obligation** and a set of **attributes** that
2536 form arguments of the action defined by the **obligation**.

```
2537 <xs:element name="Obligation" type="xacml:ObligationType"/>
2538 <xs:complexType name="ObligationType">
2539   <xs:sequence>
2540     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2541       maxOccurs="unbounded"/>
2542   </xs:sequence>
2543   <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2544 </xs:complexType>
```

2545 The `<Obligation>` element is of `ObligationType` complexType. See Section 7.16 for a description
2546 of how the set of **obligations** to be returned by the **PDP** is determined.

2547 The `<Obligation>` element contains the following elements and attributes:

2548 `ObligationId` [Required]

2549 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the **PEP**.

2550 <AttributeAssignment> [Optional]

2551 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be
2552 interpreted by the **PEP**.

2553 5.35 Element <Advice>

2554 The <Advice> element SHALL contain an identifier for the **advice** and a set of **attributes** that form
2555 arguments of the supplemental information defined by the **advice**.

```
2556 <xs:element name="Advice" type="xacml:AdviceType"/>
2557 <xs:complexType name="AdviceType">
2558   <xs:sequence>
2559     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2560     maxOccurs="unbounded"/>
2561   </xs:sequence>
2562   <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>
2563 </xs:complexType>
```

2564 The <Advice> element is of AdviceType complexType. See Section 7.16 for a description of how the
2565 set of **advice** to be returned by the **PDP** is determined.

2566 The <Advice> element contains the following elements and attributes:

2567 AdviceId [Required]

2568 **Advice** identifier. The value of the **advice** identifier MAY be interpreted by the **PEP**.

2569 <AttributeAssignment> [Optional]

2570 **Advice** arguments assignment. The values of the **advice** arguments MAY be interpreted by the
2571 **PEP**.

2572 5.36 Element <AttributeAssignment>

2573 The <AttributeAssignment> element is used for including arguments in **obligation** and **advice**
2574 expressions. It SHALL contain an AttributeId and the corresponding **attribute** value, by extending
2575 the AttributeValueType type definition. The <AttributeAssignment> element MAY be used in
2576 any way that is consistent with the schema syntax, which is a sequence of <xs:any> elements. The
2577 value specified SHALL be understood by the **PEP**, but it is not further specified by XACML. See Section
2578 7.16. Section 4.2.4.3 provides a number of examples of arguments included in **obligation** expressions.

```
2579 <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>
2580 <xs:complexType name="AttributeAssignmentType" mixed="true">
2581   <xs:complexContent>
2582     <xs:extension base="xacml:AttributeValueType">
2583       <xs:attribute name="AttributeId" type="xs:anyURI"
2584       use="required"/>
2585       <xs:attribute name="Category" type="xs:anyURI"
2586       use="optional"/>
2587       <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2588     </xs:extension>
2589   </xs:complexContent>
2590 </xs:complexType>
```

2591 The <AttributeAssignment> element is of AttributeAssignmentType complex type.

2592 The <AttributeAssignment> element contains the following attributes:

2593 AttributeId [Required]

2594 The **attribute** Identifier.

2595 Category [Optional]

2596 An optional category of the **attribute**. If this attribute is missing, the **attribute** has no category.
2597 The **PEP** SHALL interpret the significance and meaning of any `Category` attribute. Non-
2598 normative note: an expected use of the category is to disambiguate **attributes** which are relayed
2599 from the request.

2600 Issuer [Optional]

2601 An optional issuer of the **attribute**. If this attribute is missing, the **attribute** has no issuer. The
2602 **PEP** SHALL interpret the significance and meaning of any `Issuer` attribute. Non-normative note:
2603 an expected use of the issuer is to disambiguate **attributes** which are relayed from the request.

2604 5.37 Element <ObligationExpressions>

2605 The <ObligationExpressions> element SHALL contain a set of <ObligationExpression>
2606 elements.

```
2607 <xs:element name="ObligationExpressions"  
2608   type="xacml:ObligationExpressionsType"/>  
2609 <xs:complexType name="ObligationExpressionsType">  
2610   <xs:sequence>  
2611     <xs:element ref="xacml:ObligationExpression" maxOccurs="unbounded"/>  
2612   </xs:sequence>  
2613 </xs:complexType>
```

2614 The <ObligationExpressions> element is of `ObligationExpressionsType` `complexType`.

2615 The <ObligationExpressions> element contains the following element:

2616 <ObligationExpression> [One to Many]

2617 A sequence of **obligations** expressions. See Section 5.39.

2618 5.38 Element <AdviceExpressions>

2619 The <AdviceExpressions> element SHALL contain a set of <AdviceExpression> elements.

```
2620 <xs:element name="AdviceExpressions" type="xacml:AdviceExpressionsType"/>  
2621 <xs:complexType name="AdviceExpressionsType">  
2622   <xs:sequence>  
2623     <xs:element ref="xacml:AdviceExpression" maxOccurs="unbounded"/>  
2624   </xs:sequence>  
2625 </xs:complexType>
```

2626 The <AdviceExpressions> element is of `AdviceExpressionsType` `complexType`.

2627 The <AdviceExpressions> element contains the following element:

2628 <AdviceExpression> [One to Many]

2629 A sequence of **advice** expressions. See Section 5.40.

2630 5.39 Element <ObligationExpression>

2631 The <ObligationExpression> element evaluates to an **obligation** and SHALL contain an identifier
2632 for an **obligation** and a set of expressions that form arguments of the action defined by the **obligation**.
2633 The `FulfillOn` attribute SHALL indicate the **effect** for which this **obligation** must be fulfilled by the
2634 **PEP**.

```
2635 <xs:element name="ObligationExpression"  
2636   type="xacml:ObligationExpressionType"/>  
2637 <xs:complexType name="ObligationExpressionType">  
2638   <xs:sequence>  
2639     <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"  
2640       maxOccurs="unbounded"/>  
2641   </xs:sequence>
```

```

2642     <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2643     <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
2644 </xs:complexType>

```

2645 The <ObligationExpression> element is of ObligationExpressionType complexType. See
 2646 Section 7.16 for a description of how the set of **obligations** to be returned by the **PDP** is determined.

2647 The <ObligationExpression> element contains the following elements and attributes:

2648 ObligationId [Required]

2649 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the **PEP**.

2650 FulfillOn [Required]

2651 The **effect** for which this **obligation** must be fulfilled by the **PEP**.

2652 <AttributeAssignmentExpression> [Optional]

2653 **Obligation** arguments in the form of expressions. The expressions SHALL be evaluated by the
 2654 PDP to constant <AttributeValue> elements or **bags**, which shall be the attribute
 2655 assignments in the <Obligation> returned to the PEP. If an
 2656 <AttributeAssignmentExpression> evaluates to an atomic **attribute** value, then there
 2657 MUST be one resulting <AttributeAssignment> which MUST contain this single **attribute**
 2658 value. If the <AttributeAssignmentExpression> evaluates to a **bag**, then there MUST be a
 2659 resulting <AttributeAssignment> for each of the values in the **bag**. If the **bag** is empty, there
 2660 shall be no <AttributeAssignment> from this <AttributeAssignmentExpression>.The
 2661 values of the **obligation** arguments SHALL be interpreted by the **PEP**.

2662 5.40 Element <AdviceExpression>

2663 The <AdviceExpression> element evaluates to an **advice** and SHALL contain an identifier for an
 2664 **advice** and a set of expressions that form arguments of the supplemental information defined by the
 2665 **advice**. The AppliesTo attribute SHALL indicate the **effect** for which this **advice** must be provided to
 2666 the **PEP**.

```

2667 <xs:element name="AdviceExpression" type="xacml:AdviceExpressionType"/>
2668 <xs:complexType name="AdviceExpressionType">
2669   <xs:sequence>
2670     <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"
2671     maxOccurs="unbounded"/>
2672   </xs:sequence>
2673   <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>
2674   <xs:attribute name="AppliesTo" type="xacml:EffectType" use="required"/>
2675 </xs:complexType>

```

2676 The <AdviceExpression> element is of AdviceExpressionType complexType. See Section 7.16
 2677 for a description of how the set of **advice** to be returned by the **PDP** is determined.

2678 The <AdviceExpression> element contains the following elements and attributes:

2679 AdviceId [Required]

2680 **Advice** identifier. The value of the **advice** identifier MAY be interpreted by the **PEP**.

2681 AppliesTo [Required]

2682 The **effect** for which this **advice** must be provided to the **PEP**.

2683 <AttributeAssignmentExpression> [Optional]

2684 **Advice** arguments in the form of expressions. The expressions SHALL be evaluated by the PDP
 2685 to constant <AttributeValue> elements or **bags**, which shall be the attribute assignments in
 2686 the <Advice> returned to the PEP. If an <AttributeAssignmentExpression> evaluates to
 2687 an atomic **attribute** value, then there MUST be one resulting <AttributeAssignment> which
 2688 MUST contain this single **attribute** value. If the <AttributeAssignmentExpression>

2689 evaluates to a **bag**, then there MUST be a resulting <AttributeAssignment> for each of the
2690 values in the **bag**. If the **bag** is empty, there shall be no <AttributeAssignment> from this
2691 <AttributeAssignmentExpression>. The values of the **advice** arguments MAY be
2692 interpreted by the **PEP**.

2693 5.41 Element <AttributeAssignmentExpression>

2694 The <AttributeAssignmentExpression> element is used for including arguments in **obligations**
2695 and **advice**. It SHALL contain an AttributeId and an expression which SHALL be evaluated into the
2696 corresponding **attribute** value. The value specified SHALL be understood by the **PEP**, but it is not further
2697 specified by XACML. See Section 7.16. Section 4.2.4.3 provides a number of examples of arguments
2698 included in **obligations**.

```
2699 <xs:element name="AttributeAssignmentExpression"  
2700     type="xacml:AttributeAssignmentExpressionType"/>  
2701 <xs:complexType name="AttributeAssignmentExpressionType">  
2702     <xs:sequence>  
2703         <xs:element ref="xacml:Expression"/>  
2704     </xs:sequence>  
2705     <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2706     <xs:attribute name="Category" type="xs:anyURI" use="optional"/>  
2707     <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2708 </xs:complexType>
```

2709 The <AttributeAssignmentExpression> element is of AttributeAssignmentExpressionType
2710 complex type.

2711 The <AttributeAssignmentExpression> element contains the following attributes:

2712 <Expression> [Required]

2713 The expression which evaluates to a constant **attribute** value or a bag of zero or more attribute
2714 values. See section 5.25.

2715 AttributeId [Required]

2716 The **attribute** identifier. The value of the AttributeId attribute in the resulting
2717 <AttributeAssignment> element MUST be equal to this value.

2718 Category [Optional]

2719 An optional category of the **attribute**. If this attribute is missing, the **attribute** has no category.
2720 The value of the Category attribute in the resulting <AttributeAssignment> element MUST be
2721 equal to this value.

2722 Issuer [Optional]

2723 An optional issuer of the **attribute**. If this attribute is missing, the **attribute** has no issuer. The
2724 value of the Issuer attribute in the resulting <AttributeAssignment> element MUST be equal to
2725 this value.

2726 5.42 Element <Request>

2727 The <Request> element is an abstraction layer used by the **policy** language. For simplicity of
2728 expression, this document describes **policy** evaluation in terms of operations on the **context**. However a
2729 conforming **PDP** is not required to actually instantiate the **context** in the form of an XML document. But,
2730 any system conforming to the XACML specification MUST produce exactly the same **authorization**
2731 **decisions** as if all the inputs had been transformed into the form of an <Request> element.

2732 The <Request> element contains <Attributes> elements. There may be multiple <Attributes>
2733 elements with the same Category attribute if the **PDP** implements the multiple decision profile, see
2734 **[Multi]**. Under other conditions, it is a syntax error if there are multiple <Attributes> elements with the
2735 same Category (see Section 7.17.2 for error codes).


```

2736 <xs:element name="Request" type="xacml:RequestType"/>
2737 <xs:complexType name="RequestType">
2738   <xs:sequence>
2739     <xs:element ref="xacml:RequestDefaults" minOccurs="0"/>
2740     <xs:element ref="xacml:Attributes" maxOccurs="unbounded"/>
2741     <xs:element ref="xacml:MultiRequests" minOccurs="0"/>
2742   </xs:sequence>
2743   <xs:attribute name="ReturnPolicyIdList" type="xs:boolean" use="required"/>
2744   <xs:attribute name="CombinedDecision" type="xs:boolean" use="required" />
2745 </xs:complexType>

```

2746 The <Request> element is of RequestType complex type.

2747 The <Request> element contains the following elements and attributes:

2748 ReturnPolicyIdList [Required]

2749 This attribute is used to request that the **PDP** return a list of all fully applicable **policies** and
2750 **policy sets** which were used in the decision as a part of the decision response.

2751 CombinedDecision [Required]

2752 This attribute is used to request that the **PDP** combines multiple decisions into a single decision.
2753 The use of this attribute is specified in [Multi]. If the **PDP** does not implement the relevant
2754 functionality in [Multi], then the **PDP** must return an Indeterminate with a status code of
2755 urn:oasis:names:tc:xacml:1.0:status:processing-error if it receives a request with this attribute set
2756 to "true".

2757 <RequestDefaults> [Optional]

2758 Contains default values for the request, such as XPath version. See section 5.43.

2759 <Attributes> [One to Many]

2760 Specifies information about **attributes** of the request **context** by listing a sequence of
2761 <Attribute> elements associated with an **attribute** category. One or more <Attributes>
2762 elements are allowed. Different <Attributes> elements with different categories are used to
2763 represent information about the **subject**, **resource**, **action**, **environment** or other categories of
2764 the **access** request.

2765 <MultiRequests> [Optional]

2766 Lists multiple **request contexts** by references to the <Attributes> elements. Implementation
2767 of this element is optional. The semantics of this element is defined in [Multi]. If the
2768 implementation does not implement this element, it MUST return an Indeterminate result if it
2769 encounters this element. See section 5.50.

2770 5.43 Element <RequestDefaults>

2771 The <RequestDefaults> element SHALL specify default values that apply to the <Request> element.

```

2772 <xs:element name="RequestDefaults" type="xacml:RequestDefaultsType"/>
2773 <xs:complexType name="RequestDefaultsType">
2774   <xs:sequence>
2775     <xs:choice>
2776       <xs:element ref="xacml:XPathVersion"/>
2777     </xs:choice>
2778   </xs:sequence>
2779 </xs:complexType>

```

2780 <RequestDefaults> element is of RequestDefaultsType complex type.

2781 The <RequestDefaults> element contains the following elements:

2782 <XPathVersion> [Optional]

2783 Default XPath version for XPath expressions occurring in the request.

2784 5.44 Element <Attributes>

2785 The <Attributes> element specifies **attributes** of a **subject**, **resource**, **action**, **environment** or
2786 another category by listing a sequence of <Attribute> elements associated with the category.

```
2787 <xs:element name="Attributes" type="xacml:AttributesType"/>
2788 <xs:complexType name="AttributesType">
2789   <xs:sequence>
2790     <xs:element ref="xacml:Content" minOccurs="0"/>
2791     <xs:element ref="xacml:Attribute" minOccurs="0"
2792       maxOccurs="unbounded"/>
2793   </xs:sequence>
2794   <xs:attribute name="Category" type="xs:anyURI" use="required"/>
2795   <xs:attribute ref="xml:id" use="optional"/>
2796 </xs:complexType><xs:complexType name="SubjectType">
```

2797 The <Attributes> element is of AttributesType complex type.

2798 The <Attributes> element contains the following elements and attributes:

2799 Category [Required]

2800 This attribute indicates which **attribute** category the contained **attributes** belong to. The
2801 Category attribute is used to differentiate between **attributes** of **subject**, **resource**, **action**,
2802 **environment** or other categories.

2803 xml:id [Optional]

2804 This attribute provides a unique identifier for this <Attributes> element. See [XMLid] It is
2805 primarily intended to be referenced in multiple requests. See [Multi].

2806 <Content> [Optional]

2807 Specifies additional sources of **attributes** in free form XML document format which can be
2808 referenced using <AttributeSelector> elements.

2809 <Attribute> [Any Number]

2810 A sequence of **attributes** that apply to the category of the request.

2811 5.45 Element <Content>

2812 The <Content> element is a notional placeholder for additional **attributes**, typically the content of the
2813 **resource**.

```
2814 <xs:element name="Content" type="xacml:ContentType"/>
2815 <xs:complexType name="ContentType" mixed="true">
2816   <xs:sequence>
2817     <xs:any namespace="##any" processContents="lax"/>
2818   </xs:sequence>
2819 </xs:complexType>
```

2820 The <Content> element is of ContentType complex type.

2821 The <Content> element has exactly one arbitrary type child element.

2822 5.46 Element <Attribute>

2823 The <Attribute> element is the central abstraction of the request **context**. It contains **attribute** meta-
2824 data and one or more **attribute** values. The **attribute** meta-data comprises the **attribute** identifier and
2825 the **attribute** issuer. <AttributeDesignator> elements in the **policy** MAY refer to **attributes** by
2826 means of this meta-data.

```
2827 <xs:element name="Attribute" type="xacml:AttributeType"/>
2828 <xs:complexType name="AttributeType">
2829   <xs:sequence>
```

```

2830     <xs:element ref="xacml:AttributeValue" maxOccurs="unbounded"/>
2831 </xs:sequence>
2832 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
2833 <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2834 <xs:attribute name="IncludeInResult" type="xs:boolean" use="required"/>
2835 </xs:complexType>

```

2836 The <Attribute> element is of AttributeType complex type.

2837 The <Attribute> element contains the following attributes and elements:

2838 AttributeId [Required]

2839 The **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly
 2840 used **attributes**. See Appendix B.

2841 Issuer [Optional]

2842 The **Attribute** issuer. For example, this attribute value MAY be an x500Name that binds to a
 2843 public key, or it may be some other identifier exchanged out-of-band by issuing and relying
 2844 parties.

2845 IncludeInResult [Default: false]

2846 Whether to include this **attribute** in the result. This is useful to correlate requests with their
 2847 responses in case of multiple requests.

2848 <AttributeValue> [One to Many]

2849 One or more **attribute** values. Each **attribute** value MAY have contents that are empty, occur
 2850 once or occur multiple times.

2851 5.47 Element <Response>

2852 The <Response> element is an abstraction layer used by the **policy** language. Any proprietary system
 2853 using the XACML specification MUST transform an XACML **context** <Response> element into the form
 2854 of its **authorization decision**.

2855 The <Response> element encapsulates the **authorization decision** produced by the **PDP**. It includes a
 2856 sequence of one or more results, with one <Result> element per requested **resource**. Multiple results
 2857 MAY be returned by some implementations, in particular those that support the XACML Profile for
 2858 Requests for Multiple Resources [**Multi**]. Support for multiple results is OPTIONAL.

```

2859 <xs:element name="Response" type="xacml:ResponseType"/>
2860 <xs:complexType name="ResponseType">
2861 <xs:sequence>
2862 <xs:element ref="xacml:Result" maxOccurs="unbounded"/>
2863 </xs:sequence>
2864 </xs:complexType>

```

2865 The <Response> element is of ResponseType complex type.

2866 The <Response> element contains the following elements:

2867 <Result> [One to Many]

2868 An **authorization decision** result. See Section 5.48.

2869 5.48 Element <Result>

2870 The <Result> element represents an **authorization decision** result. It MAY include a set of
 2871 **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or cannot fulfill an
 2872 **obligation**, then the action of the PEP is determined by its bias, see section 7.1. It MAY include a set of
 2873 **advice** with supplemental information which MAY be safely ignored by the **PEP**.

```

2874 <xs:complexType name="ResultType">
2875 <xs:sequence>

```

```

2876     <xs:element ref="xacml:Decision"/>
2877     <xs:element ref="xacml:Status" minOccurs="0"/>
2878     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2879     <xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
2880     <xs:element ref="xacml:Attributes" minOccurs="0"
2881         maxOccurs="unbounded"/>
2882     <xs:element ref="xacml:PolicyIdentifierList" minOccurs="0"/>
2883 </xs:sequence>
2884 </xs:complexType>

```

2885 The <Result> element is of ResultType complex type.

2886 The <Result> element contains the following attributes and elements:

2887 <Decision> [Required]

2888 The **authorization decision**: “Permit”, “Deny”, “Indeterminate” or “NotApplicable”.

2889 <Status> [Optional]

2890 Indicates whether errors occurred during evaluation of the **decision request**, and optionally,
2891 information about those errors. If the <Response> element contains <Result> elements whose
2892 <Status> elements are all identical, and the <Response> element is contained in a protocol
2893 wrapper that can convey status information, then the common status information MAY be placed
2894 in the protocol wrapper and this <Status> element MAY be omitted from all <Result>
2895 elements.

2896 <Obligations> [Optional]

2897 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or cannot
2898 fulfill an **obligation**, then the action of the PEP is determined by its bias, see section 7.2. See
2899 Section 7.16 for a description of how the set of **obligations** to be returned by the **PDP** is
2900 determined.

2901 <AssociatedAdvice> [Optional]

2902 A list of **advice** that provide supplemental information to the **PEP**. If the **PEP** does not
2903 understand an **advice**, the PEP may safely ignore the **advice**. See Section 7.16 for a description
2904 of how the set of **advice** to be returned by the **PDP** is determined.

2905 <Attributes> [Optional]

2906 A list of **attributes** that were part of the request. The choice of which **attributes** are included here
2907 is made with the IncludeInResult attribute of the <Attribute> elements of the request. See
2908 section 5.46.

2909 <PolicyIdentifierList> [Optional]

2910 If the ReturnPolicyIdList attribute in the <Request> is true (see section 5.42), a **PDP** that
2911 implements this optional feature MUST return a list of all **policies** which were found to be fully
2912 applicable. That is, all **policies** where both the <Target> matched and the <Condition>
2913 evaluated to true, whether or not the <Effect> was the same or different from the <Decision>.

2914 5.49 Element <PolicyIdentifierList>

2915 The <PolicyIdentifierList> element contains a list of **policy** and **policy set** identifiers of **policies**
2916 which have been applicable to a request. The list is unordered.

```

2917 <xs:element name="PolicyIdentifierList"
2918     type="xacml:PolicyIdentifierListType"/>
2919 <xs:complexType name="PolicyIdentifierListType">
2920     <xs:choice minOccurs="0" maxOccurs="unbounded">
2921         <xs:element ref="xacml:PolicyIdReference"/>
2922         <xs:element ref="xacml:PolicySetIdReference"/>
2923     </xs:choice>

```

2924 `</xs:complexType>`

2925 The `<PolicyIdentifierList>` element is of `PolicyIdentifierListType` complex type.

2926 The `<PolicyIdentifierList>` element contains the following elements.

2927 `<PolicyIdReference>` [Any number]

2928 The identifier and version of a **policy** which was applicable to the request. See section 5.11. The
2929 `<PolicyIdReference>` element **MUST** use the `Version` attribute to specify the version and
2930 **MUST NOT** use the `LatestVersion` or `EarliestVersion` attributes.

2931 `<PolicySetIdReference>` [Any number]

2932 The identifier and version of a **policy set** which was applicable to the request. See section 5.10.
2933 The `<PolicySetIdReference>` element **MUST** use the `Version` attribute to specify the
2934 version and **MUST NOT** use the `LatestVersion` or `EarliestVersion` attributes.

2935 **5.50 Element `<MultiRequests>`**

2936 The `<MultiRequests>` element contains a list of requests by reference to `<Attributes>` elements in
2937 the enclosing `<Request>` element. The semantics of this element are defined in **[Multi]**. Support for this
2938 element is optional. If an implementation does not support this element, but receives it, the
2939 implementation **MUST** generate an "Indeterminate" response.

```
2940 <xs:element name="MultiRequests" type="xacml:MultiRequestsType"/>
2941 <xs:complexType name="MultiRequestsType">
2942   <xs:sequence>
2943     <xs:element ref="xacml:RequestReference" maxOccurs="unbounded"/>
2944   </xs:sequence>
2945 </xs:complexType>
```

2946 The `<MultiRequests>` element contains the following elements.

2947 `<RequestReference>` [one to many]

2948 Defines a request instance by reference to `<Attributes>` elements in the enclosing
2949 `<Request>` element. See section 5.51.

2950 **5.51 Element `<RequestReference>`**

2951 The `<RequestReference>` element defines an instance of a request in terms of references to
2952 `<Attributes>` elements. The semantics of this element are defined in **[Multi]**. Support for this element
2953 is optional.

```
2954 <xs:element name="RequestReference" type="xacml:RequestReference"/>
2955 <xs:complexType name="RequestReferenceType">
2956   <xs:sequence>
2957     <xs:element ref="xacml:AttributesReference" maxOccurs="unbounded"/>
2958   </xs:sequence>
2959 </xs:complexType>
```

2960 The `<RequestReference>` element contains the following elements.

2961 `<AttributesReference>` [one to many]

2962 A reference to an `<Attributes>` element in the enclosing `<Request>` element. See section
2963 5.52.

2964 **5.52 Element `<AttributesReference>`**

2965 The `<AttributesReference>` element makes a reference to an `<Attributes>` element. The
2966 meaning of this element is defined in **[Multi]**. Support for this element is optional.

```
2967 <xs:element name="AttributesReference" type="xacml:AttributesReference"/>
```

```

2968 <xs:complexType name="AttributesReferenceType">
2969   <xs:attribute name="ReferenceId" type="xs:IDREF" use="required" />
2970 </xs:complexType>

```

2971 The <AttributesReference> element contains the following attributes.

2972 ReferenceId [required]

2973 A reference to the `xml:id` attribute of an <Attributes> element in the enclosing <Request>
 2974 element.

2975 5.53 Element <Decision>

2976 The <Decision> element contains the result of *policy* evaluation.

```

2977 <xs:element name="Decision" type="xacml:DecisionType"/>
2978 <xs:simpleType name="DecisionType">
2979   <xs:restriction base="xs:string">
2980     <xs:enumeration value="Permit"/>
2981     <xs:enumeration value="Deny"/>
2982     <xs:enumeration value="Indeterminate"/>
2983     <xs:enumeration value="NotApplicable"/>
2984   </xs:restriction>
2985 </xs:simpleType>

```

2986 The <Decision> element is of `DecisionType` simple type.

2987 The values of the <Decision> element have the following meanings:

2988 "Permit": the requested **access** is permitted.

2989 "Deny": the requested **access** is denied.

2990 "Indeterminate": the **PDP** is unable to evaluate the requested **access**. Reasons for such inability
 2991 include: missing **attributes**, network errors while retrieving **policies**, division by zero during
 2992 **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc.

2993 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

2994 5.54 Element <Status>

2995 The <Status> element represents the status of the **authorization decision** result.

```

2996 <xs:element name="Status" type="xacml:StatusType"/>
2997 <xs:complexType name="StatusType">
2998   <xs:sequence>
2999     <xs:element ref="xacml:StatusCode"/>
3000     <xs:element ref="xacml:StatusMessage" minOccurs="0"/>
3001     <xs:element ref="xacml:StatusDetail" minOccurs="0"/>
3002   </xs:sequence>
3003 </xs:complexType>

```

3004 The <Status> element is of `StatusType` complex type.

3005 The <Status> element contains the following elements:

3006 <StatusCode> [Required]

3007 Status code.

3008 <StatusMessage> [Optional]

3009 A status message describing the status code.

3010 <StatusDetail> [Optional]

3011 Additional status information.

3012 5.55 Element <StatusCode>

3013 The <StatusCode> element contains a major status code value and an optional sequence of minor
3014 status codes.

```
3015 <xs:element name="StatusCode" type="xacml:StatusCodeType"/>
3016 <xs:complexType name="StatusCodeType">
3017   <xs:sequence>
3018     <xs:element ref="xacml:StatusCode" minOccurs="0"/>
3019   </xs:sequence>
3020   <xs:attribute name="Value" type="xs:anyURI" use="required"/>
3021 </xs:complexType>
```

3022 The <StatusCode> element is of StatusCodeType complex type.

3023 The <StatusCode> element contains the following attributes and elements:

3024 Value [Required]

3025 See Section B.8 for a list of values.

3026 <StatusCode> [Any Number]

3027 Minor status code. This status code qualifies its parent status code.

3028 5.56 Element <StatusMessage>

3029 The <StatusMessage> element is a free-form description of the status code.

```
3030 <xs:element name="StatusMessage" type="xs:string"/>
```

3031 The <StatusMessage> element is of xs:string type.

3032 5.57 Element <StatusDetail>

3033 The <StatusDetail> element qualifies the <Status> element with additional information.

```
3034 <xs:element name="StatusDetail" type="xacml:StatusDetailType"/>
3035 <xs:complexType name="StatusDetailType">
3036   <xs:sequence>
3037     <xs:any namespace="##any" processContents="lax" minOccurs="0"
3038       maxOccurs="unbounded"/>
3039   </xs:sequence>
3040 </xs:complexType>
```

3041 The <StatusDetail> element is of StatusDetailType complex type.

3042 The <StatusDetail> element allows arbitrary XML content.

3043 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the following
3044 XACML-defined <StatusCode> values and includes a <StatusDetail> element, then the following
3045 rules apply.

3046 urn:oasis:names:tc:xacml:1.0:status:ok

3047 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

3048 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

3049 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a
3050 <StatusDetail> element containing one or more <MissingAttributeDetail> elements.

3051 urn:oasis:names:tc:xacml:1.0:status:syntax-error

3052 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status
3053 value. A syntax error may represent either a problem with the **policy** being used or with the request
3054 **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

3055 urn:oasis:names:tc:xacml:1.0:status:processing-error

3056 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error” status
3057 value. This status code indicates an internal problem in the **PDP**. For security reasons, the **PDP** MAY
3058 choose to return no further information to the **PEP**. In the case of a divide-by-zero error or other
3059 computational error, the **PDP** MAY return a <StatusMessage> describing the nature of the error.

3060 5.58 Element <MissingAttributeDetail>

3061 The <MissingAttributeDetail> element conveys information about **attributes** required for **policy**
3062 evaluation that were missing from the request **context**.

```
3063 <xs:element name="MissingAttributeDetail"  
3064 type="xacml:MissingAttributeDetailType"/>  
3065 <xs:complexType name="MissingAttributeDetailType">  
3066 <xs:sequence>  
3067 <xs:element ref="xacml:AttributeValue" minOccurs="0"  
3068 maxOccurs="unbounded"/>  
3069 </xs:sequence>  
3070 <xs:attribute name="Category" type="xs:anyURI" use="required"/>  
3071 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
3072 <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
3073 <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
3074 </xs:complexType>
```

3075 The <MissingAttributeDetail> element is of MissingAttributeDetailType complex type.

3076 The <MissingAttributeDetail> element contains the following attributes and elements:

3077 <AttributeValue> [Optional]

3078 The required value of the missing **attribute**.

3079 Category [Required]

3080 The category identifier of the missing **attribute**.

3081 AttributeId [Required]

3082 The identifier of the missing **attribute**.

3083 DataType [Required]

3084 The data-type of the missing **attribute**.

3085 Issuer [Optional]

3086 This attribute, if supplied, SHALL specify the required Issuer of the missing **attribute**.

3087 If the **PDP** includes <AttributeValue> elements in the <MissingAttributeDetail> element, then
3088 this indicates the acceptable values for that **attribute**. If no <AttributeValue> elements are included,
3089 then this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list of
3090 **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the missing
3091 values or **attributes** will be sufficient to satisfy the **policy**.

3092 6 XPath 2.0 definitions

3093 The XPath 2.0 specification leaves a number of aspects of behavior implementation defined. This section
3094 defines how XPath 2.0 SHALL behave when hosted in XACML.

3095 <http://www.w3.org/TR/2007/REC-xpath20-20070123/#id-impl-defined-items> defines the following items:

- 3096 1. The version of Unicode that is used to construct expressions.
3097 XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.
- 3098 2. The statically-known collations.
3099 XACML leaves this implementation defined.
- 3100 3. The implicit timezone.
3101 XACML defined the implicit time zone as UTC.
- 3102 4. The circumstances in which warnings are raised, and the ways in which warnings are handled.
3103 XACML leaves this implementation defined.
- 3104 5. The method by which errors are reported to the external processing environment.
3105 An XPath error causes an XACML Indeterminate value in the element where the XPath error
3106 occurs. The StatusCode value SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-error".
3107 Implementations MAY provide additional details about the error in the response or by some other
3108 means.
- 3109 6. Whether the implementation is based on the rules of XML 1.0 or 1.1.
3110 XACML is based on XML 1.0.
- 3111 7. Whether the implementation supports the namespace axis.
3112 XACML leaves this implementation defined. It is RECOMMENDED that users of XACML do not
3113 make use of the namespace axis.
- 3114 8. Any static typing extensions supported by the implementation, if the Static Typing Feature is
3115 supported.
3116 XACML leaves this implementation defined.

3117

3118 <http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/#implementation-defined> defines the
3119 following items:

- 3120 1. Support for additional user-defined or implementation-defined types is implementation-defined.
3121 It is RECOMMENDED that implementations of XACML do not define any additional types and it is
3122 RECOMMENDED that users of XACML do not make user of any additional types.
- 3123 2. Some typed values in the data model are undefined. Attempting to access an undefined property
3124 is always an error. Behavior in these cases is implementation-defined and the host language is
3125 responsible for determining the result.
3126 An XPath error causes an XACML Indeterminate value in the element where the XPath error
3127 occurs. The StatusCode value SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-error".
3128 Implementations MAY provide additional details about the error in the response or by some other
3129 means.

3130

3131 <http://www.w3.org/TR/2007/REC-xpath-functions-20070123/#impl-def> defines the following items:

- 3132 1. The destination of the trace output is implementation-defined.
3133 XACML leaves this implementation defined.
- 3134 2. For xs:integer operations, implementations that support limited-precision integer operations must
3135 either raise an error [err:FOAR0002] or provide an implementation-defined mechanism that
3136 allows users to choose between raising an error and returning a result that is modulo the largest
3137 representable integer value.
3138 XACML leaves this implementation defined. If an implementation chooses to raise an error, the

- 3139 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 3. For xs:decimal values the number of digits of precision returned by the numeric operators is
- 3143 implementation-defined.
- 3144 XACML leaves this implementation defined.
- 3145 4. If the number of digits in the result of a numeric operation exceeds the number of digits that the
- 3146 implementation supports, the result is truncated or rounded in an implementation-defined manner.
- 3147 XACML leaves this implementation defined.
- 3148 5. It is implementation-defined which version of Unicode is supported.
- 3149 XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.
- 3150 6. For fn:normalize-unicode, conforming implementations must support normalization form "NFC"
- 3151 and may support normalization forms "NFD", "NFKC", "NFKD", "FULLY-NORMALIZED". They
- 3152 may also support other normalization forms with implementation-defined semantics.
- 3153 XACML leaves this implementation defined.
- 3154 7. The ability to decompose strings into collation units suitable for substring matching is an
- 3155 implementation-defined property of a collation.
- 3156 XACML leaves this implementation defined.
- 3157 8. All minimally conforming processors must support year values with a minimum of 4 digits (i.e.,
- 3158 YYYY) and a minimum fractional second precision of 1 millisecond or three digits (i.e., s.sss).
- 3159 However, conforming processors may set larger implementation-defined limits on the maximum
- 3160 number of digits they support in these two situations.
- 3161 XACML leaves this implementation defined, and it is RECOMMENDED that users of XACML do
- 3162 not expect greater limits and precision.
- 3163 9. The result of casting a string to xs:decimal, when the resulting value is not too large or too small
- 3164 but nevertheless has too many decimal digits to be accurately represented, is implementation-
- 3165 defined.
- 3166 XACML leaves this implementation defined.
- 3167 10. Various aspects of the processing provided by fn:doc are implementation-defined.
- 3168 Implementations may provide external configuration options that allow any aspect of the
- 3169 processing to be controlled by the user.
- 3170 XACML leaves this implementation defined.
- 3171 11. The manner in which implementations provide options to weaken the stable characteristic of
- 3172 fn:collection and fn:doc are implementation-defined.
- 3173 XACML leaves this implementation defined.

3174 7 Functional requirements

3175 This section specifies certain functional requirements that are not directly associated with the production
3176 or consumption of a particular XACML element.

3177 7.1 Unicode issues

3178 7.1.1 Normalization

3179 In Unicode, some equivalent characters can be represented by more than one different Unicode
3180 character sequence. See [CMF]. The process of converting Unicode strings into equivalent character
3181 sequences is called "normalization" [UAX15]. Some operations, such as string comparison, are sensitive
3182 to normalization. An operation is normalization-sensitive if its output(s) are different depending on the
3183 state of normalization of the input(s); if the output(s) are textual, they are deemed different only if they
3184 would remain different were they to be normalized.

3185 For more information on normalization see [CM].

3186 An XACML implementation MUST behave as if each normalization-sensitive operation normalizes input
3187 strings into Unicode Normalization Form C ("NFC"). An implementation MAY use some other form of
3188 internal processing (such as using a non-Unicode, "legacy" character encoding) as long as the externally
3189 visible results are identical to this specification.

3190 7.1.2 Version of Unicode

3191 The version of Unicode used by XACML is implementation defined. It is RECOMMENDED that the latest
3192 version is used. Also note security issues in section 9.3.

3193 7.2 Policy enforcement point

3194 This section describes the requirements for the *PEP*.

3195 An application functions in the role of the *PEP* if it guards *access* to a set of *resources* and asks the
3196 *PDP* for an *authorization decision*. The *PEP* MUST abide by the *authorization decision* as described
3197 in one of the following sub-sections

3198 In any case any *advice* in the *decision* may be safely ignored by the *PEP*.

3199 7.2.1 Base PEP

3200 If the *decision* is "Permit", then the *PEP* SHALL permit *access*. If *obligations* accompany the *decision*,
3201 then the *PEP* SHALL permit *access* only if it understands and it can and will discharge those
3202 *obligations*.

3203 If the *decision* is "Deny", then the *PEP* SHALL deny *access*. If *obligations* accompany the *decision*,
3204 then the *PEP* shall deny *access* only if it understands, and it can and will discharge those *obligations*.

3205 If the *decision* is "Not Applicable", then the *PEP*'s behavior is undefined.

3206 If the *decision* is "Indeterminate", then the *PEP*'s behavior is undefined.

3207 7.2.2 Deny-biased PEP

3208 If the *decision* is "Permit", then the *PEP* SHALL permit *access*. If *obligations* accompany the *decision*,
3209 then the *PEP* SHALL permit *access* only if it understands and it can and will discharge those
3210 *obligations*.

3211 All other *decisions* SHALL result in the denial of *access*.

3212 Note: other actions, e.g. consultation of additional *PDPs*, reformulation/resubmission of
3213 the *decision request*, etc., are not prohibited.

3214 7.2.3 Permit-biased PEP

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

3217 All other *decisions* SHALL result in the permission of *access*.

3218 Note: other actions, e.g. consultation of additional *PDPs*, reformulation/resubmission of
3219 the *decision request*, etc., are not prohibited.

3220 7.3 Attribute evaluation

3221 *Attributes* are represented in the request *context* by the *context handler*, regardless of whether or not
3222 they appeared in the original *decision request*, and are referred to in the *policy* by attribute designators
3223 and attribute selectors. A *named attribute* is the term used for the criteria that the specific attribute
3224 designators use to refer to particular *attributes* in the <Attributes> elements of the request *context*.

3225 7.3.1 Structured attributes

3226 <AttributeValue> elements MAY contain an instance of a structured XML data-type, for example
3227 <ds:KeyInfo>. XACML 3.0 supports several ways for comparing the contents of such elements.

- 3228 1. In some cases, such elements MAY be compared using one of the XACML string functions, such
3229 as "string-regexp-match", described below. This requires that the element be given the data-type
3230 "http://www.w3.org/2001/XMLSchema#string". For example, a structured data-type that is
3231 actually a ds:KeyInfo/KeyName would appear in the *Context* as:

```
3232 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3233 <ds:KeyName>jhibbert-key</ds:KeyName>  
3234 </AttributeValue>
```

3235 In general, this method will not be adequate unless the structured data-type is quite simple.

- 3236 2. The structured *attribute* MAY be made available in the <Content> element of the appropriate
3237 *attribute* category and an <AttributeSelector> element MAY be used to select the contents
3238 of a leaf sub-element of the structured data-type by means of an XPath expression. That value
3239 MAY then be compared using one of the supported XACML functions appropriate for its primitive
3240 data-type. This method requires support by the *PDP* for the optional XPath expressions feature.

- 3241 3. The structured *attribute* MAY be made available in the <Content> element of the appropriate
3242 *attribute* category and an <AttributeSelector> element MAY be used to select any node in
3243 the structured data-type by means of an XPath expression. This node MAY then be compared
3244 using one of the XPath-based functions described in Section A.3.15. This method requires
3245 support by the *PDP* for the optional XPath expressions and XPath functions features.

3246 See also Section 7.3.

3247 7.3.2 Attribute bags

3248 XACML defines implicit collections of its data-types. XACML refers to a collection of values that are of a
3249 single data-type as a *bag*. *Bags* of data-types are needed because selections of nodes from an XML
3250 *resource* or XACML request *context* may return more than one value.

3251 The <AttributeSelector> element uses an XPath expression to specify the selection of data from
3252 free form XML. The result of an XPath expression is termed a node-set, which contains all the nodes
3253 from the XML content that match the *predicate* in the XPath expression. Based on the various indexing
3254 functions provided in the XPath specification, it SHALL be implied that a resultant node-set is the
3255 collection of the matching nodes. XACML also defines the <AttributeDesignator> element to have
3256 the same matching methodology for *attributes* in the XACML request *context*.

3257 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be no
3258 notion of a **bag** containing **bags**, or a **bag** containing values of differing types; i.e., a **bag** in XACML
3259 SHALL contain only values that are of the same data-type.

3260 7.3.3 Multivalued attributes

3261 If a single <Attribute> element in a request **context** contains multiple <AttributeValue> child
3262 elements, then the **bag** of values resulting from evaluation of the <Attribute> element MUST be
3263 identical to the **bag** of values that results from evaluating a **context** in which each <AttributeValue>
3264 element appears in a separate <Attribute> element, each carrying identical meta-data.

3265 7.3.4 Attribute Matching

3266 A **named attribute** includes specific criteria with which to match **attributes** in the **context**. An **attribute**
3267 specifies a `Category`, `AttributeId` and `DataType`, and a **named attribute** also specifies the
3268 `Issuer`. A **named attribute** SHALL match an **attribute** if the values of their respective `Category`,
3269 `AttributeId`, `DataType` and optional `Issuer` attributes match. The `Category` of the **named**
3270 **attribute** MUST match, by URI equality, the `Category` of the corresponding **context attribute**. The
3271 `AttributeId` of the **named attribute** MUST match, by URI equality, the `AttributeId` of the
3272 corresponding **context attribute**. The `DataType` of the **named attribute** MUST match, by URI equality,
3273 the `DataType` of the corresponding **context attribute**. If `Issuer` is supplied in the **named attribute**,
3274 then it MUST match, using the `urn:oasis:names:tc:xacml:1.0:function:string-equal` function, the `Issuer` of
3275 the corresponding **context attribute**. If `Issuer` is not supplied in the **named attribute**, then the
3276 matching of the **context attribute** to the **named attribute** SHALL be governed by `AttributeId` and
3277 `DataType` alone, regardless of the presence, absence, or actual value of `Issuer` in the corresponding
3278 **context attribute**. In the case of an attribute selector, the matching of the **attribute** to the **named**
3279 **attribute** SHALL be governed by the XPath expression and `DataType`.

3280 7.3.5 Attribute Retrieval

3281 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**. The
3282 **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document, but the
3283 **context handler** is responsible for obtaining and supplying the requested values by whatever means it
3284 deems appropriate. The **context handler** SHALL return the values of **attributes** that match the attribute
3285 designator or attribute selector and form them into a **bag** of values with the specified data-type. If no
3286 **attributes** from the request **context** match, then the **attribute** SHALL be considered missing. If the
3287 **attribute** is missing, then `MustBePresent` governs whether the attribute designator or attribute selector
3288 returns an empty **bag** or an "Indeterminate" result. If `MustBePresent` is "False" (default value), then a
3289 missing **attribute** SHALL result in an empty **bag**. If `MustBePresent` is "True", then a missing **attribute**
3290 SHALL result in "Indeterminate". This "Indeterminate" result SHALL be handled in accordance with the
3291 specification of the encompassing expressions, **rules**, **policies** and **policy sets**. If the result is
3292 "Indeterminate", then the `AttributeId`, `DataType` and `Issuer` of the **attribute** MAY be listed in the
3293 **authorization decision** as described in Section 7.15. However, a **PDP** MAY choose not to return such
3294 information for security reasons.

3295 7.3.6 Environment Attributes

3296 Standard **environment attributes** are listed in Section B.7. If a value for one of these **attributes** is
3297 supplied in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the
3298 **context handler** SHALL supply a value. In the case of date and time **attributes**, the supplied value
3299 SHALL have the semantics of the "date and time that apply to the **decision request**".

3300 7.3.7 AttributeSelector evaluation

3301 An <AttributeSelector> element will be evaluated according to the following processing model.

3302

3303 NOTE: It is not necessary for an implementation to actually follow these steps. It is only
3304 necessary to produce results identical to those that would be produced by following these
3305 steps.

- 3306 1. Construct an XML data structure suitable for xpath processing from the <Content> element in
3307 the **attributes** category given by the *Category* attribute. The data structure shall be constructed
3308 so that the document node of this structure contains a single document element which
3309 corresponds to the single child element of the <Content> element. The constructed data
3310 structure shall be equivalent to one that would result from parsing a stand-alone XML document
3311 consisting of the contents of the <Content> element (including any comment and processing-
3312 instruction markup). Namespace declarations which are not “visibly utilized”, as defined by [exc-
3313 c14n], MAY not be present and MUST NOT be utilized by the XPath expression in step 3. The
3314 data structure must meet the requirements of the applicable xpath version.
- 3315 2. Select a context node for xpath processing from this data structure. If there is a
3316 *ContextSelectorId* attribute, the context node shall be the node selected by applying the
3317 XPath expression given in the **attribute** value of the designated **attribute** (in the **attributes**
3318 category given by the <AttributeSelector> *Category* attribute). It shall be an error if this
3319 evaluation returns no node or more than one node, in which case the return value MUST be an
3320 “Indeterminate” with a status code “urn:oasis:names:tc:xacml:1.0:status:syntax-error”. If there is
3321 no *ContextSelectorId*, the document node of the data structure shall be the context node.
- 3322 3. Evaluate the XPath expression given in the *Path* attribute against the xml data structure, using
3323 the context node selected in the previous step. It shall be an error if this evaluation returns
3324 anything other than a sequence of nodes (possibly empty), in which case the
3325 <AttributeSelector> MUST return “Indeterminate” with a status code
3326 “urn:oasis:names:tc:xacml:1.0:status:syntax-error”.
- 3327 4. If the data type is a primitive data type, convert the text value of each selected node to the
3328 desired data type, as specified in the *DataType* attribute. Each value shall be constructed using
3329 the appropriate constructor function from [XF] Section 5 listed below, corresponding to the
3330 specified data type.

3331
3332 xs:string()
3333 xs:boolean()
3334 xs:integer()
3335 xs:double()
3336 xs:dateTime()
3337 xs:date()
3338 xs:time()
3339 xs:hexBinary()
3340 xs:base64Binary()
3341 xs:anyURI()
3342 xs:yearMonthDuration()
3343 xs:dayTimeDuration()
3344

3345 If the *DataType* is not one of the primitive types listed above, then the return values shall be
3346 constructed from the nodeset in a manner specified by the of the particular *DataType* extension
3347 specification. If the data type extension does not specify an appropriate constructor function, then
3348 the <AttributeSelector> MUST return “Indeterminate” with a status code
3349 “urn:oasis:names:tc:xacml:1.0:status:syntax-error”.
3350

3351 If an error occurs when converting the values returned by the XPath expression to the specified
3352 *DataType*, then the result of the <AttributeSelector> MUST be “Indeterminate”, with a
3353 status code “urn:oasis:names:tc:xacml:1.0:status:processing-error”

3354 7.4 Expression evaluation

3355 XACML specifies expressions in terms of the elements listed below, of which the <Apply> and
3356 <Condition> elements recursively compose greater expressions. Valid expressions SHALL be type
3357 correct, which means that the types of each of the elements contained within <Apply> elements SHALL
3358 agree with the respective argument types of the function that is named by the `FunctionId` attribute.
3359 The resultant type of the <Apply> element SHALL be the resultant type of the function, which MAY be
3360 narrowed to a primitive data-type, or a **bag** of a primitive data-type, by type-unification. XACML defines
3361 an evaluation result of "Indeterminate", which is said to be the result of an invalid expression, or an
3362 operational error occurring during the evaluation of the expression.

3363 XACML defines these elements to be in the substitution group of the <Expression> element:

- 3364 • <xacml:AttributeValue>
- 3365 • <xacml:AttributeDesignator>
- 3366 • <xacml:AttributeSelector>
- 3367 • <xacml:Apply>
- 3368 • <xacml:Condition>
- 3369 • <xacml:Function>
- 3370 • <xacml:VariableReference>

3371 7.5 Arithmetic evaluation

3372 IEEE 754 [IEEE754] specifies how to evaluate arithmetic functions in a context, which specifies defaults
3373 for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all integer and
3374 double functions relying on the Extended Default Context, enhanced with double precision:

3375 flags - all set to 0

3376 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap enabler,
3377 which SHALL be set to 1

3378 precision - is set to the designated double precision

3379 rounding - is set to round-half-even (IEEE 854 §4.1)

3380 7.6 Match evaluation

3381 The **attribute** matching element <Match> appears in the <Target> element of **rules**, **policies** and
3382 **policy sets**.

3383 This element represents a Boolean expression over **attributes** of the request **context**. A matching
3384 element contains a `MatchId` attribute that specifies the function to be used in performing the match
3385 evaluation, an <AttributeValue> and an <AttributeDesignator> or <AttributeSelector>
3386 element that specifies the **attribute** in the **context** that is to be matched against the specified value.

3387 The `MatchId` attribute SHALL specify a function that takes two arguments, returning a result type of
3388 "<http://www.w3.org/2001/XMLSchema#boolean>". The **attribute** value specified in the matching element
3389 SHALL be supplied to the `MatchId` function as its first argument. An element of the **bag** returned by the
3390 <AttributeDesignator> or <AttributeSelector> element SHALL be supplied to the `MatchId`
3391 function as its second argument, as explained below. The `DataType` of the <AttributeValue>
3392 SHALL match the data-type of the first argument expected by the `MatchId` function. The `DataType` of
3393 the <AttributeDesignator> or <AttributeSelector> element SHALL match the data-type of the
3394 second argument expected by the `MatchId` function.

3395 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the
3396 `MatchId` attribute, and those functions MAY use data-types that are also extensions, so long as the
3397 extension function returns a Boolean result and takes two single base types as its inputs. The function

3398 used as the value for the `MatchId` attribute SHOULD be easily indexable. Use of non-indexable or
3399 complex functions may prevent efficient evaluation of **decision requests**.

3400 The evaluation semantics for a matching element is as follows. If an operational error were to occur while
3401 evaluating the `<AttributeDesignator>` or `<AttributeSelector>` element, then the result of the
3402 entire expression SHALL be "Indeterminate". If the `<AttributeDesignator>` or
3403 `<AttributeSelector>` element were to evaluate to an empty **bag**, then the result of the expression
3404 SHALL be "False". Otherwise, the `MatchId` function SHALL be applied between the
3405 `<AttributeValue>` and each element of the **bag** returned from the `<AttributeDesignator>` or
3406 `<AttributeSelector>` element. If at least one of those function applications were to evaluate to
3407 "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the function
3408 applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally, if all function
3409 applications evaluate to "False", then the result of the entire expression SHALL be "False".

3410 It is also possible to express the semantics of a **target** matching element in a **condition**. For instance,
3411 the **target** match expression that compares a "**subject-name**" starting with the name "John" can be
3412 expressed as follows:

```
3413 <Match  
3414 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">  
3415   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3416     John.*  
3417   </AttributeValue>  
3418   <AttributeDesignator  
3419     Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-  
3420 subject"  
3421     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3422     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3423 </Match>
```

3424 Alternatively, the same match semantics can be expressed as an `<Apply>` element in a **condition** by
3425 using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```
3426 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
3427   <Function  
3428     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match"/>  
3429   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3430     John.*  
3431   </AttributeValue>  
3432   <AttributeDesignator  
3433     Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-  
3434 subject"  
3435     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3436     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3437 </Apply>
```

3438 7.7 Target evaluation

3439 An empty **target** matches any request. Otherwise the **target** value SHALL be "Match" if all the AnyOf
3440 specified in the **target** match values in the request **context**. Otherwise, if any one of the AnyOf specified
3441 in the **target** is "No Match", then the **target** SHALL be "No Match". Otherwise, the **target** SHALL be
3442 "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"

3443 Table 1 Target match table

3444 The AnyOf SHALL match values in the request **context** if at least one of their <AllOf> elements
 3445 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"

3446 Table 2 AnyOf match table

3447 An AllOf SHALL match a value in the request **context** if the value of all its <Match> elements is "True".

3448 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"

3449 Table 3 AllOf match table

3450 7.8 VariableReference Evaluation

3451 The <VariableReference> element references a single <VariableDefinition> element contained
 3452 within the same <Policy> element. A <VariableReference> that does not reference a particular
 3453 <VariableDefinition> element within the encompassing <Policy> element is called an undefined
 3454 reference. **Policies** with undefined references are invalid.

3455 In any place where a <VariableReference> occurs, it has the effect as if the text of the
 3456 <Expression> element defined in the <VariableDefinition> element replaces the
 3457 <VariableReference> element. Any evaluation scheme that preserves this semantic is acceptable.
 3458 For instance, the expression in the <VariableDefinition> element may be evaluated to a particular
 3459 value and cached for multiple references without consequence. (I.e. the value of an <Expression>
 3460 element remains the same for the entire **policy** evaluation.) This characteristic is one of the benefits of
 3461 XACML being a declarative language.

3462 A variable reference containing circular references is invalid. The PDP MUST detect circular references
 3463 either at policy loading time or during runtime evaluation. If the PDP detects a circular reference during
 3464 runtime the variable reference evaluates to "Indeterminate" with status code
 3465 urn:oasis:names:tc:xacml:1.0:status:processing-error.

3466 7.9 Condition evaluation

3467 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to "True".
 3468 Its value SHALL be "False" if the <Condition> element evaluates to "False". The **condition** value
 3469 SHALL be "Indeterminate", if the expression contained in the <Condition> element evaluates to
 3470 "Indeterminate."

3471 7.10 Rule evaluation

3472 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves separate
 3473 evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 4.

Target	Condition	Rule Value
"Match" or no target	"True"	Effect
"Match" or no target	"False"	"NotApplicable"
"Match" or no target	"Indeterminate"	"Indeterminate"
"No-match"	Don't care	"NotApplicable"
"Indeterminate"	Don't care	"Indeterminate"

3474 Table 4 Rule truth table.

3475 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable" or
 3476 "Indeterminate", respectively, regardless of the value of the **condition**. For these cases, therefore, the
 3477 **condition** need not be evaluated.

3478 If the **target** value is "Match", or there is no **target** in the **rule**, and the **condition** value is "True", then the
 3479 **effect** specified in the enclosing <Rule> element SHALL determine the **rule's** value.

3480 7.11 Policy evaluation

3481 The value of a **policy** SHALL be determined only by its contents, considered in relation to the contents of
 3482 the request **context**. A **policy's** value SHALL be determined by evaluation of the **policy's target** and
 3483 **rules**.

3484 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**
 3485 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the **policy's**
 3486 **rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to "No-match", then
 3487 the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to "Indeterminate", then the
 3488 value of the **policy** SHALL be "Indeterminate".

3489 The **policy** truth table is shown in Table 5.

Target	Rule values	Policy Value
"Match"	At least one rule value is its Effect	Specified by the rule- combining algorithm
"Match"	All rule values are "NotApplicable"	"NotApplicable"
"Match"	At least one rule value is "Indeterminate"	Specified by the rule- combining algorithm
"No-match"	Don't care	"NotApplicable"
"Indeterminate"	Don't care	"Indeterminate"

3490 Table 5 Policy truth table

3491 A **rules** value of "At least one **rule** value is its **Effect**" means either that the <Rule> element is absent, or
 3492 one or more of the **rules** contained in the **policy** is applicable to the **decision request** (i.e., it returns the
 3493 value of its "**Effect**"; see Section 7.10). A **rules** value of "All **rule** values are 'NotApplicable'" SHALL be
 3494 used if no **rule** contained in the **policy** is applicable to the request and if no **rule** contained in the **policy**
 3495 returns a value of "Indeterminate". If no **rule** contained in the **policy** is applicable to the request, but one
 3496 or more **rule** returns a value of "Indeterminate", then the **rules** SHALL evaluate to "At least one **rule** value
 3497 is 'Indeterminate'".

3498 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be "NotApplicable" or
 3499 "Indeterminate", respectively, regardless of the value of the **rules**. For these cases, therefore, the **rules**
 3500 need not be evaluated.

3501 If the **target** value is "Match" and the **rule** value is "At least one **rule** value is its **Effect**" or "At least one
 3502 **rule** value is 'Indeterminate'", then the **rule-combining algorithm** specified in the **policy** SHALL
 3503 determine the **policy** value.

3504 Note that none of the **rule-combining algorithms** defined by XACML 3.0 take parameters. However,
 3505 non-standard combining algorithms MAY take parameters. In such a case, the values of these
 3506 parameters associated with the **rules**, MUST be taken into account when evaluating the **policy**. The
 3507 parameters and their types should be defined in the specification of the combining algorithm. If the
 3508 implementation supports combiner parameters and if combiner parameters are present in a **policy**, then
 3509 the parameter values MUST be supplied to the combining algorithm implementation.

3510 7.12 Policy Set evaluation

3511 The value of a **policy set** SHALL be determined by its contents, considered in relation to the contents of
 3512 the request **context**. A **policy set**'s value SHALL be determined by evaluation of the **policy set**'s **target**,
 3513 **policies**, and **policy sets**, according to the specified **policy-combining algorithm**.

3514 The **policy set**'s **target** SHALL be evaluated to determine the applicability of the **policy set**. If the **target**
 3515 evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of the **policy**
 3516 **set**'s **policies** and **policy sets**, according to the specified **policy-combining algorithm**. If the **target**
 3517 evaluates to "No-match", then the value of the **policy set** shall be "NotApplicable". If the **target** evaluates
 3518 to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

3519 The **policy set** truth table is shown in Table 6.

Target	Policy values	Policy set Value
"Match"	At least one policy value is its Decision	Specified by the policy-combining algorithm
"Match"	All policy values are "NotApplicable"	"NotApplicable"
"Match"	At least one policy value is "Indeterminate"	Specified by the policy-combining algorithm
"No-match"	Don't care	"NotApplicable"
"Indeterminate"	Don't care	"Indeterminate"

3520 Table 6 Policy set truth table

3521 A **policies** value of "At least one **policy** value is its Decision" SHALL be used if there are no contained or
 3522 referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets** contained in or
 3523 referenced by the **policy set** is applicable to the **decision request** (i.e., returns a value determined by its
 3524 combining algorithm) A **policies** value of "All **policy** values are 'NotApplicable'" SHALL be used if no
 3525 **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request and if no
 3526 **policy** or **policy set** contained in or referenced by the **policy set** returns a value of "Indeterminate". If no
 3527 **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request but one or
 3528 more **policy** or **policy set** returns a value of "Indeterminate", then the **policies** SHALL evaluate to "At
 3529 least one **policy** value is 'Indeterminate'".

3530 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be "NotApplicable"
 3531 or "Indeterminate", respectively, regardless of the value of the **policies**. For these cases, therefore, the
 3532 **policies** need not be evaluated.

3533 If the **target** value is “Match” and the **policies** value is “At least one **policy** value is its Decision” or “At
3534 least one **policy** value is ‘Indeterminate’”, then the **policy-combining algorithm** specified in the **policy**
3535 **set** SHALL determine the **policy set** value.

3536 Note that none of the **policy-combining algorithms** defined by XACML 3.0 take parameters. However,
3537 non-standard combining algorithms MAY take parameters. In such a case, the values of these
3538 parameters associated with the **policies**, MUST be taken into account when evaluating the **policy set**.
3539 The parameters and their types should be defined in the specification of the combining algorithm. If the
3540 implementation supports combiner parameters and if combiner parameters are present in a **policy**, then
3541 the parameter values MUST be supplied to the combining algorithm implementation.

3542 7.13 PolicySetIdReference and PolicyIdReference evaluation

3543 A policy set id reference or a policy id reference is evaluated by resolving the reference and evaluating
3544 the referenced policy set or policy.

3545 If resolving the reference fails, the reference evaluates to “Indeterminate” with status code
3546 urn:oasis:names:tc:xacml:1.0:status:processing-error.

3547 A policy set id reference or a policy id reference containing circular references is invalid. The PDP MUST
3548 detect circular references either at policy loading time or during runtime evaluation. If the PDP detects a
3549 circular reference during runtime the reference evaluates to “Indeterminate” with status code
3550 urn:oasis:names:tc:xacml:1.0:status:processing-error.

3551 7.14 Hierarchical resources

3552 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document). XACML
3553 provides several optional mechanisms for supporting hierarchical **resources**. These are described in the
3554 XACML Profile for Hierarchical Resources [**Hier**] and in the XACML Profile for Requests for Multiple
3555 Resources [**Multi**].

3556 7.15 Authorization decision

3557 In relation to a particular **decision request**, the **PDP** is defined by a **policy-combining algorithm** and a
3558 set of **policies** and/or **policy sets**. The **PDP** SHALL return a response **context** as if it had evaluated a
3559 single **policy set** consisting of this **policy-combining algorithm** and the set of **policies** and/or **policy**
3560 **sets**.

3561 The **PDP** MUST evaluate the **policy set** as specified in Sections 5 and 7. The **PDP** MUST return a
3562 response **context**, with one <Decision> element of value "Permit", "Deny", "Indeterminate" or
3563 "NotApplicable".

3564 If the **PDP** cannot make a **decision**, then an "Indeterminate" <Decision> element SHALL be returned.

3565 7.16 Obligations and advice

3566 A **rule**, **policy**, or **policy set** may contain one or more **obligation** or **advice** expressions. When such a
3567 **rule**, **policy**, or **policy set** is evaluated, the **obligation** or **advice** expression SHALL be evaluated to an
3568 **obligation** or **advice** respectively, which SHALL be passed up to the next level of evaluation (the
3569 enclosing or referencing **policy**, **policy set**, or **authorization decision**) only if the **effect** of the **rule**,
3570 **policy**, or **policy set** being evaluated matches the value of the FulfillOn attribute of the **obligation** or
3571 the AppliesTo attribute of the **advice**. If any of the **attribute** assignment expressions in an **obligation**
3572 or **advice** expression with a matching FulfillOn or AppliesTo attribute evaluates to “Indeterminate”,
3573 then the whole **rule**, **policy**, or **policy set** SHALL be “Indeterminate”. If the FulfillOn or AppliesTo
3574 attribute does not match the result of the combining algorithm or the **rule** evaluation, then any
3575 indeterminate in an **obligation** or **advice** expression has no effect.

3576 As a consequence of this procedure, no **obligations** or **advice** SHALL be returned to the **PEP** if the **rule**,
3577 **policies**, or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is
3578 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **rule**, **policy**, or **policy**
3579 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3580 If the *PDP*'s evaluation is viewed as a tree of *rules*, *policy sets* and *policies*, each of which returns
3581 "Permit" or "Deny", then the set of *obligations* and *advice* returned by the *PDP* to the *PEP* will include
3582 only the *obligations* and *advice* associated with those paths where the *effect* at each level of evaluation
3583 is the same as the *effect* being returned by the *PDP*. In situations where any lack of determinism is
3584 unacceptable, a deterministic combining algorithm, such as ordered-deny-overrides, should be used.
3585 Also see Section 7.2.

3586 7.17 Exception handling

3587 XACML specifies behavior for the *PDP* in the following situations.

3588 7.17.1 Unsupported functionality

3589 If the *PDP* attempts to evaluate a *policy set* or *policy* that contains an optional element type or function
3590 that the *PDP* does not support, then the *PDP* SHALL return a <Decision> value of "Indeterminate". If a
3591 <StatusCode> element is also returned, then its value SHALL be
3592 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and
3593 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported function.

3594 7.17.2 Syntax and type errors

3595 If a *policy* that contains invalid syntax is evaluated by the XACML *PDP* at the time a *decision request* is
3596 received, then the result of that *policy* SHALL be "Indeterminate" with a *StatusCode* value of
3597 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3598 If a *policy* that contains invalid static data-types is evaluated by the XACML *PDP* at the time a *decision*
3599 *request* is received, then the result of that *policy* SHALL be "Indeterminate" with a *StatusCode* value of
3600 "urn:oasis:names:tc:xacml:1.0:status:processing-error".

3601 7.17.3 Missing attributes

3602 The absence of matching *attributes* in the request *context* for any of the attribute designators attribute or
3603 selectors that are found in the *policy* will result in an enclosing <AllOf> element to return a value of
3604 "Indeterminate", if the designator or selector has the *MustBePresent* XML attribute set to true, as
3605 described in Sections 5.29 and 5.30 and may result in a <Decision> element containing the
3606 "Indeterminate" value. If, in this case, and a status code is supplied, then the value

3607 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3608 SHALL be used, to indicate that more information is needed in order for a definitive *decision* to be
3609 rendered. In this case, the <Status> element MAY list the names and data-types of any *attributes* that
3610 are needed by the *PDP* to refine its *decision* (see Section 5.58). A *PEP* MAY resubmit a refined request
3611 *context* in response to a <Decision> element contents of "Indeterminate" with a status code of

3612 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3613 by adding *attribute* values for the *attribute* names that were listed in the previous response. When the
3614 *PDP* returns a <Decision> element contents of "Indeterminate", with a status code of

3615 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute",

3616 it MUST NOT list the names and data-types of any *attribute* for which values were supplied in the original
3617 request. Note, this requirement forces the *PDP* to eventually return an *authorization decision* of
3618 "Permit", "Deny", or "Indeterminate" with some other status code, in response to successively-refined
3619 requests.

3620 8 XACML extensibility points (non-normative)

3621 This section describes the points within the XACML model and schema where extensions can be added.

3622 8.1 Extensible XML attribute types

3623 The following XML attributes have values that are URIs. These may be extended by the creation of new
3624 URIs associated with new semantics for these attributes.

3625 *Category*,

3626 *AttributeId*,

3627 *DataType*,

3628 *FunctionId*,

3629 *MatchId*,

3630 *ObligationId*,

3631 *AdviceId*,

3632 *PolicyCombiningAlgId*,

3633 *RuleCombiningAlgId*,

3634 *StatusCode*,

3635 *SubjectCategory*.

3636 See Section 5 for definitions of these *attribute* types.

3637 8.2 Structured attributes

3638 <AttributeValue> elements MAY contain an instance of a structured XML data-type. Section 7.3.1
3639 describes a number of standard techniques to identify data items within such a structured *attribute*.
3640 Listed here are some additional techniques that require XACML extensions.

- 3641 1. For a given structured data-type, a community of XACML users MAY define new *attribute*
3642 identifiers for each leaf sub-element of the structured data-type that has a type conformant with
3643 one of the XACML-defined primitive data-types. Using these new *attribute* identifiers, the *PEPs*
3644 or *context handlers* used by that community of users can flatten instances of the structured
3645 data-type into a sequence of individual <Attribute> elements. Each such <Attribute>
3646 element can be compared using the XACML-defined functions. Using this method, the structured
3647 data-type itself never appears in an <AttributeValue> element.
- 3648 2. A community of XACML users MAY define a new function that can be used to compare a value of
3649 the structured data-type against some other value. This method may only be used by *PDPs* that
3650 support the new function.

3651 9 Security and privacy considerations (non- 3652 normative)

3653 This section identifies possible security and privacy compromise scenarios that should be considered
3654 when implementing an XACML-based system. The section is informative only. It is left to the
3655 implementer to decide whether these compromise scenarios are practical in their environment and to
3656 select appropriate safeguards.

3657 9.1 Threat model

3658 We assume here that the adversary has access to the communication channel between the XACML
3659 actors and is able to interpret, insert, delete, and modify messages or parts of messages.

3660 Additionally, an actor may use information from a former message maliciously in subsequent transactions.
3661 It is further assumed that *rules* and *policies* are only as reliable as the actors that create and use them.
3662 Thus it is incumbent on each actor to establish appropriate trust in the other actors upon which it relies.
3663 Mechanisms for trust establishment are outside the scope of this specification.

3664 The messages that are transmitted between the actors in the XACML model are susceptible to attack by
3665 malicious third parties. Other points of vulnerability include the *PEP*, the *PDP*, and the *PAP*. While some
3666 of these entities are not strictly within the scope of this specification, their compromise could lead to the
3667 compromise of *access control* enforced by the *PEP*.

3668 It should be noted that there are other components of a distributed system that may be compromised,
3669 such as an operating system and the domain-name system (DNS) that are outside the scope of this
3670 discussion of threat models. Compromise in these components may also lead to a policy violation.

3671 The following sections detail specific compromise scenarios that may be relevant to an XACML system.

3672 9.1.1 Unauthorized disclosure

3673 XACML does not specify any inherent mechanisms to protect the confidentiality of the messages
3674 exchanged between actors. Therefore, an adversary could observe the messages in transit. Under
3675 certain security *policies*, disclosure of this information is a violation. Disclosure of *attributes* or the types
3676 of *decision requests* that a *subject* submits may be a breach of privacy policy. In the commercial
3677 sector, the consequences of unauthorized disclosure of personal data may range from embarrassment to
3678 the custodian, to imprisonment and/or large fines in the case of medical or financial data.

3679 Unauthorized disclosure is addressed by confidentiality safeguards.

3680 9.1.2 Message replay

3681 A message replay attack is one in which the adversary records and replays legitimate messages between
3682 XACML actors. This attack may lead to denial of service, the use of out-of-date information or
3683 impersonation.

3684 Prevention of replay attacks requires the use of message freshness safeguards.

3685 Note that encryption of the message does not mitigate a replay attack since the message is simply
3686 replayed and does not have to be understood by the adversary.

3687 9.1.3 Message insertion

3688 A message insertion attack is one in which the adversary inserts messages in the sequence of messages
3689 between XACML actors.

3690 The solution to a message insertion attack is to use mutual authentication and message sequence
3691 integrity safeguards between the actors. It should be noted that just using SSL mutual authentication is
3692 not sufficient. This only proves that the other party is the one identified by the *subject* of the X.509

3693 certificate. In order to be effective, it is necessary to confirm that the certificate **subject** is authorized to
3694 send the message.

3695 9.1.4 Message deletion

3696 A message deletion attack is one in which the adversary deletes messages in the sequence of messages
3697 between XACML actors. Message deletion may lead to denial of service. However, a properly designed
3698 XACML system should not render an incorrect **authorization decision** as a result of a message deletion
3699 attack.

3700 The solution to a message deletion attack is to use message sequence integrity safeguards between the
3701 actors.

3702 9.1.5 Message modification

3703 If an adversary can intercept a message and change its contents, then they may be able to alter an
3704 **authorization decision**. A message integrity safeguard can prevent a successful message modification
3705 attack.

3706 9.1.6 NotApplicable results

3707 A result of "NotApplicable" means that the **PDP** could not locate a **policy** whose **target** matched the
3708 information in the **decision request**. In general, it is highly recommended that a "Deny" **effect policy** be
3709 used, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is returned instead.

3710 In some security models, however, such as those found in many web servers, an **authorization decision**
3711 of "NotApplicable" is treated as equivalent to "Permit". There are particular security considerations that
3712 must be taken into account for this to be safe. These are explained in the following paragraphs.

3713 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the **policy** to
3714 match elements in the **decision request** be closely aligned with the data syntax used by the applications
3715 that will be submitting the **decision request**. A failure to match will result in "NotApplicable" and be
3716 treated as "Permit". So an unintended failure to match may allow unintended **access**.

3717 Commercial http responders allow a variety of syntaxes to be treated equivalently. The "%" can be used
3718 to represent characters by hex value. The URL path "/./" provides multiple ways of specifying the same
3719 value. Multiple character sets may be permitted and, in some cases, the same printed character can be
3720 represented by different binary values. Unless the matching algorithm used by the **policy** is sophisticated
3721 enough to catch these variations, unintended **access** may be permitted.

3722 It may be safe to treat "NotApplicable" as "Permit" only in a closed environment where all applications that
3723 formulate a **decision request** can be guaranteed to use the exact syntax expected by the **policies**. In a
3724 more open environment, where **decision requests** may be received from applications that use any legal
3725 syntax, it is strongly recommended that "NotApplicable" NOT be treated as "Permit" unless matching
3726 **rules** have been very carefully designed to match all possible applicable inputs, regardless of syntax or
3727 type variations. Note, however, that according to Section 7.2, a **PEP** must deny **access** unless it
3728 receives an explicit "Permit" **authorization decision**.

3729 9.1.7 Negative rules

3730 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care, negative
3731 **rules** can lead to policy violations, therefore some authorities recommend that they not be used.
3732 However, negative **rules** can be extremely efficient in certain cases, so XACML has chosen to include
3733 them. Nevertheless, it is recommended that they be used with care and avoided if possible.

3734 A common use for negative **rules** is to deny **access** to an individual or subgroup when their membership
3735 in a larger group would otherwise permit them **access**. For example, we might want to write a **rule** that
3736 allows all vice presidents to see the unpublished financial data, except for Joe, who is only a ceremonial
3737 vice president and can be indiscreet in his communications. If we have complete control over the
3738 administration of **subject attributes**, a superior approach would be to define "Vice President" and
3739 "Ceremonial Vice President" as distinct groups and then define **rules** accordingly. However, in some

3740 environments this approach may not be feasible. (It is worth noting in passing that referring to individuals
3741 in **rules** does not scale well. Generally, shared **attributes** are preferred.)

3742 If not used with care, negative **rules** can lead to policy violations in two common cases: when **attributes**
3743 are suppressed and when the base group changes. An example of suppressed **attributes** would be if we
3744 have a **policy** that **access** should be permitted, unless the **subject** is a credit risk. If it is possible that
3745 the **attribute** of being a credit risk may be unknown to the **PDP** for some reason, then unauthorized
3746 **access** may result. In some environments, the **subject** may be able to suppress the publication of
3747 **attributes** by the application of privacy controls, or the server or repository that contains the information
3748 may be unavailable for accidental or intentional reasons.

3749 An example of a changing base group would be if there is a **policy** that everyone in the engineering
3750 department may change software source code, except for secretaries. Suppose now that the department
3751 was to merge with another engineering department and the intent is to maintain the same **policy**.
3752 However, the new department also includes individuals identified as administrative assistants, who ought
3753 to be treated in the same way as secretaries. Unless the **policy** is altered, they will unintentionally be
3754 permitted to change software source code. Problems of this type are easy to avoid when one individual
3755 administers all **policies**, but when administration is distributed, as XACML allows, this type of situation
3756 must be explicitly guarded against.

3757 9.1.8 Denial of service

3758 A denial of service attack is one in which the adversary overloads an XACML actor with excessive
3759 computations or network traffic such that legitimate users cannot access the services provided by the
3760 actor.

3761 The urn:oasis:names:tc:xacml:3.0:function:access-permitted function may lead to hard to predict behavior
3762 in the **PDP**. It is possible that the function is invoked during the recursive invocations of the **PDP** such that
3763 loops are formed. Such loops may in some cases lead to large numbers of requests to be generated
3764 before the **PDP** can detect the loop and abort evaluation. Such loops could cause a denial of service at
3765 the **PDP**, either because of a malicious **policy** or because of a mistake in a **policy**.

3766 9.2 Safeguards

3767 9.2.1 Authentication

3768 Authentication provides the means for one party in a transaction to determine the identity of the other
3769 party in the transaction. Authentication may be in one direction, or it may be bilateral.

3770 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the
3771 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an adversary
3772 could provide false or invalid **authorization decisions**, leading to a policy violation.

3773 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust to
3774 determine what, if any, sensitive data should be passed. One should keep in mind that even simple
3775 "Permit" or "Deny" responses could be exploited if an adversary were allowed to make unlimited requests
3776 to a **PDP**.

3777 Many different techniques may be used to provide authentication, such as co-located code, a private
3778 network, a VPN, or digital signatures. Authentication may also be performed as part of the
3779 communication protocol used to exchange the **contexts**. In this case, authentication may be performed
3780 either at the message level or at the session level.

3781 9.2.2 Policy administration

3782 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects** may
3783 use this information to determine how to gain unauthorized **access**.

3784 To prevent this threat, the repository used for the storage of **policies** may itself require **access control**.
3785 In addition, the <Status> element should be used to return values of missing **attributes** only when
3786 exposure of the identities of those **attributes** will not compromise security.

3787 9.2.3 Confidentiality

3788 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired
3789 recipients and not by anyone else who encounters the message while it is in transit. There are two areas
3790 in which confidentiality should be considered: one is confidentiality during transmission; the other is
3791 confidentiality within a <Policy> element.

3792 9.2.3.1 Communication confidentiality

3793 In some environments it is deemed good practice to treat all data within an **access control** system as
3794 confidential. In other environments, **policies** may be made freely available for distribution, inspection,
3795 and audit. The idea behind keeping **policy** information secret is to make it more difficult for an adversary
3796 to know what steps might be sufficient to obtain unauthorized **access**. Regardless of the approach
3797 chosen, the security of the **access control** system should not depend on the secrecy of the **policy**.

3798 Any security considerations related to transmitting or exchanging XACML <Policy> elements are
3799 outside the scope of the XACML standard. While it is important to ensure that the integrity and
3800 confidentiality of <Policy> elements is maintained when they are exchanged between two parties, it is
3801 left to the implementers to determine the appropriate mechanisms for their environment.

3802 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL. Using a
3803 point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points is
3804 compromised.

3805 9.2.3.2 Statement level confidentiality

3806 In some cases, an implementation may want to encrypt only parts of an XACML <Policy> element.

3807 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used to
3808 encrypt all or parts of an XML document. This specification is recommended for use with XACML.

3809 It should go without saying that if a repository is used to facilitate the communication of cleartext (i.e.,
3810 unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to store this
3811 sensitive data.

3812 9.2.4 Policy integrity

3813 The XACML **policy** used by the **PDP** to evaluate the request **context** is the heart of the system.
3814 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of the
3815 **policy**. One is to ensure that <Policy> elements have not been altered since they were originally
3816 created by the **PAP**. The other is to ensure that <Policy> elements have not been inserted or deleted
3817 from the set of **policies**.

3818 In many cases, both aspects can be achieved by ensuring the integrity of the actors and implementing
3819 session-level mechanisms to secure the communication between actors. The selection of the appropriate
3820 mechanisms is left to the implementers. However, when **policy** is distributed between organizations to
3821 be acted on at a later time, or when the **policy** travels with the protected **resource**, it would be useful to
3822 sign the **policy**. In these cases, the XML Signature Syntax and Processing standard from W3C is
3823 recommended to be used with XACML.

3824 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures should
3825 not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not request a **policy**
3826 based on who signed it or whether or not it has been signed (as such a basis for selection would, itself,
3827 be a matter of policy). However, the **PDP** must verify that the key used to sign the **policy** is one
3828 controlled by the purported **issuer** of the **policy**. The means to do this are dependent on the specific
3829 signature technology chosen and are outside the scope of this document.

3830 9.2.5 Policy identifiers

3831 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure that
3832 these are unique. Confusion between identifiers could lead to misidentification of the **applicable policy**.

3833 This specification is silent on whether a **PAP** must generate a new identifier when a **policy** is modified or
3834 may use the same identifier in the modified **policy**. This is a matter of administrative practice. However,
3835 care must be taken in either case. If the identifier is reused, there is a danger that other **policies** or
3836 **policy sets** that reference it may be adversely affected. Conversely, if a new identifier is used, these
3837 other **policies** may continue to use the prior **policy**, unless it is deleted. In either case the results may
3838 not be what the **policy** administrator intends.

3839 If a **PDP** is provided with **policies** from distinct sources which might not be fully trusted, as in the use of
3840 the administration profile [**XACMLAdmin**], there is a concern that someone could intentionally publish a
3841 **policy** with an id which collides with another **policy**. This could cause **policy** references that point to the
3842 wrong **policy**, and may cause other unintended consequences in an implementation which is predicated
3843 upon having unique **policy** identifiers.

3844 If this issue is a concern it is RECOMMENDED that distinct **policy** issuers or sources are assigned
3845 distinct namespaces for **policy** identifiers. One method is to make sure that the **policy** identifier begins
3846 with a string which has been assigned to the particular **policy** issuer or source. The remainder of the
3847 **policy** identifier is an issuer-specific unique part. For instance, Alice from Example Inc. could be assigned
3848 the **policy** identifiers which begin with http://example.com/xacml/policyId/alice/. The **PDP** or another
3849 trusted component can then verify that the authenticated source of the **policy** is Alice at Example Inc, or
3850 otherwise reject the **policy**. Anyone else will be unable to publish **policies** with identifiers which collide
3851 with the **policies** of Alice.

3852 9.2.6 Trust model

3853 Discussions of authentication, integrity and confidentiality safeguards necessarily assume an underlying
3854 trust model: how can one actor come to believe that a given key is uniquely associated with a specific,
3855 identified actor so that the key can be used to encrypt data for that actor or verify signatures (or other
3856 integrity structures) from that actor? Many different types of trust models exist, including strict
3857 hierarchies, distributed authorities, the Web, the bridge, and so on.

3858 It is worth considering the relationships between the various actors of the **access control** system in terms
3859 of the interdependencies that do and do not exist.

- 3860 • None of the entities of the authorization system are dependent on the **PEP**. They may collect data
3861 from it, (for example authentication data) but are responsible for verifying it themselves.
- 3862 • The correct operation of the system depends on the ability of the **PEP** to actually enforce **policy**
3863 **decisions**.
- 3864 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the **PDP** is
3865 supplied with the correct inputs. Other than that, the **PDP** does not depend on the **PEP**.
- 3866 • The **PDP** depends on the **PAP** to supply appropriate **policies**. The **PAP** is not dependent on other
3867 components.

3868 9.2.7 Privacy

3869 It is important to be aware that any transactions that occur with respect to **access control** may reveal
3870 private information about the actors. For example, if an XACML **policy** states that certain data may only
3871 be read by **subjects** with "Gold Card Member" status, then any transaction in which a **subject** is
3872 permitted **access** to that data leaks information to an adversary about the **subject's** status. Privacy
3873 considerations may therefore lead to encryption and/or to **access control** requirements surrounding the
3874 enforcement of XACML **policy** instances themselves: confidentiality-protected channels for the
3875 request/response protocol messages, protection of **subject attributes** in storage and in transit, and so
3876 on.

3877 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope of
3878 XACML. The **decision** regarding whether, how, and when to deploy such mechanisms is left to the
3879 implementers associated with the environment.

3880 **9.3 Unicode security issues**

3881 There are many security considerations related to use of Unicode. An XACML implementation SHOULD
3882 follow the advice given in the relevant version of **[UTR36]**.

3883 10 Conformance

3884 10.1 Introduction

3885 The XACML specification addresses the following aspect of conformance:

3886 The XACML specification defines a number of functions, etc. that have somewhat special applications,
3887 therefore they are not required to be implemented in an implementation that claims to conform with the
3888 OASIS standard.

3889 10.2 Conformance tables

3890 This section lists those portions of the specification that **MUST** be included in an implementation of a **PDP**
3891 that claims to conform to XACML v3.0. A set of test cases has been created to assist in this process.
3892 These test cases can be located from the OASIS XACML TC Web page. The site hosting the test cases
3893 contains a full description of the test cases and how to execute them.

3894 Note: "M" means mandatory-to-implement. "O" means optional.

3895 The implementation **MUST** follow sections 5, 6, 7, A, B and C where they apply to implemented items in
3896 the following tables.

3897 10.2.1 Schema elements

3898 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

3899 **10.2.2 Identifier Prefixes**

3900 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

3901 **10.2.3 Algorithms**

3902 The implementation MUST include the **rule-** and **policy-combining algorithms** associated with the
3903 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

3904 10.2.4 Status Codes

3905 Implementation support for the <StatusCode> element is optional, but if the element is supported, then
3906 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

3907 10.2.5 Attributes

3908 The implementation MUST support the **attributes** associated with the following identifiers as specified by
3909 XACML. If values for these **attributes** are not present in the **decision request**, then their values MUST
3910 be supplied by the **context handler**. So, unlike most other **attributes**, their semantics are not
3911 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

3912 10.2.6 Identifiers

3913 The implementation MUST use the **attributes** associated with the following identifiers in the way XACML
3914 has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that uses XACML,
3915 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

3916 10.2.7 Data-types

3917 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

3918 10.2.8 Functions

3919 The implementation MUST properly process those functions associated with the identifiers marked with
3920 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: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: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-is-in	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-is-in	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: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: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:1.0: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: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: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

3921 **10.2.9 Identifiers planned for future deprecation**

3922 These identifiers are associated with previous versions of XACML and newer alternatives exist in XACML
 3923 3.0. They are planned to be deprecated at some unspecified point in the future. It is RECOMMENDED
 3924 that these identifiers not be used in new policies and requests.

3925 The implementation MUST properly process those features associated with the identifiers marked with an
 3926 "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

3927

3928 A. Data-types and functions (normative)

3929 A.1 Introduction

3930 This section specifies the data-types and functions used in XACML to create *predicates* for *conditions*
3931 and *target* matches.

3932 This specification combines the various standards set forth by IEEE and ANSI for string representation of
3933 numeric values, as well as the evaluation of arithmetic functions. It describes the primitive data-types and
3934 *bags*. The standard functions are named and their operational semantics are described.

3935 A.2 Data-types

3936 Although XML instances represent all data-types as strings, an XACML *PDP* must operate on types of
3937 data that, while they have string representations, are not just strings. Types such as Boolean, integer,
3938 and double MUST be converted from their XML string representations to values that can be compared
3939 with values in their domain of discourse, such as numbers. The following primitive data-types are
3940 specified for use with XACML and have explicit data representations:

- 3941 • <http://www.w3.org/2001/XMLSchema#string>
- 3942 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3943 • <http://www.w3.org/2001/XMLSchema#integer>
- 3944 • <http://www.w3.org/2001/XMLSchema#double>
- 3945 • <http://www.w3.org/2001/XMLSchema#time>
- 3946 • <http://www.w3.org/2001/XMLSchema#date>
- 3947 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3948 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3949 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3950 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3951 • <http://www.w3.org/2001/XMLSchema#dayTimeDuration>
- 3952 • <http://www.w3.org/2001/XMLSchema#yearMonthDuration>
- 3953 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3954 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>
- 3955 • <urn:oasis:names:tc:xacml:2.0:data-type:ipAddress>
- 3956 • <urn:oasis:names:tc:xacml:2.0:data-type:dnsName>
- 3957 • <urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression>

3958 For the sake of improved interoperability, it is RECOMMENDED that all time references be in UTC time.

3959 An XACML *PDP* SHALL be capable of converting string representations into various primitive data-types.
3960 For doubles, XACML SHALL use the conversions described in [IEEE754].

3961 XACML defines four data-types representing identifiers for *subjects* or *resources*; these are:

3962 “urn:oasis:names:tc:xacml:1.0:data-type:x500Name”,

3963 “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”

3964 “urn:oasis:names:tc:xacml:2.0:data-type:ipAddress” and

3965 “urn:oasis:names:tc:xacml:2.0:data-type:dnsName”

3966 These types appear in several standard applications, such as TLS/SSL and electronic mail.

3967 X.500 directory name

3968 The "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" primitive type represents an ITU-T Rec.
3969 X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253
3970 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished
3971 Names".

3972 **RFC 822 name**

3973 The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents an electronic
3974 mail address. The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2,
3975 Command Argument Syntax, under the term "Mailbox".

3976 **IP address**

3977 The "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" primitive type represents an IPv4 or IPv6
3978 network address, with optional mask and optional port or port range. The syntax SHALL be:

3979 ipAddress = address ["/" mask] [":" [portrange]]

3980 For an IPv4 address, the address and mask are formatted in accordance with the syntax for a
3981 "host" in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2.

3982 For an IPv6 address, the address and mask are formatted in accordance with the syntax for an
3983 "ipv6reference" in IETF RFC 2732 "Format for Literal IPv6 Addresses in URL's". (Note that an
3984 IPv6 address or mask, in this syntax, is enclosed in literal "[" "]" brackets.)

3985 **DNS name**

3986 The "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" primitive type represents a Domain
3987 Name Service (DNS) host name, with optional port or port range. The syntax SHALL be:

3988 dnsName = hostname [":" portrange]

3989 The hostname is formatted in accordance with IETF RFC 2396 "Uniform Resource Identifiers
3990 (URI): Generic Syntax", section 3.2, except that a wildcard "*" may be used in the left-most
3991 component of the hostname to indicate "any subdomain" under the domain specified to its right.

3992 For both the "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and
3993 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" data-types, the port or port range syntax
3994 SHALL be

3995 portrange = portnumber | "-"portnumber | portnumber "-"portnumber

3996 where "portnumber" is a decimal port number. If the port number is of the form "-x", where "x" is
3997 a port number, then the range is all ports numbered "x" and below. If the port number is of the
3998 form "x-", then the range is all ports numbered "x" and above. [This syntax is taken from the Java
3999 SocketPermission.]

4000 **XPath expression**

4001 The "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" primitive type represents an
4002 XPath expression selects over the XML in a <Content> element. The syntax is defined by the
4003 XPath W3C recommendation. The content of this data type also includes the context in which
4004 namespaces prefixes in the expression are resolved, which distinguishes it from a plain string and
4005 the XACML **attribute** category of the <Content> element to which it applies. When the value is
4006 encoded in an <AttributeValue> element, the namespace context is given by the
4007 <AttributeValue> element and an XML attribute called XPathCategory gives the category of
4008 the <Content> element where the expression applies.

4009 The XPath expression MUST be evaluated in a context which is equivalent of a stand alone XML
4010 document with the only child of the <Content> element as the document element. Namespace
4011 declarations which are not "visibly utilized", as defined by [exc-c14n], MAY not be present and
4012 MUST NOT be utilized by the XPath expression. The context node of the XPath expression is the
4013 document node of this stand alone document.

4014 A.3 Functions

4015 XACML specifies the following functions. If an argument of one of these functions were to evaluate to
4016 "Indeterminate", then the function SHALL be set to "Indeterminate".

4017 A.3.1 Equality predicates

4018 The following functions are the equality functions for the various primitive types. Each function for a
4019 particular data-type follows a specified standard convention for that data-type.

- 4020 • urn:oasis:names:tc:xacml:1.0:function:string-equal
4021 This function SHALL take two arguments of data-type
4022 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4023 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if
4024 the value of both of its arguments are of equal length and each string is determined to be equal.
4025 Otherwise, it SHALL return "False". The comparison SHALL use Unicode codepoint collation, as
4026 defined for the identifier <http://www.w3.org/2005/xpath-functions/collation/codepoint> by **[XF]**.
- 4027 • urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case
4028 This function SHALL take two arguments of data-type
4029 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4030 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be "True" if and only if the
4031 two strings are equal as defined by urn:oasis:names:tc:xacml:1.0:function:string-equal after they
4032 have both been converted to lower case with urn:oasis:names:tc:xacml:1.0:function:string-
4033 normalize-to-lower-case.
- 4034 • urn:oasis:names:tc:xacml:1.0:function:boolean-equal
4035 This function SHALL take two arguments of data-type
4036 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return an
4037 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if
4038 the arguments are equal. Otherwise, it SHALL return "False".
- 4039 • urn:oasis:names:tc:xacml:1.0:function:integer-equal
4040 This function SHALL take two arguments of data-type
4041 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an
4042 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if
4043 the two arguments represent the same number.
- 4044 • urn:oasis:names:tc:xacml:1.0:function:double-equal
4045 This function SHALL take two arguments of data-type
4046 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an
4047 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on doubles
4048 according to IEEE 754 **[IEEE754]**.
- 4049 • urn:oasis:names:tc:xacml:1.0:function:date-equal
4050 This function SHALL take two arguments of data-type
4051 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4052 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to
4053 the "op:date-equal" function **[XF]** Section 10.4.9.
- 4054 • urn:oasis:names:tc:xacml:1.0:function:time-equal
4055 This function SHALL take two arguments of data-type
4056 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
4057 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to
4058 the "op:time-equal" function **[XF]** Section 10.4.12.
- 4059 • urn:oasis:names:tc:xacml:1.0:function:dateTime-equal

- 4060 This function SHALL take two arguments of data-type
 4061 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
 4062 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to
 4063 the "op:dateTime-equal" function [XF] Section 10.4.6.
- 4064 • urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal

4065 This function SHALL take two arguments of data-type
 4066 "http://www.w3.org/2001/XMLSchema#dayTimeDuration" and SHALL return an
 4067 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
 4068 according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical
 4069 representation of each argument MUST be converted to a value expressed in fractional seconds
 4070 [XF] Section 10.3.2.
 - 4071 • urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal

4072 This function SHALL take two arguments of data-type
 4073 "http://www.w3.org/2001/XMLSchema#yearMonthDuration" and SHALL return an
 4074 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation
 4075 according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical
 4076 representation of each argument MUST be converted to a value expressed in fractional seconds
 4077 [XF] Section 10.3.2.
 - 4078 • urn:oasis:names:tc:xacml:1.0:function:anyURI-equal

4079 This function SHALL take two arguments of data-type
 4080 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an
 4081 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if
 4082 the values of the two arguments are equal on a codepoint-by-codepoint basis.
 - 4083 • urn:oasis:names:tc:xacml:1.0:function:x500Name-equal

4084 This function SHALL take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name"
 4085 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if
 4086 and only if each Relative Distinguished Name (RDN) in the two arguments matches. Otherwise,
 4087 it SHALL return "False". Two RDNs shall be said to match if and only if the result of the following
 4088 operations is "True".

 - 4089 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory Access
 4090 Protocol (v3): UTF-8 String Representation of Distinguished Names".
 - 4091 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute
 4092 ValuePairs in that RDN in ascending order when compared as octet strings (described in
 4093 ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
 - 4094 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key
 4095 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section 4.1.2.4
 4096 "Issuer".
 - 4097 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal

4098 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-
 4099 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It
 4100 SHALL return "True" if and only if the two arguments are equal. Otherwise, it SHALL return
 4101 "False". An RFC822 name consists of a local-part followed by "@" followed by a domain-part.
 4102 The local-part is case-sensitive, while the domain-part (which is usually a DNS host name) is not
 4103 case-sensitive. Perform the following operations:

 - 4104 1. Normalize the domain-part of each argument to lower case
 - 4105 2. Compare the expressions by applying the function
 4106 "urn:oasis:names:tc:xacml:1.0:function:string-equal" to the normalized arguments.
 - 4107 • urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal

4108 This function SHALL take two arguments of data-type
4109 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an
4110 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet sequences
4111 represented by the value of both arguments have equal length and are equal in a conjunctive,
4112 point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function.
4113 Otherwise, it SHALL return "False". The conversion from the string representation to an octet
4114 sequence SHALL be as specified in [XS] Section 3.2.15.

- 4115 • urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal

4116 This function SHALL take two arguments of data-type
4117 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an
4118 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet sequences
4119 represented by the value of both arguments have equal length and are equal in a conjunctive,
4120 point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function.
4121 Otherwise, it SHALL return "False". The conversion from the string representation to an octet
4122 sequence SHALL be as specified in [XS] Section 3.2.16.

4123 A.3.2 Arithmetic functions

4124 All of the following functions SHALL take two arguments of the specified data-type, integer, or double,
4125 and SHALL return an element of integer or double data-type, respectively. However, the "add" and
4126 "multiply" functions MAY take more than two arguments. Each function evaluation operating on doubles
4127 SHALL proceed as specified by their logical counterparts in IEEE 754 [IEEE754]. For all of these
4128 functions, if any argument is "Indeterminate", then the function SHALL evaluate to "Indeterminate". In the
4129 case of the divide functions, if the divisor is zero, then the function SHALL evaluate to "Indeterminate".

- 4130 • urn:oasis:names:tc:xacml:1.0:function:integer-add
4131 This function MUST accept two or more arguments.

- 4132 • urn:oasis:names:tc:xacml:1.0:function:double-add
4133 This function MUST accept two or more arguments.

- 4134 • urn:oasis:names:tc:xacml:1.0:function:integer-subtract
- 4135 • urn:oasis:names:tc:xacml:1.0:function:double-subtract
- 4136 • urn:oasis:names:tc:xacml:1.0:function:integer-multiply
4137 This function MUST accept two or more arguments.

- 4138 • urn:oasis:names:tc:xacml:1.0:function:double-multiply
4139 This function MUST accept two or more arguments.

- 4140 • urn:oasis:names:tc:xacml:1.0:function:integer-divide
- 4141 • urn:oasis:names:tc:xacml:1.0:function:double-divide
- 4142 • urn:oasis:names:tc:xacml:1.0:function:integer-mod

4143 The following functions SHALL take a single argument of the specified data-type. The round and floor
4144 functions SHALL take a single argument of data-type "http://www.w3.org/2001/XMLSchema#double" and
4145 return a value of the data-type "http://www.w3.org/2001/XMLSchema#double".

- 4146 • urn:oasis:names:tc:xacml:1.0:function:integer-abs
- 4147 • urn:oasis:names:tc:xacml:1.0:function:double-abs
- 4148 • urn:oasis:names:tc:xacml:1.0:function:round
- 4149 • urn:oasis:names:tc:xacml:1.0:function:floor

4150 A.3.3 String conversion functions

4151 The following functions convert between values of the data-type
4152 "http://www.w3.org/2001/XMLSchema#string" primitive types.

- 4153 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-space
 - 4154 This function SHALL take one argument of data-type
 - 4155 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by stripping off all
 - 4156 leading and trailing white space characters. The whitespace characters are defined in the
 - 4157 metasympol S (Production 3) of **[XML]**.
- 4158 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case
 - 4159 This function SHALL take one argument of data-type
 - 4160 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by converting each
 - 4161 upper case character to its lower case equivalent. Case mapping shall be done as specified for
 - 4162 the fn:lower-case function in **[XF]** with no tailoring for particular languages or environments.

4163 **A.3.4 Numeric data-type conversion functions**

4164 The following functions convert between the data-type "http://www.w3.org/2001/XMLSchema#integer"
4165 and "http://www.w3.org/2001/XMLSchema#double" primitive types.

- 4166 • urn:oasis:names:tc:xacml:1.0:function:double-to-integer
 - 4167 This function SHALL take one argument of data-type
 - 4168 "http://www.w3.org/2001/XMLSchema#double" and SHALL truncate its numeric value to a whole
 - 4169 number and return an element of data-type "http://www.w3.org/2001/XMLSchema#integer".
- 4170 • urn:oasis:names:tc:xacml:1.0:function:integer-to-double
 - 4171 This function SHALL take one argument of data-type
 - 4172 "http://www.w3.org/2001/XMLSchema#integer" and SHALL promote its value to an element of
 - 4173 data-type "http://www.w3.org/2001/XMLSchema#double" with the same numeric value. If the
 - 4174 integer argument is outside the range which can be represented by a double, the result SHALL
 - 4175 be Indeterminate, with the status code "urn:oasis:names:tc:xacml:1.0:status:processing-error".

4176 **A.3.5 Logical functions**

4177 This section contains the specification for logical functions that operate on arguments of data-type
4178 "http://www.w3.org/2001/XMLSchema#boolean".

- 4179 • urn:oasis:names:tc:xacml:1.0:function:or
 - 4180 This function SHALL return "False" if it has no arguments and SHALL return "True" if at least one
 - 4181 of its arguments evaluates to "True". The order of evaluation SHALL be from first argument to
 - 4182 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",
 - 4183 leaving the rest of the arguments unevaluated.
- 4184 • urn:oasis:names:tc:xacml:1.0:function:and
 - 4185 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of its
 - 4186 arguments evaluates to "False". The order of evaluation SHALL be from first argument to last.
 - 4187 The evaluation SHALL stop with a result of "False" if any argument evaluates to "False", leaving
 - 4188 the rest of the arguments unevaluated.
- 4189 • urn:oasis:names:tc:xacml:1.0:function:n-of
 - 4190 The first argument to this function SHALL be of data-type
 - 4191 http://www.w3.org/2001/XMLSchema#integer. The remaining arguments SHALL be of data-type
 - 4192 http://www.w3.org/2001/XMLSchema#boolean. The first argument specifies the minimum
 - 4193 number of the remaining arguments that MUST evaluate to "True" for the expression to be
 - 4194 considered "True". If the first argument is 0, the result SHALL be "True". If the number of
 - 4195 arguments after the first one is less than the value of the first argument, then the expression
 - 4196 SHALL result in "Indeterminate". The order of evaluation SHALL be: first evaluate the integer
 - 4197 value, and then evaluate each subsequent argument. The evaluation SHALL stop and return
 - 4198 "True" if the specified number of arguments evaluate to "True". The evaluation of arguments
 - 4199 SHALL stop if it is determined that evaluating the remaining arguments will not satisfy the
 - 4200 requirement.

- 4201 • urn:oasis:names:tc:xacml:1.0:function:not
4202 This function SHALL take one argument of data-type
4203 "http://www.w3.org/2001/XMLSchema#boolean". If the argument evaluates to "True", then the
4204 result of the expression SHALL be "False". If the argument evaluates to "False", then the result
4205 of the expression SHALL be "True".

4206 Note: When evaluating and, or, or n-of, it MAY NOT be necessary to attempt a full evaluation of each
4207 argument in order to determine whether the evaluation of the argument would result in "Indeterminate".
4208 Analysis of the argument regarding the availability of its **attributes**, or other analysis regarding errors,
4209 such as "divide-by-zero", may render the argument error free. Such arguments occurring in the
4210 expression in a position after the evaluation is stated to stop need not be processed.

4211 A.3.6 Numeric comparison functions

4212 These functions form a minimal set for comparing two numbers, yielding a Boolean result. For doubles
4213 they SHALL comply with the rules governed by IEEE 754 [IEEE754].

- 4214 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
- 4215 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
- 4216 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than
- 4217 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
- 4218 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than
- 4219 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal
- 4220 • urn:oasis:names:tc:xacml:1.0:function:double-less-than
- 4221 • urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal

4222 A.3.7 Date and time arithmetic functions

4223 These functions perform arithmetic operations with date and time.

- 4224 • urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration
4225 This function SHALL take two arguments, the first SHALL be of data-type
4226 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be of data-type
4227 "http://www.w3.org/2001/XMLSchema#dayTimeDuration". It SHALL return a result of
4228 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by
4229 adding the second argument to the first argument according to the specification of adding
4230 durations to date and time [XS] Appendix E.
- 4231 • urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration
4232 This function SHALL take two arguments, the first SHALL be a
4233 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
4234 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
4235 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by
4236 adding the second argument to the first argument according to the specification of adding
4237 durations to date and time [XS] Appendix E.
- 4238 • urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration
4239 This function SHALL take two arguments, the first SHALL be a
4240 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
4241 "http://www.w3.org/2001/XMLSchema#dayTimeDuration". It SHALL return a result of
4242 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive duration,
4243 then this function SHALL return the value by adding the corresponding negative duration, as per
4244 the specification [XS] Appendix E. If the second argument is a negative duration, then the result
4245 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-
4246 dayTimeDuration" had been applied to the corresponding positive duration.

- 4247 • urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration
 - 4248 This function SHALL take two arguments, the first SHALL be a
 - 4249 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
 - 4250 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
 - 4251 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive duration,
 - 4252 then this function SHALL return the value by adding the corresponding negative duration, as per
 - 4253 the specification **[XS]** Appendix E. If the second argument is a negative duration, then the result
 - 4254 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-
 - 4255 yearMonthDuration" had been applied to the corresponding positive duration.
- 4256 • urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration
 - 4257 This function SHALL take two arguments, the first SHALL be a
 - 4258 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
 - 4259 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
 - 4260 "http://www.w3.org/2001/XMLSchema#date". This function SHALL return the value by adding the
 - 4261 second argument to the first argument according to the specification of adding duration to date
 - 4262 **[XS]** Appendix E.
- 4263 • urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration
 - 4264 This function SHALL take two arguments, the first SHALL be a
 - 4265 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
 - 4266 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
 - 4267 "http://www.w3.org/2001/XMLSchema#date". If the second argument is a positive duration, then
 - 4268 this function SHALL return the value by adding the corresponding negative duration, as per the
 - 4269 specification **[XS]** Appendix E. If the second argument is a negative duration, then the result
 - 4270 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration"
 - 4271 had been applied to the corresponding positive duration.

4272 **A.3.8 Non-numeric comparison functions**

4273 These functions perform comparison operations on two arguments of non-numerical types.

- 4274 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than
 - 4275 This function SHALL take two arguments of data-type
 - 4276 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
 - 4277 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 - 4278 argument is lexicographically strictly greater than the second argument. Otherwise, it SHALL
 - 4279 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
 - 4280 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.
- 4281 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal
 - 4282 This function SHALL take two arguments of data-type
 - 4283 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
 - 4284 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 - 4285 argument is lexicographically greater than or equal to the second argument. Otherwise, it SHALL
 - 4286 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
 - 4287 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.
- 4288 • urn:oasis:names:tc:xacml:1.0:function:string-less-than
 - 4289 This function SHALL take two arguments of data-type
 - 4290 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
 - 4291 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only the first
 - 4292 argument is lexicographically strictly less than the second argument. Otherwise, it SHALL return
 - 4293 "False". The comparison SHALL use Unicode codepoint collation, as defined for the identifier
 - 4294 http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.
- 4295 • urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal

- 4296 This function SHALL take two arguments of data-type
 4297 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
 4298 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only the first
 4299 argument is lexicographically less than or equal to the second argument. Otherwise, it SHALL
 4300 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
 4301 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.
- 4302 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than

4303 This function SHALL take two arguments of data-type
 4304 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4305 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4306 argument is greater than the second argument according to the order relation specified for
 4307 "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it SHALL return
 4308 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does
 4309 not. In such cases, the time-in-range function should be used.
 - 4310 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal

4311 This function SHALL take two arguments of data-type
 4312 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4313 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4314 argument is greater than or equal to the second argument according to the order relation
 4315 specified for "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it
 4316 SHALL return "False". Note: it is illegal to compare a time that includes a time-zone value with
 4317 one that does not. In such cases, the time-in-range function should be used.
 - 4318 • urn:oasis:names:tc:xacml:1.0:function:time-less-than

4319 This function SHALL take two arguments of data-type
 4320 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4321 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4322 argument is less than the second argument according to the order relation specified for
 4323 "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it SHALL return
 4324 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does
 4325 not. In such cases, the time-in-range function should be used.
 - 4326 • urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal

4327 This function SHALL take two arguments of data-type
 4328 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4329 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4330 argument is less than or equal to the second argument according to the order relation specified
 4331 for "http://www.w3.org/2001/XMLSchema#time" **[XS]** Section 3.2.8. Otherwise, it SHALL return
 4332 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does
 4333 not. In such cases, the time-in-range function should be used.
 - 4334 • urn:oasis:names:tc:xacml:2.0:function:time-in-range

4335 This function SHALL take three arguments of data-type
 4336 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an
 4337 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument falls
 4338 in the range defined inclusively by the second and third arguments. Otherwise, it SHALL return
 4339 "False". Regardless of its value, the third argument SHALL be interpreted as a time that is equal
 4340 to, or later than by less than twenty-four hours, the second argument. If no time zone is provided
 4341 for the first argument, it SHALL use the default time zone at the **context handler**. If no time zone
 4342 is provided for the second or third arguments, then they SHALL use the time zone from the first
 4343 argument.
 - 4344 • urn:oasis:names:tc:xacml:1.0:function:date-time-greater-than

4345 This function SHALL take two arguments of data-type
 4346 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an

- 4347 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4348 argument is greater than the second argument according to the order relation specified for
 4349 "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7. Otherwise, it
 4350 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an
 4351 implicit time-zone value SHALL be assigned, as described in [XS].
- 4352 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal

4353 This function SHALL take two arguments of data-type
 4354 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
 4355 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4356 argument is greater than or equal to the second argument according to the order relation
 4357 specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7.
 4358 Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone
 4359 value, then an implicit time-zone value SHALL be assigned, as described in [XS].
 - 4360 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than

4361 This function SHALL take two arguments of data-type
 4362 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an
 4363 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4364 argument is less than the second argument according to the order relation specified for
 4365 "http://www.w3.org/2001/XMLSchema#dateTime" by [XS, part 2, section 3.2.7]. Otherwise, it
 4366 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an
 4367 implicit time-zone value SHALL be assigned, as described in [XS].
 - 4368 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal

4369 This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#
 4370 dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL
 4371 return "True" if and only if the first argument is less than or equal to the second argument
 4372 according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" by
 4373 [XS] part 2, section 3.2.7. Otherwise, it SHALL return "False". Note: if a dateTime value does
 4374 not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described
 4375 in [XS].
 - 4376 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than

4377 This function SHALL take two arguments of data-type
 4378 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
 4379 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4380 argument is greater than the second argument according to the order relation specified for
 4381 "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it SHALL
 4382 return "False". Note: if a date value does not include a time-zone value, then an implicit time-
 4383 zone value SHALL be assigned, as described in [XS].
 - 4384 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal

4385 This function SHALL take two arguments of data-type
 4386 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
 4387 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4388 argument is greater than or equal to the second argument according to the order relation
 4389 specified for "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9.
 4390 Otherwise, it SHALL return "False". Note: if a date value does not include a time-zone value,
 4391 then an implicit time-zone value SHALL be assigned, as described in [XS].
 - 4392 • urn:oasis:names:tc:xacml:1.0:function:date-less-than

4393 This function SHALL take two arguments of data-type
 4394 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
 4395 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
 4396 argument is less than the second argument according to the order relation specified for
 4397 "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it SHALL

4398 return "False". Note: if a date value does not include a time-zone value, then an implicit time-
4399 zone value SHALL be assigned, as described in [XS].

- 4400 • urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal
4401 This function SHALL take two arguments of data-type
4402 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an
4403 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
4404 argument is less than or equal to the second argument according to the order relation specified
4405 for "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it
4406 SHALL return "False". Note: if a date value does not include a time-zone value, then an implicit
4407 time-zone value SHALL be assigned, as described in [XS].

4408 A.3.9 String functions

4409 The following functions operate on strings and convert to and from other data types.

- 4410 • urn:oasis:names:tc:xacml:2.0:function:string-concatenate
4411 This function SHALL take two or more arguments of data-type
4412 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
4413 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the concatenation, in order,
4414 of the arguments.
- 4415 • urn:oasis:names:tc:xacml:3.0:function:boolean-from-string
4416 This function SHALL take one argument of data-type
4417 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4418 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be the string converted to a
4419 boolean.
- 4420 • urn:oasis:names:tc:xacml:3.0:function:string-from-boolean
4421 This function SHALL take one argument of data-type
4422 "http://www.w3.org/2001/XMLSchema#boolean", and SHALL return an
4423 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the boolean converted to a
4424 string.
- 4425 • urn:oasis:names:tc:xacml:3.0:function:integer-from-string
4426 This function SHALL take one argument of data-type
4427 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4428 "http://www.w3.org/2001/XMLSchema#integer". The result SHALL be the string converted to an
4429 integer.
- 4430 • urn:oasis:names:tc:xacml:3.0:function:string-from-integer
4431 This function SHALL take one argument of data-type
4432 "http://www.w3.org/2001/XMLSchema#integer", and SHALL return an
4433 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the integer converted to a
4434 string.
- 4435 • urn:oasis:names:tc:xacml:3.0:function:double-from-string
4436 This function SHALL take one argument of data-type
4437 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4438 "http://www.w3.org/2001/XMLSchema#double". The result SHALL be the string converted to a
4439 double.
- 4440 • urn:oasis:names:tc:xacml:3.0:function:string-from-double
4441 This function SHALL take one argument of data-type
4442 "http://www.w3.org/2001/XMLSchema#double", and SHALL return an
4443 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the double converted to a
4444 string.

- 4445 • urn:oasis:names:tc:xacml:3.0:function:time-from-string
4446 This function SHALL take one argument of data-type
4447 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4448 "http://www.w3.org/2001/XMLSchema#time". The result SHALL be the string converted to a time.
- 4449 • urn:oasis:names:tc:xacml:3.0:function:string-from-time
4450 This function SHALL take one argument of data-type
4451 "http://www.w3.org/2001/XMLSchema#time", and SHALL return an
4452 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the time converted to a
4453 string.
- 4454 • urn:oasis:names:tc:xacml:3.0:function:date-from-string
4455 This function SHALL take one argument of data-type
4456 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4457 "http://www.w3.org/2001/XMLSchema#date". The result SHALL be the string converted to a
4458 date.
- 4459 • urn:oasis:names:tc:xacml:3.0:function:string-from-date
4460 This function SHALL take one argument of data-type
4461 "http://www.w3.org/2001/XMLSchema#date", and SHALL return an
4462 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the date converted to a
4463 string.
- 4464 • urn:oasis:names:tc:xacml:3.0:function:dateTime-from-string
4465 This function SHALL take one argument of data-type
4466 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4467 "http://www.w3.org/2001/XMLSchema#dateTime". The result SHALL be the string converted to a
4468 dateTime.
- 4469 urn:oasis:names:tc:xacml:3.0:function:string-from-dateTime
4470 This function SHALL take one argument of data-type
4471 "http://www.w3.org/2001/XMLSchema#dateTime", and SHALL return an
4472 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the dateTime converted to a
4473 string.
- 4474 • urn:oasis:names:tc:xacml:3.0:function:anyURI-from-string
4475 This function SHALL take one argument of data-type
4476 "http://www.w3.org/2001/XMLSchema#string", and SHALL return a
4477 "http://www.w3.org/2001/XMLSchema#anyURI". The result SHALL be the URI constructed by
4478 converting the argument to an URI.
- 4479 • urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI
4480 This function SHALL take one argument of data-type
4481 "http://www.w3.org/2001/XMLSchema#anyURI", and SHALL return an
4482 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the URI converted to a
4483 string.
- 4484 • urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-from-string
4485 This function SHALL take one argument of data-type
4486 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4487 "http://www.w3.org/2001/XMLSchema#dayTimeDuration ". The result SHALL be the string
4488 converted to a dayTimeDuration.
- 4489 • urn:oasis:names:tc:xacml:3.0:function:string-from-dayTimeDuration
4490 This function SHALL take one argument of data-type
4491 "http://www.w3.org/2001/XMLSchema#dayTimeDuration ", and SHALL return an

- 4492 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the dayTimeDuration
4493 converted to a string.
- 4494 • urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-from-string
4495 This function SHALL take one argument of data-type
4496 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4497 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". The result SHALL be the string
4498 converted to a yearMonthDuration.
 - 4499 • urn:oasis:names:tc:xacml:3.0:function:string-from-yearMonthDuration
4500 This function SHALL take one argument of data-type
4501 "http://www.w3.org/2001/XMLSchema#yearMonthDuration", and SHALL return an
4502 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the yearMonthDuration
4503 converted to a string.
 - 4504 • urn:oasis:names:tc:xacml:3.0:function:x500Name-from-string
4505 This function SHALL take one argument of data-type
4506 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4507 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". The result SHALL be the string converted
4508 to an x500Name.
 - 4509 • urn:oasis:names:tc:xacml:3.0:function:string-from-x500Name
4510 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:1.0:data-
4511 type:x500Name", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result
4512 SHALL be the x500Name converted to a string.
 - 4513 • urn:oasis:names:tc:xacml:3.0:function:rfc822Name-from-string
4514 This function SHALL take one argument of data-type
4515 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4516 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". The result SHALL be the string converted
4517 to an rfc822Name.
 - 4518 • urn:oasis:names:tc:xacml:3.0:function:string-from-rfc822Name
4519 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:1.0:data-
4520 type:rfc822Name", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The
4521 result SHALL be the rfc822Name converted to a string.
 - 4522 • urn:oasis:names:tc:xacml:3.0:function:ipAddress-from-string
4523 This function SHALL take one argument of data-type
4524 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4525 "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". The result SHALL be the string converted to
4526 an ipAddress.
 - 4527 • urn:oasis:names:tc:xacml:3.0:function:string-from-ipAddress
4528 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-
4529 type:ipAddress", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result
4530 SHALL be the ipAddress converted to a string.
 - 4531 • urn:oasis:names:tc:xacml:3.0:function:dnsName-from-string
4532 This function SHALL take one argument of data-type
4533 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
4534 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName". The result SHALL be the string converted to
4535 a dnsName.
 - 4536 • urn:oasis:names:tc:xacml:3.0:function:string-from-dnsName
4537 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-
4538 type:dnsName", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result
4539 SHALL be the dnsName converted to a string.

- 4540 • urn:oasis:names:tc:xacml:3.0:function:string-starts-with
 - 4541 This function SHALL take two arguments of data-type
 - 4542 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
 - 4543 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
 - 4544 begins with the first string, and false otherwise. Equality testing SHALL be done as defined for
 - 4545 urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4546 • urn:oasis:names:tc:xacml:3.0:function:anyURI-starts-with
 - 4547 This function SHALL take a first argument of data-
 - 4548 type "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
 - 4549 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
 - 4550 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
 - 4551 to a string begins with the string, and false otherwise. Equality testing SHALL be done as defined
 - 4552 for urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4553 • urn:oasis:names:tc:xacml:3.0:function:string-ends-with
 - 4554 This function SHALL take two arguments of data-type
 - 4555 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
 - 4556 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
 - 4557 ends with the first string, and false otherwise. Equality testing SHALL be done as defined for
 - 4558 urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4559 • urn:oasis:names:tc:xacml:3.0:function:anyURI-ends-with
 - 4560 This function SHALL take a first argument of data-type
 - 4561 "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
 - 4562 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
 - 4563 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
 - 4564 to a string ends with the string, and false otherwise. Equality testing SHALL be done as defined
 - 4565 for urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4566 • urn:oasis:names:tc:xacml:3.0:function:string-contains
 - 4567 This function SHALL take two arguments of data-type
 - 4568 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
 - 4569 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
 - 4570 contains the first string, and false otherwise. Equality testing SHALL be done as defined for
 - 4571 urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4572 • urn:oasis:names:tc:xacml:3.0:function:anyURI-contains
 - 4573 This function SHALL take a first argument of data-type
 - 4574 "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
 - 4575 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
 - 4576 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
 - 4577 to a string contains the string, and false otherwise. Equality testing SHALL be done as defined for
 - 4578 urn:oasis:names:tc:xacml:1.0:function:string-equal.
- 4579 • urn:oasis:names:tc:xacml:3.0:function:string-substring
 - 4580 This function SHALL take a first argument of data-type
 - 4581 "http://www.w3.org/2001/XMLSchema#string" and a second and a third argument of type
 - 4582 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return a
 - 4583 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first
 - 4584 argument beginning at the position given by the second argument and ending at the position
 - 4585 before the position given by the third argument. The first character of the string has position zero.
 - 4586 The negative integer value -1 given for the third arguments indicates the end of the string. If the
 - 4587 second or third arguments are out of bounds, then the function MUST evaluate to Indeterminate
 - 4588 with a status code of urn:oasis:names:tc:xacml:1.0:status:processing-error.
- 4589 • urn:oasis:names:tc:xacml:3.0:function:anyURI-substring

4590 This function SHALL take a first argument of data-type
4591 "http://www.w3.org/2001/XMLSchema#anyURI" and a second and a third argument of type
4592 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return a
4593 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first
4594 argument converted to a string beginning at the position given by the second argument and
4595 ending at the position before the position given by the third argument. The first character of the
4596 URI converted to a string has position zero. The negative integer value -1 given for the third
4597 arguments indicates the end of the string. If the second or third arguments are out of bounds,
4598 then the function MUST evaluate to Indeterminate with a status code of
4599 urn:oasis:names:tc:xacml:1.0:status:processing-error. If the resulting substring
4600 is not syntactically a valid URI, then the function MUST evaluate to Indeterminate with a status
4601 code of urn:oasis:names:tc:xacml:1.0:status:processing-error.
4602

4603 A.3.10 Bag functions

4604 These functions operate on a **bag** of 'type' values, where type is one of the primitive data-types, and x.x
4605 is a version of XACML where the function has been defined. Some additional conditions defined for
4606 each function below SHALL cause the expression to evaluate to "Indeterminate".

- 4607 • urn:oasis:names:tc:xacml:x.x:function:type-one-and-only

4608 This function SHALL take a **bag** of 'type' values as an argument and SHALL return a value of '-
4609 type'. It SHALL return the only value in the **bag**. If the **bag** does not have one and only one
4610 value, then the expression SHALL evaluate to "Indeterminate".

- 4611 • urn:oasis:names:tc:xacml:x.x:function:type-bag-size

4612 This function SHALL take a **bag** of 'type' values as an argument and SHALL return an
4613 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

- 4614 • urn:oasis:names:tc:xacml:x.x:function:type-is-in

4615 This function SHALL take an argument of 'type' as the first argument and a **bag** of type values as
4616 the second argument and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The
4617 function SHALL evaluate to "True" if and only if the first argument matches by the
4618 "urn:oasis:names:tc:xacml:x.x:function:type-equal" any value in the **bag**. Otherwise, it SHALL
4619 return "False".

- 4620 • urn:oasis:names:tc:xacml:x.x:function:type-bag

4621 This function SHALL take any number of arguments of 'type' and return a **bag** of 'type' values
4622 containing the values of the arguments. An application of this function to zero arguments SHALL
4623 produce an empty **bag** of the specified data-type.

4624 A.3.11 Set functions

4625 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**.

- 4626 • urn:oasis:names:tc:xacml:x.x:function:type-intersection

4627 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
4628 **bag** of 'type' values such that it contains only elements that are common between the two **bags**,
4629 which is determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal". No duplicates, as
4630 determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal", SHALL exist in the result.

- 4631 • urn:oasis:names:tc:xacml:x.x:function:type-at-least-one-member-of

4632 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
4633 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if and
4634 only if at least one element of the first argument is contained in the second argument as
4635 determined by "urn:oasis:names:tc:xacml:x.x:function:type-is-in".

- 4636 • urn:oasis:names:tc:xacml:x.x:function:type-union

4637 This function SHALL take two or more arguments that are both a **bag** of 'type' values. The
4638 expression SHALL return a **bag** of 'type' such that it contains all elements of all the argument
4639 **bags**. No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
4640 SHALL exist in the result.

4641 • urn:oasis:names:tc:xacml:x.x:function:type-subset

4642 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
4643 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
4644 argument is a subset of the second argument. Each argument SHALL be considered to have had
4645 its duplicates removed, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",
4646 before the subset calculation.

4647 • urn:oasis:names:tc:xacml:x.x:function:type-set-equals

4648 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a
4649 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return the result of applying
4650 "urn:oasis:names:tc:xacml:1.0:function:and" to the application of
4651 "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the first and second arguments and the
4652 application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the second and first
4653 arguments.

4654 A.3.12 Higher-order bag functions

4655 This section describes functions in XACML that perform operations on **bags** such that functions may be
4656 applied to the **bags** in general.

4657 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally specify
4658 the semantics of these functions. Although the English description is adequate, a formal specification of
4659 the semantics is helpful.

4660 For a quick summary, in the following Haskell notation, a function definition takes the form of clauses that
4661 are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty list, whereas the
4662 expression "(x:xs)" matches against an argument of a non-empty list of which "x" represents the first
4663 element of the list, and "xs" is the rest of the list, which may be an empty list. We use the Haskell notion
4664 of a list, which is an ordered collection of elements, to model the XACML **bags** of values.

4665 A simple Haskell definition of a familiar function "urn:oasis:names:tc:xacml:1.0:function:and" that takes a
4666 list of values of type Boolean is defined as follows:

```
4667 and:: [Bool] -> Bool
4668 and []      = True
4669 and (x:xs)  = x && (and xs)
```

4670 The first definition line denoted by a ":" formally describes the data-type of the function, which takes a list
4671 of Booleans, denoted by "[Bool]", and returns a Boolean, denoted by "Bool". The second definition line is
4672 a clause that states that the function "and" applied to the empty list is "True". The third definition line is a
4673 clause that states that for a non-empty list, such that the first element is "x", which is a value of data-type
4674 Bool, the function "and" applied to x SHALL be combined with, using the logical conjunction function,
4675 which is denoted by the infix symbol "&&", the result of recursively applying the function "and" to the rest
4676 of the list. Of course, an application of the "and" function is "True" if and only if the list to which it is
4677 applied is empty or every element of the list is "True". For example, the evaluation of the following
4678 Haskell expressions,

4679 (and []), (and [True]), (and [True,True]), (and [True,True,False])

4680 evaluate to "True", "True", "True", and "False", respectively.

4681 • urn:oasis:names:tc:xacml:1.0:function:any-of

4682 This function applies a Boolean function between specific primitive values and a **bag** of values,
4683 and SHALL return "True" if and only if the **predicate** is "True" for at least one element of the **bag**.

4684 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL
4685 be an <Function> element that names a Boolean function that takes n arguments of primitive
4686 types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-
4687 types and one SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated as
4688 if the function named in the <Function> argument were applied to the n-1 non-bag arguments
4689 and each element of the bag argument and the results are combined with
4690 "urn:oasis:names:tc:xacml:1.0:function:or".

4691 For example, the following expression SHALL return "True":

```
4692 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
4693   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4694   <AttributeValue  
4695     DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4696     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4697       <AttributeValue  
4698         DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4699         <AttributeValue  
4700           DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4701           <AttributeValue  
4702             DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4703             <AttributeValue  
4704               DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4705             </Apply>  
4706           </Apply>
```

4707 This expression is "True" because the first argument is equal to at least one of the elements of
4708 the **bag**, according to the function.

4709 • urn:oasis:names:tc:xacml:1.0:function:all-of

4710 This function applies a Boolean function between a specific primitive values and a **bag** of values,
4711 and returns "True" if and only if the **predicate** is "True" for every element of the **bag**.

4712 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL
4713 be a <Function> element that names a Boolean function that takes n arguments of primitive
4714 types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-
4715 types and one SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated as
4716 if the function named in the <Function> argument were applied to the n-1 non-bag arguments
4717 and each element of the bag argument and the results are combined with
4718 "urn:oasis:names:tc:xacml:1.0:function:and".

4719 For example, the following expression SHALL evaluate to "True":

```
4720 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">  
4721   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-  
4722     greater-than"/>  
4723   <AttributeValue  
4724     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>  
4725     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4726       <AttributeValue  
4727         DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>  
4728         <AttributeValue  
4729           DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>  
4730           <AttributeValue  
4731             DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>  
4732             <AttributeValue  
4733               DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>  
4734             </Apply>  
4735           </Apply>
```

4736 This expression is "True" because the first argument (10) is greater than all of the elements of the
4737 **bag** (9,3,4 and 2).

4738 • urn:oasis:names:tc:xacml:1.0:function:any-of-any

4739 This function applies a Boolean function on each tuple from the cross product on all bags
4740 arguments, and returns "True" if and only if the **predicate** is "True" for at least one inside-function
4741 call.

4742 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL
4743 be an <Function> element that names a Boolean function that takes n arguments. The
4744 remaining arguments are either primitive data types or bags of primitive types. The expression
4745 SHALL be evaluated as if the function named in the <Function> argument was applied between
4746 every tuple of the cross product on all bags and the primitive values, and the results were
4747 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the result of
4748 the expression SHALL be "True" if and only if the applied **predicate** is "True" for at least one
4749 function call on the tuples from the **bags** and primitive values.

4750 For example, the following expression SHALL evaluate to "True":

```
4751 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">  
4752   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4753   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4754     <AttributeValue  
4755       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4756     <AttributeValue  
4757       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>  
4758   </Apply>  
4759   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4760     <AttributeValue  
4761       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4762     <AttributeValue  
4763       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4764     <AttributeValue  
4765       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4766     <AttributeValue  
4767       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4768   </Apply>  
4769 </Apply>
```

4770 This expression is "True" because at least one of the elements of the first **bag**, namely "Ringo", is
4771 equal to at least one of the elements of the second **bag**.

4772 • urn:oasis:names:tc:xacml:1.0:function:all-of-any

4773 This function applies a Boolean function between the elements of two **bags**. The expression
4774 SHALL be "True" if and only if the supplied **predicate** is 'True' between each element of the first
4775 **bag** and any element of the second **bag**.

4776 This function SHALL take three arguments. The first argument SHALL be an <Function>
4777 element that names a Boolean function that takes two arguments of primitive types. The second
4778 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a
4779 primitive data-type. The expression SHALL be evaluated as if the
4780 "urn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each value of the first
4781 **bag** and the whole of the second **bag** using the supplied xacml:Function, and the results were
4782 then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4783 In Haskell, taking advantage of the "any_of" function defined in Haskell above, the semantics of
4784 the "all_of_any" function are as follows:

```
4785 all_of_any :: ( a -> b -> Bool )      -> [a]-> [b] -> Bool  
4786 all_of_any f []      ys              = True  
4787 all_of_any f (x:xs)  ys              = (any_of f x ys) && (all_of_any f xs ys)
```

4788 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first element of
4789 the list as "x" and the rest of the list as "xs".

4790 For example, the following expression SHALL evaluate to "True":

```

4791 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4792   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-
4793   greater-than"/>
4794   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4795     <AttributeValue
4796     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4797     <AttributeValue
4798     DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4799   </Apply>
4800   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4801     <AttributeValue
4802     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4803     <AttributeValue
4804     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4805     <AttributeValue
4806     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4807     <AttributeValue
4808     DataType="http://www.w3.org/2001/XMLSchema#integer">19</AttributeValue>
4809   </Apply>
4810 </Apply>

```

4811 This expression is "True" because each of the elements of the first **bag** is greater than at least
4812 one of the elements of the second **bag**.

4813 • urn:oasis:names:tc:xacml:1.0:function:any-of-all

4814 This function applies a Boolean function between the elements of two **bags**. The expression
4815 SHALL be "True" if and only if the supplied **predicate** is "True" between each element of the
4816 second **bag** and any element of the first **bag**.

4817 This function SHALL take three arguments. The first argument SHALL be an <Function>
4818 element that names a Boolean function that takes two arguments of primitive types. The second
4819 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a
4820 primitive data-type. The expression SHALL be evaluated as if the
4821 "rn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each value of the
4822 second **bag** and the whole of the first **bag** using the supplied xacml:Function, and the results
4823 were then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4824 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics of the
4825 "any_of_all" function are as follows:

```

4826 any_of_all :: ( a -> b -> Bool )      -> [a]-> [b] -> Bool
4827 any_of_all f []          ys           = False
4828 any_of_all f (x:xs)     ys           = (all_of f x ys) || ( any_of_all f xs ys)

```

4829 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first
4830 element of the list as "x" and the rest of the list as "xs".

4831 For example, the following expression SHALL evaluate to "True":

```

4832 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4833   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-
4834   greater-than"/>
4835   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4836     <AttributeValue
4837     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4838     <AttributeValue
4839     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4840   </Apply>
4841   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4842     <AttributeValue
4843     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4844     <AttributeValue
4845     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>

```



```

4846     <AttributeValue
4847     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4848     <AttributeValue
4849     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4850   </Apply>
4851 </Apply>

```

4852 This expression is "True" because, for all of the values in the second **bag**, there is a value in the
4853 first **bag** that is greater.

4854 • urn:oasis:names:tc:xacml:1.0:function:all-of-all

4855 This function applies a Boolean function between the elements of two **bags**. The expression
4856 SHALL be "True" if and only if the supplied **predicate** is "True" between each and every element
4857 of the first **bag** collectively against all the elements of the second **bag**.

4858 This function SHALL take three arguments. The first argument SHALL be an <Function>
4859 element that names a Boolean function that takes two arguments of primitive types. The second
4860 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a
4861 primitive data-type. The expression is evaluated as if the function named in the <Function>
4862 element were applied between every element of the second argument and every element of the
4863 third argument and the results were combined using "urn:oasis:names:tc:xacml:1.0:function:and".
4864 The semantics are that the result of the expression is "True" if and only if the applied **predicate** is
4865 "True" for all elements of the first **bag** compared to all the elements of the second **bag**.

4866 In Haskell, taking advantage of the "all_of" function defined in Haskell above, the semantics of the
4867 "all_of_all" function is as follows:

```

4868     all_of_all :: ( a -> b -> Bool )    -> [a] -> [b] -> Bool
4869     all_of_all f []          ys         = True
4870     all_of_all f (x:xs)     ys         = (all_of f x ys) && (all_of_all f xs ys)

```

4871 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first element of
4872 the list as "x" and the rest of the list as "xs".

4873 For example, the following expression SHALL evaluate to "True":

```

4874 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4875   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-
4876   greater-than"/>
4877   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4878     <AttributeValue
4879     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4880     <AttributeValue
4881     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4882   </Apply>
4883   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4884     <AttributeValue
4885     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4886     <AttributeValue
4887     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4888     <AttributeValue
4889     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4890     <AttributeValue
4891     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4892   </Apply>
4893 </Apply>

```

4894 This expression is "True" because all elements of the first **bag**, "5" and "6", are each greater than
4895 all of the integer values "1", "2", "3", "4" of the second **bag**.

4896 • urn:oasis:names:tc:xacml:1.0:function:map

4897 This function converts a **bag** of values to another **bag** of values.

4898 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL
4899 be a <Function> element naming a function that takes a n arguments of a primitive data-type
4900 and returns a value of a primitive data-type Under the remaining n arguments, n-1 parameters
4901 SHALL be values of primitive data-types and one SHALL be a **bag** of a primitive data-type. The
4902 expression SHALL be evaluated as if the function named in the <Function> argument were
4903 applied to the n-1 non-bag arguments and each element of the bag argument and resulting in a
4904 **bag** of the converted value. The result SHALL be a **bag** of the primitive data-type that is returned
4905 by the function named in the <xacml:Function> element.

4906 For example, the following expression,

```
4907 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">  
4908   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-  
4909   normalize-to-lower-case">  
4910     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4911       <AttributeValue  
4912       DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>  
4913       <AttributeValue  
4914       DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>  
4915     </Apply>  
4916   </Apply>
```

4917 evaluates to a **bag** containing "hello" and "world!".

4918 A.3.13 Regular-expression-based functions

4919 These functions operate on various types using regular expressions and evaluate to
4920 "http://www.w3.org/2001/XMLSchema#boolean".

- 4921 • urn:oasis:names:tc:xacml:1.0:function:string-regexp-match

4922 This function decides a regular expression match. It SHALL take two arguments of
4923 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
4924 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4925 expression and the second argument SHALL be a general string. The function specification
4926 SHALL be that of the "xf:matches" function with the arguments reversed [XF] Section 7.6.2.

- 4927 • urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match

4928 This function decides a regular expression match. It SHALL take two arguments; the first is of
4929 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4930 "http://www.w3.org/2001/XMLSchema#anyURI". It SHALL return an
4931 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4932 expression and the second argument SHALL be a URI. The function SHALL convert the second
4933 argument to type "http://www.w3.org/2001/XMLSchema#string", then apply
4934 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

- 4935 • urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match

4936 This function decides a regular expression match. It SHALL take two arguments; the first is of
4937 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4938 "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". It SHALL return an
4939 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4940 expression and the second argument SHALL be an IPv4 or IPv6 address. The function SHALL
4941 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply
4942 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

- 4943 • urn:oasis:names:tc:xacml:2.0:function:dnsName-regexp-match

4944 This function decides a regular expression match. It SHALL take two arguments; the first is of
4945 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4946 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName". It SHALL return an
4947 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4948 expression and the second argument SHALL be a DNS name. The function SHALL convert the

- 4949 second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply
4950 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4951 • urn:oasis:names:tc:xacml:2.0:function:rfc822Name-regexp-match
- 4952 This function decides a regular expression match. It SHALL take two arguments; the first is of
4953 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4954 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". It SHALL return an
4955 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4956 expression and the second argument SHALL be an RFC 822 name. The function SHALL convert
4957 the second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply
4958 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4959 • urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match
- 4960 This function decides a regular expression match. It SHALL take two arguments; the first is of
4961 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type
4962 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". It SHALL return an
4963 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular
4964 expression and the second argument SHALL be an X.500 directory name. The function SHALL
4965 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply
4966 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

4967 **A.3.14 Special match functions**

4968 These functions operate on various types and evaluate to
4969 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching algorithm.

- 4970 • urn:oasis:names:tc:xacml:1.0:function:x500Name-match
- 4971 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name"
4972 and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It shall return "True" if and
4973 only if the first argument matches some terminal sequence of RDNs from the second argument
4974 when compared using x500Name-equal.
- 4975 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match
- 4976 This function SHALL take two arguments, the first is of data-type
4977 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type
4978 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an
4979 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if the
4980 first argument matches the second argument according to the following specification.
- 4981 An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The local-
4982 part is case-sensitive, while the domain-part (which is usually a DNS name) is not case-sensitive.
- 4983 The second argument contains a complete rfc822Name. The first argument is a complete or
4984 partial rfc822Name used to select appropriate values in the second argument as follows.
- 4985 In order to match a particular address in the second argument, the first argument must specify the
4986 complete mail address to be matched. For example, if the first argument is
4987 "Anderson@sun.com", this matches a value in the second argument of "Anderson@sun.com"
4988 and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com", "anderson@sun.com" or
4989 "Anderson@east.sun.com".
- 4990 In order to match any address at a particular domain in the second argument, the first argument
4991 must specify only a domain name (usually a DNS name). For example, if the first argument is
4992 "sun.com", this matches a value in the first argument of "Anderson@sun.com" or
4993 "Baxter@SUN.COM", but not "Anderson@east.sun.com".
- 4994 In order to match any address in a particular domain in the second argument, the first argument
4995 must specify the desired domain-part with a leading ".". For example, if the first argument is
4996 ".east.sun.com", this matches a value in the second argument of "Anderson@east.sun.com" and
4997 "anne.anderson@ISRG.EAST.SUN.COM" but not "Anderson@sun.com".

4998 **A.3.15 XPath-based functions**

4999 This section specifies functions that take XPath expressions for arguments. An XPath expression
5000 evaluates to a node-set, which is a set of XML nodes that match the expression. A node or node-set is
5001 not in the formal data-type system of XACML. All comparison or other operations on node-sets are
5002 performed in isolation of the particular function specified. The context nodes and namespace mappings
5003 of the XPath expressions are defined by the XPath data-type, see section B.3. The following functions
5004 are defined:

- 5005 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-count

5006 This function SHALL take an “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression” as an
5007 argument and evaluates to an “http://www.w3.org/2001/XMLSchema#integer”. The value
5008 returned from the function SHALL be the count of the nodes within the node-set that match the
5009 given XPath expression. If the <Content> element of the category to which the XPath
5010 expression applies to is not present in the request, this function SHALL return a value of zero.

- 5011 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal

5012 This function SHALL take two “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression”
5013 arguments and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. The function
5014 SHALL return "True" if any of the XML nodes in the node-set matched by the first argument
5015 equals any of the XML nodes in the node-set matched by the second argument. Two nodes are
5016 considered equal if they have the same identity. If the <Content> element of the category to
5017 which either XPath expression applies to is not present in the request, this function SHALL return
5018 a value of “False”.

- 5019 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

5020 This function SHALL take two “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression”
5021 arguments and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function
5022 SHALL evaluate to "True" if one of the following two conditions is satisfied: (1) Any of the XML
5023 nodes in the node-set matched by the first argument is equal to any of the XML nodes in the
5024 node-set matched by the second argument; (2) any node below any of the XML nodes in the
5025 node-set matched by the first argument is equal to any of the XML nodes in the node-set
5026 matched by the second argument. Two nodes are considered equal if they have the same
5027 identity. If the <Content> element of the category to which either XPath expression applies to is
5028 not present in the request, this function SHALL return a value of “False”.

5029 NOTE: The first **condition** is equivalent to "xpath-node-equal", and guarantees that "xpath-node-equal" is
5030 a special case of "xpath-node-match".

5031 **A.3.16 Other functions**

- 5032 • urn:oasis:names:tc:xacml:3.0:function:access-permitted

5033 This function SHALL take an “http://www.w3.org/2001/XMLSchema#anyURI” and an
5034 “http://www.w3.org/2001/XMLSchema#string” as arguments. The first argument SHALL be
5035 interpreted as an **attribute** category. The second argument SHALL be interpreted as the XML
5036 content of an <Attributes> element with *Category* equal to the first argument. The function
5037 evaluates to an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return
5038 "True" if and only if the **policy** evaluation described below returns the value of "Permit".

5039 The following evaluation is described as if the **context** is actually instantiated, but it is only
5040 required that an equivalent result be obtained.

5041 The function SHALL construct a new **context**, by copying all the information from the current
5042 **context**, omitting any <Attributes> element with *Category* equal to the first argument. The
5043 second function argument SHALL be added to the **context** as the content of an <Attributes>
5044 element with *Category* equal to the first argument.

5045 The function SHALL invoke a complete **policy** evaluation using the newly constructed **context**.
5046 This evaluation SHALL be completely isolated from the evaluation which invoked the function, but
5047 shall use all current **policies** and combining algorithms, including any per request **policies**.

5048 The **PDP** SHALL detect any loop which may occur if successive evaluations invoke this function
5049 by counting the number of total invocations of any instance of this function during any single initial
5050 invocation of the **PDP**. If the total number of invocations exceeds the bound for such invocations,
5051 the initial invocation of this function evaluates to Indeterminate with a
5052 “urn:oasis:names:tc:xacml:1.0:status:processing-error” status code. Also, see the security
5053 considerations in section 9.1.8.

5054 **A.3.17 Extension functions and primitive types**

5055 Functions and primitive types are specified by string identifiers allowing for the introduction of functions in
5056 addition to those specified by XACML. This approach allows one to extend the XACML module with
5057 special functions and special primitive data-types.

5058 In order to preserve the integrity of the XACML evaluation strategy, the result of an extension function
5059 SHALL depend only on the values of its arguments. Global and hidden parameters SHALL NOT affect
5060 the evaluation of an expression. Functions SHALL NOT have side effects, as evaluation order cannot be
5061 guaranteed in a standard way.

5062 **A.4 Functions, data types and algorithms planned for deprecation**

5063 The following functions, data types and algorithms have been defined by previous versions of XACML
5064 and newer and better alternatives are defined in XACML 3.0. Their use is discouraged for new use and
5065 they are candidates for deprecation in future versions of XACML.

5066 The following xpath based functions have been replaced with equivalent functions which use the new
5067 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression datatype instead of strings.

- 5068 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-count
 - 5069 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-count
- 5070 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal
 - 5071 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal
- 5072 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-match
 - 5073 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

5074 The following URI and string concatenation function has been replaced with a string to URI conversion
5075 function, which allows the use of the general string functions with URI through string conversion.

- 5076 • urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate
 - 5077 • Replaced by urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI

5078 The following identifiers have been replaced with official identifiers defined by W3C.

- 5079 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
 - 5080 • Replaced with <http://www.w3.org/2001/XMLSchema#dayTimeDuration>
- 5081 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
 - 5082 • Replaced with <http://www.w3.org/2001/XMLSchema#yearMonthDuration>

5083 The following functions have been replaced with functions which use the updated dayTimeDuration and
5084 yearMonthDuration data types.

- 5085 • urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal
 - 5086 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal
- 5087 • urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal
 - 5088 • Replaced with urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal

- 5089 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration
- 5090 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration
- 5091 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration
- 5092 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration
- 5093 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration
- 5094 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration
- 5095 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration
- 5096 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration
- 5097 • urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration
- 5098 • Replaced with urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration
- 5099 • urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration
- 5100 • Replaced with urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration
- 5101 The following combining algorithms have been replaced with new variants which allow for better handling
- 5102 of “Indeterminate” results.
- 5103 • urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides
- 5104 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides
- 5105 • urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides
- 5106 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides
- 5107 • urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides
- 5108 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides
- 5109 • urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
- 5110 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides
- 5111 • urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides
- 5112 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides
- 5113 • urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides
- 5114 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-overrides
- 5115 • urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides
- 5116 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-overrides
- 5117 • urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides
- 5118 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-overrides

5119 B. XACML identifiers (normative)

5120 This section defines standard identifiers for commonly used entities.

5121 B.1 XACML namespaces

5122 XACML is defined using this identifier.

5123 `urn:oasis:names:tc:xacml:3.0:core:schema`

5124 B.2 Attribute categories

5125 The following **attribute** category identifiers MUST be used when an XACML 2.0 or earlier **policy** or
5126 request is translated into XACML 3.0.

5127 **Attributes** previously placed in the **Resource**, **Action**, and **Environment** sections of a request are
5128 placed in an **attribute** category with the following identifiers respectively. It is RECOMMENDED that they
5129 are used to list **attributes** of **resources**, **actions**, and the **environment** respectively when authoring
5130 XACML 3.0 **policies** or requests.

5131 `urn:oasis:names:tc:xacml:3.0:attribute-category:resource`

5132 `urn:oasis:names:tc:xacml:3.0:attribute-category:action`

5133 `urn:oasis:names:tc:xacml:3.0:attribute-category:environment`

5134 **Attributes** previously placed in the **Subject** section of a request are placed in an **attribute** category
5135 which is identical of the **subject** category in XACML 2.0, as defined below. It is RECOMMENDED that
5136 they are used to list **attributes** of **subjects** when authoring XACML 3.0 **policies** or requests.

5137 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity in a
5138 request chain. If **subject** category is not specified in XACML 2.0, this is the default translation value.

5139 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

5140 This identifier indicates the system entity that will receive the results of the request (used when it is
5141 distinct from the access-**subject**).

5142 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

5143 This identifier indicates a system entity through which the **access** request was passed. There may be
5144 more than one. No means is provided to specify the order in which they passed the message.

5145 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

5146 This identifier indicates a system entity associated with a local or remote codebase that generated the
5147 request. Corresponding **subject attributes** might include the URL from which it was loaded and/or the
5148 identity of the code-signer. There may be more than one. No means is provided to specify the order in
5149 which they processed the request.

5150 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

5151 This identifier indicates a system entity associated with the computer that initiated the **access** request.
5152 An example would be an IPsec identity.

5153 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

5154 B.3 Data-types

5155 The following identifiers indicate data-types that are defined in Section A.2.

5156 `urn:oasis:names:tc:xacml:1.0:data-type:x500Name`.

5157 `urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`

5158 `urn:oasis:names:tc:xacml:2.0:data-type:ipAddress`

5159 `urn:oasis:names:tc:xacml:2.0:data-type:dnsName`

5160 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression

5161 The following data-type identifiers are defined by XML Schema [XS].

5162 <http://www.w3.org/2001/XMLSchema#string>

5163 <http://www.w3.org/2001/XMLSchema#boolean>

5164 <http://www.w3.org/2001/XMLSchema#integer>

5165 <http://www.w3.org/2001/XMLSchema#double>

5166 <http://www.w3.org/2001/XMLSchema#time>

5167 <http://www.w3.org/2001/XMLSchema#date>

5168 <http://www.w3.org/2001/XMLSchema#dateTime>

5169 <http://www.w3.org/2001/XMLSchema#anyURI>

5170 <http://www.w3.org/2001/XMLSchema#hexBinary>

5171 <http://www.w3.org/2001/XMLSchema#base64Binary>

5172 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.

5174 <http://www.w3.org/2001/XMLSchema#dayTimeDuration>

5175 <http://www.w3.org/2001/XMLSchema#yearMonthDuration>

5176 B.4 Subject attributes

5177 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).

5179 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.

5181 This identifier indicates the name of the **subject**.

5182 urn:oasis:names:tc:xacml:1.0:subject:subject-id

5183 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.

5185 urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier

5186 This identifier indicates a public key used to confirm the **subject's** identity.

5187 urn:oasis:names:tc:xacml:1.0:subject:key-info

5188 This identifier indicates the time at which the **subject** was authenticated.

5189 urn:oasis:names:tc:xacml:1.0:subject:authentication-time

5190 This identifier indicates the method used to authenticate the **subject**.

5191 urn:oasis:names:tc:xacml:1.0:subject:authentication-method

5192 This identifier indicates the time at which the **subject** initiated the **access** request, according to the **PEP**.

5193 urn:oasis:names:tc:xacml:1.0:subject:request-time

5194 This identifier indicates the time at which the **subject's** current session began, according to the **PEP**.

5195 urn:oasis:names:tc:xacml:1.0:subject:session-start-time

5196 The following identifiers indicate the location where authentication credentials were activated.

5197 This identifier indicates that the location is expressed as an IP address.

5198 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address

5199 The corresponding **attribute** SHALL be of data-type "<http://www.w3.org/2001/XMLSchema#string>".

5200 This identifier indicates that the location is expressed as a DNS name.

5201 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name

5202 The corresponding **attribute** SHALL be of data-type "<http://www.w3.org/2001/XMLSchema#string>".

5203 Where a suitable **attribute** is already defined in LDAP [LDAP-1], [LDAP-2], the XACML identifier SHALL
5204 be formed by adding the **attribute** name to the URI of the LDAP specification. For example, the **attribute**
5205 name for the userPassword defined in the RFC 2256 SHALL be:
5206 `http://www.ietf.org/rfc/rfc2256.txt#userPassword`

5207 B.5 Resource attributes

5208 These identifiers indicate **attributes** of the **resource**. When used, it is RECOMMENDED they appear
5209 within the <Attributes> element of the request **context** with Category
5210 `urn:oasis:names:tc:xacml:3.0:attribute-category:resource`.

5211 This **attribute** identifies the **resource** to which **access** is requested.

5212 `urn:oasis:names:tc:xacml:1.0:resource:resource-id`

5213 This **attribute** identifies the namespace of the top element(s) of the contents of the <Content> element.
5214 In the case where the **resource** content is supplied in the request **context** and the **resource**
5215 namespaces are defined in the **resource**, the **PEP** MAY provide this **attribute** in the request to indicate
5216 the namespaces of the **resource** content. In this case there SHALL be one value of this **attribute** for
5217 each unique namespace of the top level elements in the <Content> element. The type of the
5218 corresponding **attribute** SHALL be “`http://www.w3.org/2001/XMLSchema#anyURI`”.

5219 `urn:oasis:names:tc:xacml:2.0:resource:target-namespace`

5220 B.6 Action attributes

5221 These identifiers indicate **attributes** of the **action** being requested. When used, it is RECOMMENDED
5222 they appear within the <Attributes> element of the request **context** with Category
5223 `urn:oasis:names:tc:xacml:3.0:attribute-category:action`.

5224 This **attribute** identifies the **action** for which **access** is requested.

5225 `urn:oasis:names:tc:xacml:1.0:action:action-id`

5226 Where the **action** is implicit, the value of the action-id **attribute** SHALL be

5227 `urn:oasis:names:tc:xacml:1.0:action:implied-action`

5228 This **attribute** identifies the namespace in which the action-id **attribute** is defined.

5229 `urn:oasis:names:tc:xacml:1.0:action:action-namespace`

5230 B.7 Environment attributes

5231 These identifiers indicate **attributes** of the **environment** within which the **decision request** is to be
5232 evaluated. When used in the **decision request**, it is RECOMMENDED they appear in the
5233 <Attributes> element of the request **context** with Category `urn:oasis:names:tc:xacml:3.0:attribute-`
5234 `category:environment`.

5235 This identifier indicates the current time at the **context handler**. In practice it is the time at which the
5236 request **context** was created. For this reason, if these identifiers appear in multiple places within a
5237 <Policy> or <PolicySet>, then the same value SHALL be assigned to each occurrence in the
5238 evaluation procedure, regardless of how much time elapses between the processing of the occurrences.

5239 `urn:oasis:names:tc:xacml:1.0:environment:current-time`

5240 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#time`”.

5241 `urn:oasis:names:tc:xacml:1.0:environment:current-date`

5242 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#date`”.

5243 `urn:oasis:names:tc:xacml:1.0:environment:current-dateTime`

5244 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#dateTime`”.

5245 **B.8 Status codes**

5246 The following status code values are defined.

5247 This identifier indicates success.

5248 urn:oasis:names:tc:xacml:1.0:status:ok

5249 This identifier indicates that all the **attributes** necessary to make a **policy decision** were not available (see Section 5.58).

5251 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

5252 This identifier indicates that some **attribute** value contained a syntax error, such as a letter in a numeric field.

5254 urn:oasis:names:tc:xacml:1.0:status:syntax-error

5255 This identifier indicates that an error occurred during **policy** evaluation. An example would be division by zero.

5257 urn:oasis:names:tc:xacml:1.0:status:processing-error

5258 **B.9 Combining algorithms**

5259 The deny-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5261 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides

5262 The deny-overrides **policy-combining algorithm** has the following value for the policyCombiningAlgId attribute:

5264 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides

5265 The permit-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5267 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides

5268 The permit-overrides **policy-combining algorithm** has the following value for the policyCombiningAlgId attribute:

5270 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides

5271 The first-applicable **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5273 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable

5274 The first-applicable **policy-combining algorithm** has the following value for the policyCombiningAlgId attribute:

5276 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable

5277 The only-one-applicable-policy **policy-combining algorithm** has the following value for the policyCombiningAlgId attribute:

5279 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

5280 The ordered-deny-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5282 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides

5283 The ordered-deny-overrides **policy-combining algorithm** has the following value for the policyCombiningAlgId attribute:

5285 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-overrides

5287 The ordered-permit-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5289 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-
5290 overrides

5291 The ordered-permit-overrides **policy-combining algorithm** has the following value for the
5292 policyCombiningAlgId attribute:

5293 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-
5294 overrides

5295 The deny-unless-permit **rule-combining algorithm** has the following value for the
5296 policyCombiningAlgId attribute:

5297 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit

5298 The permit-unless-deny **rule-combining algorithm** has the following value for the
5299 policyCombiningAlgId attribute:

5300 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny

5301 The deny-unless-permit **policy-combining algorithm** has the following value for the
5302 policyCombiningAlgId attribute:

5303 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit

5304 The permit-unless-deny **policy-combining algorithm** has the following value for the
5305 policyCombiningAlgId attribute:

5306 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny

5307 The legacy deny-overrides **rule-combining algorithm** has the following value for the
5308 ruleCombiningAlgId attribute:

5309 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5310 The legacy deny-overrides **policy-combining algorithm** has the following value for the
5311 policyCombiningAlgId attribute:

5312 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

5313 The legacy permit-overrides **rule-combining algorithm** has the following value for the
5314 ruleCombiningAlgId attribute:

5315 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

5316 The legacy permit-overrides **policy-combining algorithm** has the following value for the
5317 policyCombiningAlgId attribute:

5318 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

5319 The legacy ordered-deny-overrides **rule-combining algorithm** has the following value for the
5320 ruleCombiningAlgId attribute:

5321 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides

5322 The legacy ordered-deny-overrides **policy-combining algorithm** has the following value for the
5323 policyCombiningAlgId attribute:

5324 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-
5325 overrides

5326 The legacy ordered-permit-overrides **rule-combining algorithm** has the following value for the
5327 ruleCombiningAlgId attribute:

5328 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-
5329 overrides

5330 The legacy ordered-permit-overrides **policy-combining algorithm** has the following value for the
5331 policyCombiningAlgId attribute:

5332 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-
5333 overrides

5334

5335

C. Combining algorithms (normative)

5336 This section contains a description of the **rule-** and **policy-combining algorithms** specified by XACML.
5337 Pseudo code is normative, descriptions in English are non-normative.

5338 The legacy **combining algorithms** are defined in previous versions of XACML, and are retained for
5339 compatibility reasons. It is RECOMMENDED that the new **combining algorithms** are used instead of the
5340 legacy **combining algorithms** for new use.

5341 C.1 Extended Indeterminate value

5342 Some combining algorithms are defined in terms of an extended set of "Indeterminate" values. For these
5343 algorithms, the **PDP** MUST keep track of the extended set of "Indeterminate" values during **rule** and
5344 **policy** combining. The extended set associated with the "Indeterminate" contains the potential effect
5345 values which could have occurred if there would not have been an error causing the "Indeterminate". The
5346 possible extended set "Indeterminate" values are

- 5347 • "Indeterminate{D}": an "Indeterminate" from a **policy** or **rule** which could have evaluated to "Deny",
5348 but not "Permit"
- 5349 • "Indeterminate{P}": an "Indeterminate" from a **policy** or **rule** which could have evaluated to "Permit",
5350 but not "Deny"
- 5351 • "Indeterminate{DP}": an "Indeterminate" from a **policy** or **rule** which could have evaluated to "Deny"
5352 or "Permit".

5353 The combining algorithms which are defined in terms of the extended "Indeterminate" make use of the
5354 additional information to allow for better treatment of errors in the algorithms.

5355 The following define the base cases for rule evaluation:

- 5356 • A **rule** which evaluates to "Indeterminate" and has Effect="Permit" results in an
5357 "Indeterminate{P}".
- 5358 • A **rule** which evaluates to "Indeterminate" and has Effect="Deny" results in an "Indeterminate{D}".

5359 C.2 Deny-overrides

5360 This section defines the "Deny-overrides" **rule-combining algorithm** of a **policy** and **policy-combining**
5361 **algorithm** of a **policy set**.

5362 This **combining algorithm** makes use of the extended "Indeterminate".

5363 The **rule combining algorithm** defined here has the following identifier:

5364 `urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides`

5365 The **policy combining algorithm** defined here has the following identifier:

5366 `urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides`

5367 The following is a non-normative informative description of this **combining algorithm**.

5368 The deny overrides **combining algorithm** is intended for those cases where a deny
5369 decision should have priority over a permit decision. This algorithm has the following
5370 behavior.

- 5371 1. If any decision is "Deny", the result is "Deny".
- 5372 2. Otherwise, if any decision is "Indeterminate{DP}", the result is "Indeterminate{DP}".
- 5373 3. Otherwise, if any decision is "Indeterminate{D}" and another decision is "Indeterminate{P} or
5374 Permit, the result is "Indeterminate{DP}".
- 5375 4. Otherwise, if any decision is "Indeterminate{D}", the result is "Indeterminate{D}".
- 5376 5. Otherwise, if any decision is "Permit", the result is "Permit".

5377 6. Otherwise, if any decision is "Indeterminate{P}", the result is "Indeterminate{P}".

5378 7. Otherwise, the result is "NotApplicable".

5379 The following pseudo-code represents the normative specification of this **combining algorithm**.

```
5380 Decision denyOverridesCombiningAlgorithm(Decision[] decisions)
5381 {
5382     Boolean atLeastOneErrorD = false;
5383     Boolean atLeastOneErrorP = false;
5384     Boolean atLeastOneErrorDP = false;
5385     Boolean atLeastOnePermit = false;
5386     for( i=0 ; i < lengthOf(decisions) ; i++ )
5387     {
5388         Decision decision = decisions[i];
5389         if (decision == Deny)
5390         {
5391             return Deny;
5392         }
5393         if (decision == Permit)
5394         {
5395             atLeastOnePermit = true;
5396             continue;
5397         }
5398         if (decision == NotApplicable)
5399         {
5400             continue;
5401         }
5402         if (decision == Indeterminate{D})
5403         {
5404             atLeastOneErrorD = true;
5405             continue;
5406         }
5407         if (decision == Indeterminate{P})
5408         {
5409             atLeastOneErrorP = true;
5410             continue;
5411         }
5412         if (decision == Indeterminate{DP})
5413         {
5414             atLeastOneErrorDP = true;
5415             continue;
5416         }
5417     }
5418     if (atLeastOneErrorDP)
5419     {
5420         return Indeterminate{DP};
5421     }
5422     if (atLeastOneErrorD && (atLeastOneErrorP || atLeastOnePermit))
5423     {
5424         return Indeterminate{DP};
5425     }
5426     if (atLeastOneErrorD)
5427     {
5428         return Indeterminate{D};
5429     }
5430     if (atLeastOnePermit)
5431     {
5432         return Permit;
5433     }
5434     if (atLeastOneErrorP)
5435     {
5436         return Indeterminate{P};
5437     }
5438     return NotApplicable;
```

5439

```
}

```

5440 **Obligations** and **advice** shall be combined as described in Section 7.16.

5441 C.3 Ordered-deny-overrides

5442 The following specification defines the "Ordered-deny-overrides" **rule-combining algorithm** of a **policy**.

5443 The behavior of this algorithm is identical to that of the "Deny-overrides" **rule-combining**
5444 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL
5445 match the order as listed in the **policy**.

5446 The **rule combining algorithm** defined here has the following identifier:

5447 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides

5448 The following specification defines the "Ordered-deny-overrides" **policy-combining algorithm** of a
5449 **policy set**.

5450 The behavior of this algorithm is identical to that of the "Deny-overrides" **policy-combining**
5451 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL
5452 match the order as listed in the **policy set**.

5453 The **policy combining algorithm** defined here has the following identifier:

5454 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-
5455 overrides

5456 C.4 Permit-overrides

5457 This section defines the "Permit-overrides" **rule-combining algorithm** of a **policy** and **policy-combining**
5458 **algorithm** of a **policy set**.

5459 This **combining algorithm** makes use of the extended "Indeterminate".

5460 The **rule combining algorithm** defined here has the following identifier:

5461 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides

5462 The **policy combining algorithm** defined here has the following identifier:

5463 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides

5464 The following is a non-normative informative description of this combining algorithm.

5465 The permit overrides **combining algorithm** is intended for those cases where a permit
5466 decision should have priority over a deny decision. This algorithm has the following
5467 behavior.

- 5468 1. If any decision is "Permit", the result is "Permit".
- 5469 2. Otherwise, if any decision is "Indeterminate{DP}", the result is "Indeterminate{DP}".
- 5470 3. Otherwise, if any decision is "Indeterminate{P}" and another decision is
5471 "Indeterminate{D} or Deny, the result is "Indeterminate{DP}".
- 5472 4. Otherwise, if any decision is "Indeterminate{P}", the result is "Indeterminate{P}".
- 5473 5. Otherwise, if decision is "Deny", the result is "Deny".
- 5474 6. Otherwise, if any decision is "Indeterminate{D}", the result is "Indeterminate{D}".
- 5475 7. Otherwise, the result is "NotApplicable".

5476 The following pseudo-code represents the normative specification of this **combining algorithm**.

```
5477 Decision permitOverridesCombiningAlgorithm(Decision[] decisions)
5478 {
5479     Boolean atLeastOneErrorD = false;
5480     Boolean atLeastOneErrorP = false;
5481     Boolean atLeastOneErrorDP = false;
```

```

5482 Boolean atLeastOneDeny = false;
5483 for( i=0 ; i < lengthOf(decisions) ; i++ )
5484 {
5485     Decision decision = decisions[i];
5486     if (decision == Deny)
5487     {
5488         atLeastOneDeny = true;
5489         continue;
5490     }
5491     if (decision == Permit)
5492     {
5493         return Permit;
5494     }
5495     if (decision == NotApplicable)
5496     {
5497         continue;
5498     }
5499     if (decision == Indeterminate{D})
5500     {
5501         atLeastOneErrorD = true;
5502         continue;
5503     }
5504     if (decision == Indeterminate{P})
5505     {
5506         atLeastOneErrorP = true;
5507         continue;
5508     }
5509     if (decision == Indeterminate{DP})
5510     {
5511         atLeastOneErrorDP = true;
5512         continue;
5513     }
5514 }
5515 if (atLeastOneErrorDP)
5516 {
5517     return Indeterminate{DP};
5518 }
5519 if (atLeastOneErrorP && (atLeastOneErrorD || atLeastOneDeny))
5520 {
5521     return Indeterminate{DP};
5522 }
5523 if (atLeastOneErrorP)
5524 {
5525     return Indeterminate{P};
5526 }
5527 if (atLeastOneDeny)
5528 {
5529     return Deny;
5530 }
5531 if (atLeastOneErrorD)
5532 {
5533     return Indeterminate{D};
5534 }
5535 return NotApplicable;
5536 }

```

5537 **Obligations** and **advice** shall be combined as described in Section 7.16.

5538 C.5 Ordered-permit-overrides

5539 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a **policy**.

5540 The behavior of this algorithm is identical to that of the “Permit-overrides” **rule-combining**
5541 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL
5542 match the order as listed in the **policy**.

5543 The **rule combining algorithm** defined here has the following identifier:

5544 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-
5545 overrides

5546 The following specification defines the “Ordered-permit-overrides” **policy-combining algorithm** of a
5547 **policy set**.

5548 The behavior of this algorithm is identical to that of the “Permit-overrides” **policy-combining**
5549 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL
5550 match the order as listed in the **policy set**.

5551 The **policy combining algorithm** defined here has the following identifier:

5552 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-
5553 overrides

5554 C.6 Deny-unless-permit

5555 This section defines the “Deny-unless-permit” **rule-combining algorithm** of a **policy** or **policy-**
5556 **combining algorithm** of a **policy set**.

5557 The **rule combining algorithm** defined here has the following identifier:

5558 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit

5559 The **policy combining algorithm** defined here has the following identifier:

5560 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit

5561 The following is a non-normative informative description of this **combining algorithm**.

5562 The “Deny-unless-permit” **combining algorithm** is intended for those cases where a
5563 permit decision should have priority over a deny decision, and an “Indeterminate” or
5564 “NotApplicable” must never be the result. It is particularly useful at the top level in a
5565 **policy** structure to ensure that a **PDP** will always return a definite “Permit” or “Deny”
5566 result. This algorithm has the following behavior.

- 5567 1. If any decision is "Permit", the result is "Permit".
- 5568 2. Otherwise, the result is "Deny".

5569 The following pseudo-code represents the normative specification of this **combining algorithm**.

```
5570 Decision denyUnlessPermitCombiningAlgorithm(Decision[] decisions)  
5571 {  
5572     for( i=0 ; i < lengthOf(decisions) ; i++ )  
5573     {  
5574         if (decisions[i] == Permit)  
5575         {  
5576             return Permit;  
5577         }  
5578     }  
5579     return Deny;  
5580 }
```

5581 **Obligations** and **advice** shall be combined as described in Section 7.16.

5582 C.7 Permit-unless-deny

5583 This section defines the “Permit-unless-deny” **rule-combining algorithm** of a **policy** or **policy-**
5584 **combining algorithm** of a **policy set**.

5585 The **rule combining algorithm** defined here has the following identifier:

5586 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny

5587 The **policy combining algorithm** defined here has the following identifier:

5588 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny

5589 The following is a non-normative informative description of this **combining algorithm**.

5590 The "Permit-unless-deny" **combining algorithm** is intended for those cases where a
5591 deny decision should have priority over a permit decision, and an "Indeterminate" or
5592 "NotApplicable" must never be the result. It is particularly useful at the top level in a
5593 **policy** structure to ensure that a **PDP** will always return a definite "Permit" or "Deny"
5594 result. This algorithm has the following behavior.

- 5595 1. If any decision is "Deny", the result is "Deny".
5596 2. Otherwise, the result is "Permit".

5597 The following pseudo-code represents the normative specification of this **combining algorithm**.

```
5598 Decision permitUnlessDenyCombiningAlgorithm(Decision[] decisions)
5599 {
5600     for( i=0 ; i < lengthOf(decisions) ; i++ )
5601     {
5602         if (decisions[i] == Deny)
5603         {
5604             return Deny;
5605         }
5606     }
5607     return Permit;
5608 }
```

5609 **Obligations** and **advice** shall be combined as described in Section 7.16.

5610 C.8 First-applicable

5611 This section defines the "First-applicable" **rule-combining algorithm** of a **policy** and **policy-combining**
5612 **algorithm** of a **policy set**.

5613 The **rule combining algorithm** defined here has the following identifier:

5614 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable

5615 The following is a non-normative informative description of the "First-Applicable" **rule-combining**
5616 **algorithm** of a **policy**.

5617 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a particular
5618 **rule**, if the **target** matches and the **condition** evaluates to "True", then the evaluation of the
5619 **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the result of the evaluation
5620 of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected in the evaluation, if the
5621 **target** evaluates to "False" or the **condition** evaluates to "False", then the next **rule** in the order
5622 SHALL be evaluated. If no further **rule** in the order exists, then the **policy** SHALL evaluate to
5623 "NotApplicable".

5624 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation SHALL
5625 halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error status.

5626 The following pseudo-code represents the normative specification of this **rule-combining algorithm**.

```
5627 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule[] rules)
5628 {
5629     for( i = 0 ; i < lengthOf(rules) ; i++ )
5630     {
5631         Decision decision = evaluate(rules[i]);
5632         if (decision == Deny)
5633         {
5634             return Deny;
5635         }
5636     }
5637 }
```

```

5636     if (decision == Permit)
5637     {
5638         return Permit;
5639     }
5640     if (decision == NotApplicable)
5641     {
5642         continue;
5643     }
5644     if (decision == Indeterminate)
5645     {
5646         return Indeterminate;
5647     }
5648 }
5649 return NotApplicable;
5650 }

```

5651 The **policy combining algorithm** defined here has the following identifier:

5652 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable

5653 The following is a non-normative informative description of the "First-applicable" **policy-combining algorithm** of a **policy set**.

5655 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular **policy**, if
5656 the **target** evaluates to "True" and the **policy** evaluates to a determinate value of "Permit" or
5657 "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to the **effect** value of
5658 that **policy**. For a particular **policy**, if the **target** evaluate to "False", or the **policy** evaluates to
5659 "NotApplicable", then the next **policy** in the order SHALL be evaluated. If no further **policy** exists
5660 in the order, then the **policy set** SHALL evaluate to "NotApplicable".

5661 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**, the
5662 reference to the **policy** is considered invalid, or the **policy** itself evaluates to "Indeterminate",
5663 then the evaluation of the **policy-combining algorithm** shall halt, and the **policy set** shall
5664 evaluate to "Indeterminate" with an appropriate error status.

5665 The following pseudo-code represents the normative specification of this policy-combination algorithm.

```

5666 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy[] policies)
5667 {
5668     for( i = 0 ; i < lengthOf(policies) ; i++ )
5669     {
5670         Decision decision = evaluate(policies[i]);
5671         if(decision == Deny)
5672         {
5673             return Deny;
5674         }
5675         if(decision == Permit)
5676         {
5677             return Permit;
5678         }
5679         if (decision == NotApplicable)
5680         {
5681             continue;
5682         }
5683         if (decision == Indeterminate)
5684         {
5685             return Indeterminate;
5686         }
5687     }
5688     return NotApplicable;
5689 }

```

5690 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.16.

5691 C.9 Only-one-applicable

5692 This section defines the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

5693 The **policy combining algorithm** defined here has the following identifier:

5694 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

5695 The following is a non-normative informative description of the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

5697 In the entire set of **policies** in the **policy set**, if no **policy** is considered applicable by virtue of its **target**, then the result of the policy-combination algorithm SHALL be "NotApplicable". If more than one **policy** is considered applicable by virtue of its **target**, then the result of the policy-combination algorithm SHALL be "Indeterminate".

5701 If only one **policy** is considered applicable by evaluation of its **target**, then the result of the **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

5703 If an error occurs while evaluating the **target** of a **policy**, or a reference to a **policy** is considered invalid or the **policy** evaluation results in "Indeterminate", then the **policy set** SHALL evaluate to "Indeterminate", with the appropriate error status.

5706 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```
5707 Decision onlyOneApplicablePolicyPolicyCombiningAlogrithm(Policy[] policies)
5708 {
5709     Boolean          atLeastOne      = false;
5710     Policy           selectedPolicy = null;
5711     ApplicableResult appResult;
5712
5713     for ( i = 0; i < lengthOf(policies) ; i++ )
5714     {
5715         appResult = isApplicable(policies[i]);
5716
5717         if ( appResult == Indeterminate )
5718         {
5719             return Indeterminate;
5720         }
5721         if( appResult == Applicable )
5722         {
5723             if ( atLeastOne )
5724             {
5725                 return Indeterminate;
5726             }
5727             else
5728             {
5729                 atLeastOne      = true;
5730                 selectedPolicy = policies[i];
5731             }
5732         }
5733         if ( appResult == NotApplicable )
5734         {
5735             continue;
5736         }
5737     }
5738     if ( atLeastOne )
5739     {
5740         return evaluate(selectedPolicy);
5741     }
5742     else
5743     {
5744         return NotApplicable;
5745     }
5746 }
```

5747 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.16.

5748 C.10 Legacy Deny-overrides

5749 This section defines the legacy “Deny-overrides” *rule-combining algorithm* of a *policy* and *policy-*
5750 *combining algorithm* of a *policy set*.

5751

5752 The *rule combining algorithm* defined here has the following identifier:

5753 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5754 The following is a non-normative informative description of this combining algorithm.

5755 The “Deny-overrides” rule combining algorithm is intended for those cases where a deny
5756 decision should have priority over a permit decision. This algorithm has the following
5757 behavior.

- 5758 1. If any rule evaluates to "Deny", the result is "Deny".
- 5759 2. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is
5760 "Indeterminate".
- 5761 3. Otherwise, if any rule evaluates to "Permit", the result is "Permit".
- 5762 4. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate", the result is
5763 "Indeterminate".
- 5764 5. Otherwise, the result is "NotApplicable".

5765 The following pseudo-code represents the normative specification of this *rule-combining algorithm*.

```
5766 Decision denyOverridesRuleCombiningAlgorithm(Rule[] rules)
5767 {
5768     Boolean atLeastOneError = false;
5769     Boolean potentialDeny = false;
5770     Boolean atLeastOnePermit = false;
5771     for( i=0 ; i < lengthOf(rules) ; i++ )
5772     {
5773         Decision decision = evaluate(rules[i]);
5774         if (decision == Deny)
5775         {
5776             return Deny;
5777         }
5778         if (decision == Permit)
5779         {
5780             atLeastOnePermit = true;
5781             continue;
5782         }
5783         if (decision == NotApplicable)
5784         {
5785             continue;
5786         }
5787         if (decision == Indeterminate)
5788         {
5789             atLeastOneError = true;
5790
5791             if (effect(rules[i]) == Deny)
5792             {
5793                 potentialDeny = true;
5794             }
5795             continue;
5796         }
5797     }
5798     if (potentialDeny)
5799     {
5800         return Indeterminate;
5801     }
5802     if (atLeastOnePermit)
5803     {
```

```

5804     return Permit;
5805     }
5806     if (atLeastOneError)
5807     {
5808         return Indeterminate;
5809     }
5810     return NotApplicable;
5811 }

```

5812 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.16.

5813 The **policy combining algorithm** defined here has the following identifier:

5814 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

5815 The following is a non-normative informative description of this combining algorithm.

5816 The "Deny-overrides" policy combining algorithm is intended for those cases where a
5817 deny decision should have priority over a permit decision. This algorithm has the
5818 following behavior.

- 5819 1. If any policy evaluates to "Deny", the result is "Deny".
- 5820 2. Otherwise, if any policy evaluates to "Indeterminate", the result is "Deny".
- 5821 3. Otherwise, if any policy evaluates to "Permit", the result is "Permit".
- 5822 4. Otherwise, the result is "NotApplicable".

5823 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```

5824 Decision denyOverridesPolicyCombiningAlgorithm(Policy[] policies)
5825 {
5826     Boolean atLeastOnePermit = false;
5827     for( i=0 ; i < lengthOf(policies) ; i++ )
5828     {
5829         Decision decision = evaluate(policies[i]);
5830         if (decision == Deny)
5831         {
5832             return Deny;
5833         }
5834         if (decision == Permit)
5835         {
5836             atLeastOnePermit = true;
5837             continue;
5838         }
5839         if (decision == NotApplicable)
5840         {
5841             continue;
5842         }
5843         if (decision == Indeterminate)
5844         {
5845             return Deny;
5846         }
5847     }
5848     if (atLeastOnePermit)
5849     {
5850         return Permit;
5851     }
5852     return NotApplicable;
5853 }

```

5854 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.16.

5855 C.11 Legacy Ordered-deny-overrides

5856 The following specification defines the legacy "Ordered-deny-overrides" **rule-combining algorithm** of a
5857 **policy**.

5858 The behavior of this algorithm is identical to that of the “Deny-overrides” **rule-combining**
5859 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL
5860 match the order as listed in the **policy**.

5861 The **rule combining algorithm** defined here has the following identifier:

5862 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides

5863 The following specification defines the legacy “Ordered-deny-overrides” **policy-combining algorithm** of
5864 a **policy set**.

5865 The behavior of this algorithm is identical to that of the “Deny-overrides” **policy-combining**
5866 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL
5867 match the order as listed in the **policy set**.

5868 The **rule combining algorithm** defined here has the following identifier:

5869 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-
5870 overrides

5871 C.12 Legacy Permit-overrides

5872 This section defines the legacy “Permit-overrides” **rule-combining algorithm** of a **policy** and **policy-**
5873 **combining algorithm** of a **policy set**.

5874 The **rule combining algorithm** defined here has the following identifier:

5875 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

5876 The following is a non-normative informative description of this combining algorithm.

5877 The “Permit-overrides” rule combining algorithm is intended for those cases where a
5878 permit decision should have priority over a deny decision. This algorithm has the
5879 following behavior.

- 5880 1. If any rule evaluates to "Permit", the result is "Permit".
- 5881 2. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate" the result is
5882 "Indeterminate".
- 5883 3. Otherwise, if any rule evaluates to "Deny", the result is "Deny".
- 5884 4. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is
5885 "Indeterminate".
- 5886 5. Otherwise, the result is "NotApplicable".

5887 The following pseudo-code represents the normative specification of this **rule-combining algorithm**.

```
5888 Decision permitOverridesRuleCombiningAlgorithm(Rule[] rules)
5889 {
5890     Boolean atLeastOneError = false;
5891     Boolean potentialPermit = false;
5892     Boolean atLeastOneDeny = false;
5893     for( i=0 ; i < lengthOf(rules) ; i++ )
5894     {
5895         Decision decision = evaluate(rules[i]);
5896         if (decision == Deny)
5897         {
5898             atLeastOneDeny = true;
5899             continue;
5900         }
5901         if (decision == Permit)
5902         {
5903             return Permit;
5904         }
5905         if (decision == NotApplicable)
5906         {
5907             continue;
5908         }
5909     }
5910 }
```

```

5909     if (decision == Indeterminate)
5910     {
5911         atLeastOneError = true;
5912
5913         if (effect(rules[i]) == Permit)
5914         {
5915             potentialPermit = true;
5916         }
5917         continue;
5918     }
5919 }
5920 if (potentialPermit)
5921 {
5922     return Indeterminate;
5923 }
5924 if (atLeastOneDeny)
5925 {
5926     return Deny;
5927 }
5928 if (atLeastOneError)
5929 {
5930     return Indeterminate;
5931 }
5932 return NotApplicable;
5933 }

```

5934 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.16.

5935 The **policy combining algorithm** defined here has the following identifier:

5936 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

5937 The following is a non-normative informative description of this combining algorithm.

5938 The "Permit-overrides" policy combining algorithm is intended for those cases where a
5939 permit decision should have priority over a deny decision. This algorithm has the
5940 following behavior.

- 5941 1. If any policy evaluates to "Permit", the result is "Permit".
- 5942 2. Otherwise, if any policy evaluates to "Deny", the result is "Deny".
- 5943 3. Otherwise, if any policy evaluates to "Indeterminate", the result is "Indeterminate".
- 5944 4. Otherwise, the result is "NotApplicable".

5945 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```

5946 Decision permitOverridesPolicyCombiningAlgorithm(Policy[] policies)
5947 {
5948     Boolean atLeastOneError = false;
5949     Boolean atLeastOneDeny = false;
5950     for( i=0 ; i < lengthOf(policies) ; i++ )
5951     {
5952         Decision decision = evaluate(policies[i]);
5953         if (decision == Deny)
5954         {
5955             atLeastOneDeny = true;
5956             continue;
5957         }
5958         if (decision == Permit)
5959         {
5960             return Permit;
5961         }
5962         if (decision == NotApplicable)
5963         {
5964             continue;
5965         }
5966         if (decision == Indeterminate)

```

```

5967     {
5968         atLeastOneError = true;
5969         continue;
5970     }
5971 }
5972 if (atLeastOneDeny)
5973 {
5974     return Deny;
5975 }
5976 if (atLeastOneError)
5977 {
5978     return Indeterminate;
5979 }
5980 return NotApplicable;
5981 }

```

5982 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.16.

5983 C.13 Legacy Ordered-permit-overrides

5984 The following specification defines the legacy "Ordered-permit-overrides" **rule-combining algorithm** of a
5985 **policy**.

5986 The behavior of this algorithm is identical to that of the "Permit-overrides" **rule-combining**
5987 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL
5988 match the order as listed in the **policy**.

5989 The **rule combining algorithm** defined here has the following identifier:

5990 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-
5991 overrides

5992 The following specification defines the legacy "Ordered-permit-overrides" **policy-combining algorithm** of
5993 a **policy set**.

5994 The behavior of this algorithm is identical to that of the "Permit-overrides" **policy-combining**
5995 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL
5996 match the order as listed in the **policy set**.

5997 The **policy combining algorithm** defined here has the following identifier:

5998 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-
5999 overrides

6000

6001 **D. Acknowledgements**

6002 The following individuals have participated in the creation of this specification and are gratefully
6003 acknowledged:

- 6004
- 6005 Anil Saldhana
 - 6006 Anil Tappetla
 - 6007 Anne Anderson
 - 6008 Anthony Nadalin
 - 6009 Bill Parducci
 - 6010 Craig Forster
 - 6011 David Chadwick
 - 6012 David Staggs
 - 6013 Dilli Arumugam
 - 6014 Duane DeCouteau
 - 6015 Erik Rissanen
 - 6016 Gareth Richards
 - 6017 Hal Lockhart
 - 6018 Jan Herrmann
 - 6019 John Tolbert
 - 6020 Ludwig Seitz
 - 6021 Michiharu Kudo
 - 6022 Naomaru Itoi
 - 6023 Paul Tyson
 - 6024 Prateek Mishra
 - 6025 Rich Levinson
 - 6026 Ronald Jacobson
 - 6027 Seth Proctor
 - 6028 Sridhar Muppidi
 - 6029 Tim Moses
 - 6030 Vernon Murdoch

6031

E. Revision History

6032 [optional; should not be included in OASIS Standards]

6033

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 <Content> element the context node of all xpath expressions and introduced categorization of XPath expressions so they point to a specific <Content> element.</p> <p>Added <Content> element to the policy issuer.</p> <p>Added description of the <PolicyIssuer> element.</p> <p>Updated the schema figure in the introduction to reflect the new AllOf/AnyOf schema.</p> <p>Remove duplicate <CombinerParameters> element in the <Policy> element in the schema.</p> <p>Removed default attributes in the schema. (Version in <Policy(Set)> and MustBePresent in <AttributeDesignator> in <AttributeSelector>)</p> <p>Removed references in section 7.3 to the <Condition> 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 <Attribute>. Also added this attribute in the associated schema file.</p> <p>Removed description of non-existing XML attribute "ResourceId" from the element <Result>.</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 <Description> to <Apply>.</p> <p>Added XPath versioning to the request.</p> <p>Added security considerations about denial service and the access-permitted function.</p> <p>Changed <Target> 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 <Function> 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 <Advice> feature</p> <p>Made obligations mandatory (per the 19th 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 <Request></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 <AttributeAssignment></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"> • uri-starts-with to anyURI-starts-with • uri-ends-with to anyURI-ends-with • uri-contains to anyURI-contains • uri-substring to anyURI-substring <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 <Policy(Set)IdReference> in <PolicyIdList>.</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 <Content> 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.

6034

6035