



# eXtensible Access Control Markup Language (XACML) Version 3.0

## Committee Draft 04

1 July 2010

### Specification URIs:

#### This 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>

#### Previous Version:

<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-03-en.html>  
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-03-en.doc> (Authoritative)  
<http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-03-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](mailto:bill@parducci.net)>  
Hal Lockhart, Oracle <[hal.lockhart@oracle.com](mailto:hal.lockhart@oracle.com)>

### Editor:

Erik Rissanen, Axiomatics AB <[erik@axiomatics.com](mailto: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  
44 particular **subject, resource** or **action**
- 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  
49 **attribute** holder (which may be of type: **subject, resource, action** or **environment**) and  
50 (optionally) the identity of the issuing authority
- 51 **Obligation**
- 52 An operation specified in a **rule, policy** or **policy set** that should be performed by the **PEP** in  
53 conjunction with the enforcement of an **authorization decision**
- 54 **Policy**
- 55 A set of **rules**, an identifier for the **rule-combining algorithm** and (optionally) a set of  
56 **obligations** or **advice**. May be a component of a **policy set**
- 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  
64 Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198].  
65 This term corresponds to "Access Decision Function" (ADF) in [ISO10181-3].
- 66 **Policy enforcement point (PEP)**
- 67 The system entity that performs **access control**, by making **decision requests** and enforcing  
68 **authorization decisions**. This term is defined in a joint effort by the IETF Policy Framework  
69 Working Group and the Distributed Management Task Force (DMTF)/Common Information Model  
70 (CIM) in [RFC3198]. This term corresponds to "Access Enforcement Function" (AEF) in  
71 [ISO10181-3].
- 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  
76 **obligations** or **advice**. May be a component of another **policy set**
- 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].

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

### 1.3 Schema organization and namespaces

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

### 1.4 Normative References

**[CMF]** Martin J. Dürst et al, eds., *Character Model for the World Wide Web 1.0: Fundamentals*, W3C Recommendation 15 February 2005, <http://www.w3.org/TR/2005/REC-charmod-20050215/>

**[DS]** D. Eastlake et al., *XML-Signature Syntax and Processing*, <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.

**[exc-c14n]** J. Boyer et al, eds., *Exclusive XML Canonicalization, Version 1.0*, W3C Recommendation 18 July 2002, <http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/>

**[Hancock]** Hancock, *Polymorphic Type Checking*, in Simon L. Peyton Jones, *Implementation of Functional Programming Languages*, Section 8, Prentice-Hall International, 1987.

**[Haskell]** Haskell, a purely functional language. Available at <http://www.haskell.org/>

**[Hier]** OASIS Committee Draft 03, *XACML v3.0 Hierarchical Resource Profile Version 1.0*, 11 March 2010, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-hierarchical-v1-spec-cd-03-en.doc>

**[IEEE754]** IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-7653-8, IEEE Product No. SH10116-TBR.

**[ISO10181-3]** ISO/IEC 10181-3:1996 Information technology – Open Systems Interconnection - Security frameworks for open systems: Access control framework.

**[Kudo00]** Kudo M and Hada S, *XML document security based on provisional authorization*, Proceedings of the Seventh ACM Conference on Computer and Communications Security, Nov 2000, Athens, Greece, pp 87-96.

**[LDAP-1]** RFC2256, *A summary of the X500(96) User Schema for use with LDAPv3*, Section 5, M Wahl, December 1997, <http://www.ietf.org/rfc/rfc2256.txt>

**[LDAP-2]** RFC2798, *Definition of the inetOrgPerson*, M. Smith, April 2000 <http://www.ietf.org/rfc/rfc2798.txt>

**[MathML]** *Mathematical Markup Language (MathML)*, Version 2.0, W3C Recommendation, 21 October 2003. Available at: <http://www.w3.org/TR/2003/REC-MathML-20031021/>

**[Multi]** OASIS Committee Draft 03, *XACML v3.0 Multiple Decision Profile Version 1.0*, 11 March 2010, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-multiple-v1-spec-cd-03-en.doc>

**[Perritt93]** Perritt, H. Knowbots, *Permissions Headers and Contract Law*, Conference on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993. Available at: <http://www.ifla.org/documents/infopol/copyright/perh2.txt>

**[RBAC]** David Ferraiolo and Richard Kuhn, *Role-Based Access Controls*, 15th National Computer Security Conference, 1992.

**[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

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

## 207 1.5 Non-Normative References

208	<b>[CM]</b>	<i>Character model model for the World Wide Web 1.0: Normalization</i> , W3C Working Draft, 27 October 2005, <a href="http://www.w3.org/TR/2005/WD-charmod-norm-20051027/">http://www.w3.org/TR/2005/WD-charmod-norm-20051027/</a> , World Wide Web Consortium.
209		
210		
211	<b>[Hinton94]</b>	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.
212		
213		
214	<b>[Sloman94]</b>	Sloman, M. <i>Policy Driven Management for Distributed Systems</i> . Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.
215		

216

## 2 Background (non-normative)

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

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

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

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

### 2.1 Requirements

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

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

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

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

## 267 2.2 Rule and policy combining

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

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

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

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

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

## 286 2.3 Combining algorithms

287 XACML defines a number of combining algorithms that can be identified by a RuleCombiningAlgId or  
288 PolicyCombiningAlgId attribute of the <Policy> or <PolicySet> elements, respectively. The  
289 **rule-combining algorithm** defines a procedure for arriving at an **authorization decision** given the  
290 individual results of evaluation of a set of **rules**. Similarly, the **policy-combining algorithm** defines a  
291 procedure for arriving at an **authorization decision** given the individual results of evaluation of a set of  
292 **policies**. Standard combining algorithms are defined for:

- 293 • Deny-overrides (Ordered and Unordered),
- 294 • Permit-overrides (Ordered and Unordered),
- 295 • First-applicable and
- 296 • Only-one-applicable.

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

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

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

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

309 applicable, the result of the combining algorithm is the result of evaluating the single **applicable policy** or  
310 **policy set**.

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

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

## 314 2.4 Multiple subjects

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

## 321 2.5 Policies based on subject and resource attributes

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

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

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

## 338 2.6 Multi-valued attributes

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

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

## 348 2.7 Policies based on resource contents

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

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

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

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

## 359 2.8 Operators

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

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

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

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

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

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

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

## 388 2.9 Policy distribution

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

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

## 400 2.10 Policy indexing

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

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

405 Two approaches are supported:

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

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

## 414 2.11 Abstraction layer

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

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

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

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

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

## 436 2.12 Actions performed in conjunction with enforcement

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

## 446 2.13 Supplemental information about a decision

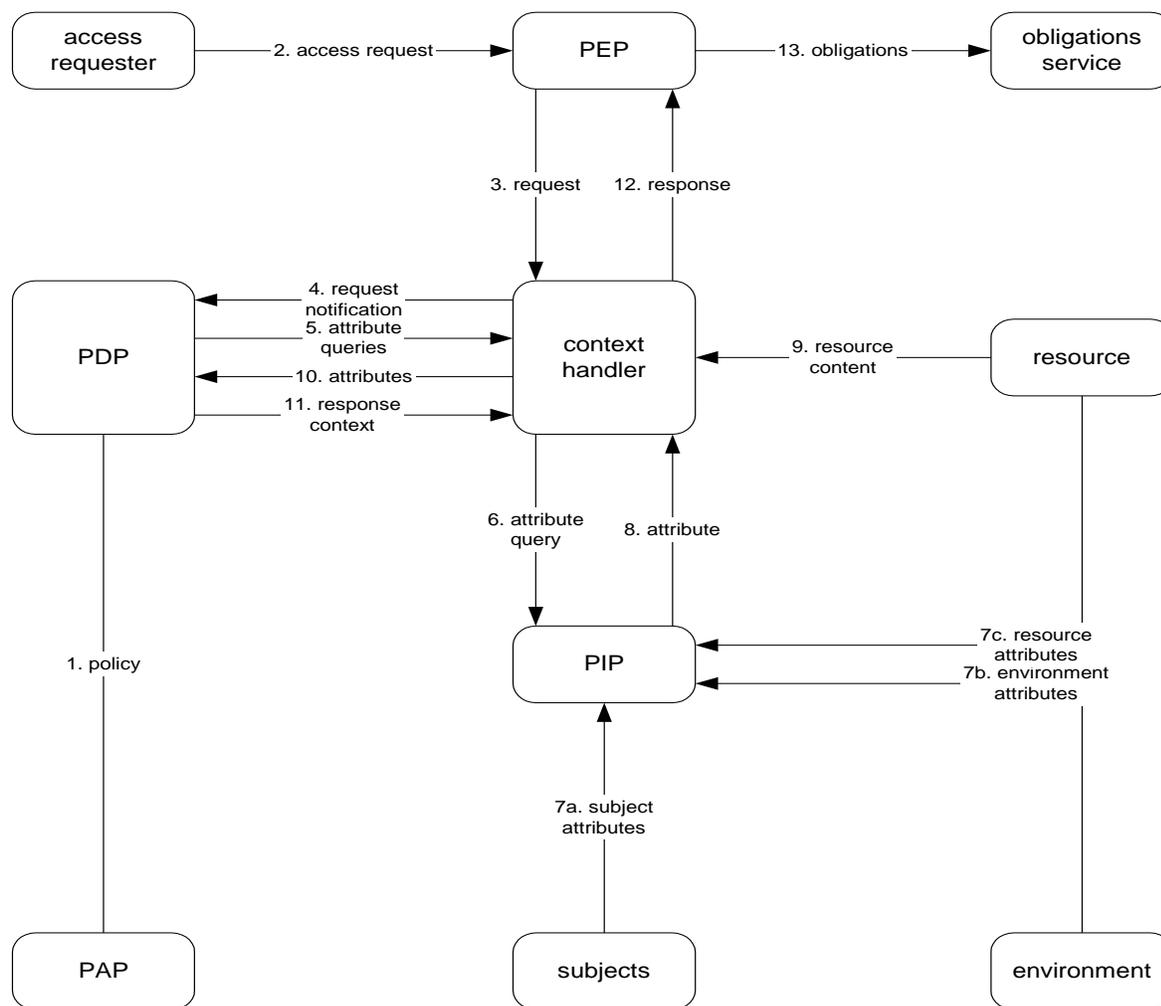
447 In some applications it is helpful to specify supplemental information about a decision. XACML provides  
448 facilities to specify supplemental information about a decision with the <Advice> element. Such **advice**  
449 may be safely ignored by the **PEP**.

### 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.



454  
455 Figure 1 - Data-flow diagram

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

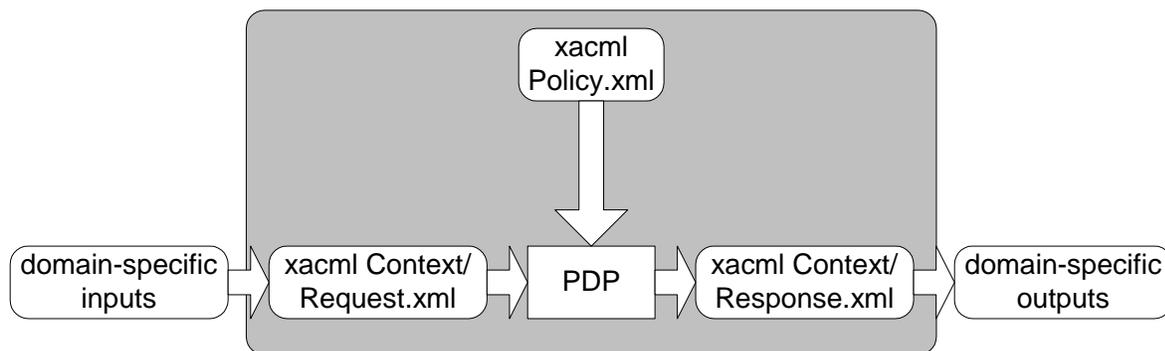
462 The model operates by the following steps.

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

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

## 485 3.2 XACML context

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



497  
498 *Figure 2 - XACML context*

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

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

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

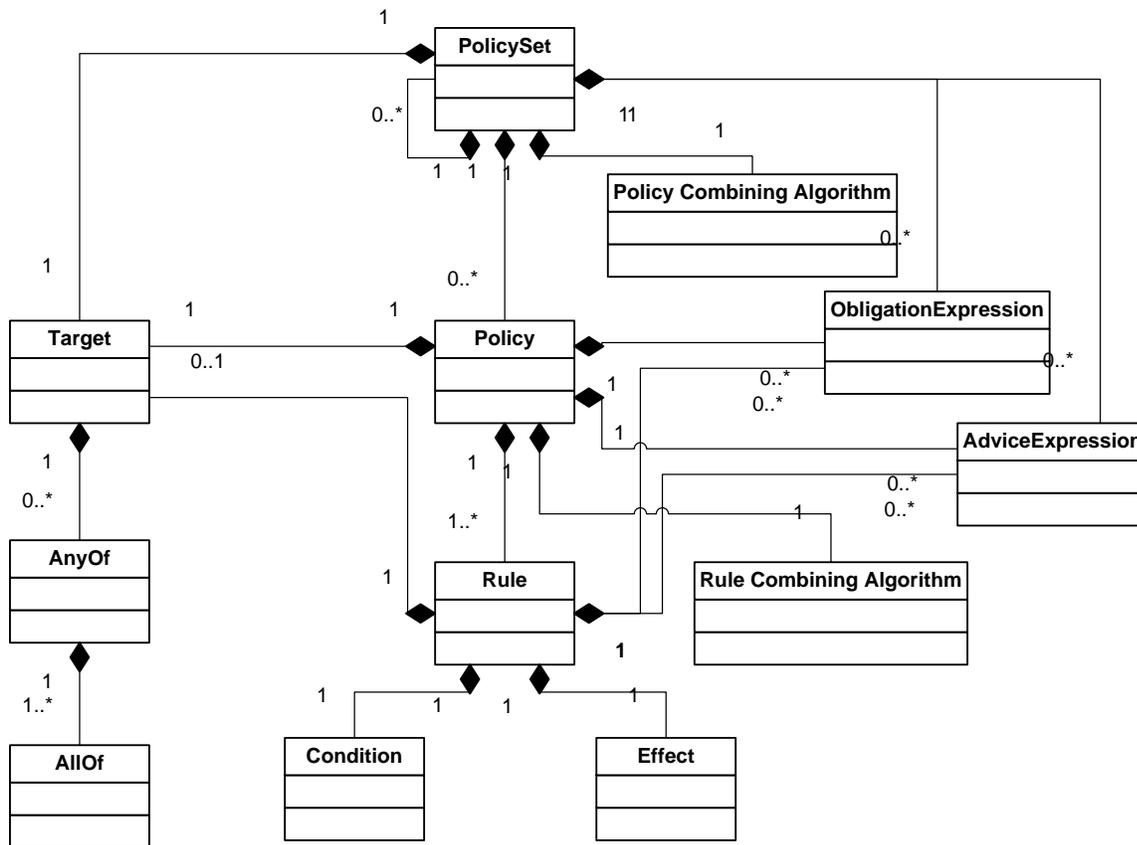
504 **3.3 Policy language model**

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

- 506 • **Rule**;
- 507 • **Policy**; and
- 508 • **Policy set**.

509 These are described in the following sub-sections.

510



511

512 *Figure 3 - Policy language model*

513 **3.3.1 Rule**

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

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

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

- 517 • a **target**,
- 518 • an **effect**,
- 519 • a **condition**,
- 520 • **obligation** expressions, and
- 521 • **advice** expressions

522 These are discussed in the following sub-sections.

### 523 3.3.1.1 Rule target

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

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

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

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

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

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

### 549 3.3.1.2 Effect

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

### 552 3.3.1.3 Condition

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

### 555 3.3.1.4 Obligation expressions

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

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

### 560 3.3.1.5 Advice

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

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

## 565 3.3.2 Policy

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

- 568 • a *target*;
  - 569 • a *rule-combining algorithm*-identifier;
  - 570 • a set of *rules*;
  - 571 • *obligation* expressions and
  - 572 • *advice* expressions
- 573 *Rules* are described above. The remaining components are described in the following sub-sections.

### 574 3.3.2.1 Policy target

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

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

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

### 594 3.3.2.2 Rule-combining algorithm

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

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

### 600 3.3.2.3 Obligation expressions

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

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

### 605 3.3.2.4 Advice

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

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

### 610 3.3.3 Policy set

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

- 612 • a **target**;
- 613 • a **policy-combining algorithm**-identifier
- 614 • a set of **policies**;
- 615 • **obligation** expressions, and
- 616 • **advice** expressions

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

#### 619 3.3.3.1 Policy-combining algorithm

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

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

#### 626 3.3.3.2 Obligation expressions

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

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

#### 632 3.3.3.3 Advice expressions

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

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

638

## 4 Examples (non-normative)

639

This section contains two examples of the use of XACML for illustrative purposes. The first example is a relatively simple one to illustrate the use of **target**, **context**, matching functions and **subject attributes**.

641

The second example additionally illustrates the use of the **rule-combining algorithm**, **conditions** and **obligations**.

642

643

### 4.1 Example one

644

#### 4.1.1 Example policy

645

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

646

647

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

648

649

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

650

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

691

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

692

693

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

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

## 737 4.1.2 Example request context

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

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

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

```

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

```

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

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

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

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

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

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

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

### 801 4.1.3 Example response context

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

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

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

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

819 [c6] closes the response *context*.

## 820 4.2 Example two

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

### 824 4.2.1 Example medical record instance

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

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

```

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

```

## 909 4.2.2 Example request context

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

```

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

```

```

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

```

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

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

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

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

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

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

999 [e30] - [e42] **Resource** content. The XML **resource** instance, **access** to all or part of which may be  
1000 requested, is placed here.

1001 [e43] - [e49] The identifier of the **Resource** instance for which **access** is requested, which is an XPath  
1002 expression into the <Content> element that selects the data to be accessed.

1003 [e57] - [e64] **Action attributes** are placed in the urn:oasis:names:tc:xacml:3.0:attribute-category:action  
1004 **attribute** category of the <Request> element.

1005 [e59] - [e63] **Action** identifier.

## 1006 4.2.3 Example plain-language rules

1007 The following plain-language **rules** are to be enforced:

1008 Rule 1: A person, identified by his or her patient number, may read any record for which he or she is  
1009 the designated patient.

1010 Rule 2: A person may read any record for which he or she is the designated parent or guardian, and  
1011 for which the patient is under 16 years of age.

1012 Rule 3: A physician may write to any medical element for which he or she is the designated primary  
1013 care physician, provided an email is sent to the patient.

1014 Rule 4: An administrator shall not be permitted to read or write to medical elements of a patient  
1015 record.

1016 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

## 1017 4.2.4 Example XACML rule instances

### 1018 4.2.4.1 Rule 1

1019 **Rule 1** illustrates a simple **rule** with a single <Condition> element. It also illustrates the use of the  
1020 <VariableDefinition> element to define a function that may be used throughout the **policy**. The  
1021 following XACML <Rule> instance expresses **Rule 1**:

```
1022 [f1] <?xml version="1.0" encoding="UTF-8"?>
1023 [f2] <Policy
1024 [f3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1025 [f4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1026 [f5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1027 [f6]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1028 [f7]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:1"
1029 [f8]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1030 algorithm:deny-overrides"
1031 [f9]   Version="1.0">
1032 [f10]   <PolicyDefaults>
1033 [f11]     <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1034 [f12]   </PolicyDefaults>
1035 [f13]   <Target/>
1036 [f14]   <VariableDefinition VariableId="17590034">
1037 [f15]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1038 [f16]       <Apply
1039 [f17]         FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1040 [f18]           <AttributeDesignator
1041 [f19]             MustBePresent="false"
1042 [f20]             Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
1043 subject"
1044 [f21]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:patient-
1045 number"
1046 [f22]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1047 [f23]           </Apply>
1048 [f24]         <Apply
1049 [f25]           FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
```

```

1050 [f26]         <AttributeSelector
1051 [f27]             MustBePresent="false"
1052 [f28]             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1053 [f29]             Path="md:record/md:patient/md:patient-number/text()"
1054 [f30]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1055 [f31]         </Apply>
1056 [f32]     </Apply>
1057 [f33] </VariableDefinition>
1058 [f34] <Rule
1059 [f35]     RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"
1060 [f36]     Effect="Permit">
1061 [f37]     <Description>
1062 [f38]         A person may read any medical record in the
1063 [f39]         http://www.med.example.com/schemas/record.xsd namespace
1064 [f40]         for which he or she is the designated patient
1065 [f41]     </Description>
1066 [f42]     <Target>
1067 [f43]         <AnyOf>
1068 [f44]             <AllOf>
1069 [f45]                 <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1070 [f46]                     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1071 [f47]                         >urn:example:med:schemas:record</AttributeValue>
1072 [f48]                     <AttributeDesignator
1073 [f49]                         MustBePresent="false"
1074 [f50]                         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1075 [f51]                         AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1076 [f52]                         DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1077 [f53]                     </Match>
1078 [f54]                 <Match
1079 [f55]                     MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1080 [f56]                         <AttributeValue
1081 [f57]                             DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1082 [f58]                             XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1083 [f59]                             >md:record</AttributeValue>
1084 [f60]                         <AttributeDesignator
1085 [f61]                             MustBePresent="false"
1086 [f62]                             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1087 [f63]                             AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1088 [f64]                             DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1089 [f65]                         </Match>
1090 [f66]                     </AllOf>
1091 [f67]                 </AnyOf>
1092 [f68]             <AnyOf>
1093 [f69]                 <AllOf>
1094 [f70]                     <Match
1095 [f71]                         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1096 [f72]                             <AttributeValue
1097 [f73]                                 DataType="http://www.w3.org/2001/XMLSchema#string"
1098 [f74]                                 >read</AttributeValue>
1099 [f75]                             <AttributeDesignator
1100 [f76]                                 MustBePresent="false"
1101 [f77]                                 Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1102 [f78]                                 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1103 [f79]                                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1104 [f80]                             </Match>
1105 [f81]                         </AllOf>
1106 [f82]                     </AnyOf>
1107 [f83]                 </Target>
1108 [f84]             <Condition>
1109 [f85]                 <VariableReference VariableId="17590034"/>
1110 [f86]             </Condition>
1111 [f87]     </Rule>
1112 [f88] </Policy>

```

1113 [f3] - [f6] XML namespace declarations.

1114 [f11] XPath expressions in the **policy** are to be interpreted according to the 1.0 version of the XPath specification.

1116 [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**.

1118 [f15] The `FunctionId` attribute names the function to be used for comparison. In this case, comparison  
1119 is done with the “urn:oasis:names:tc:xacml:1.0:function:string-equal” function; this function takes two  
1120 arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1121 [f17] The first argument of the variable definition is a function specified by the `FunctionId` attribute.  
1122 Since urn:oasis:names:tc:xacml:1.0:function:string-equal takes arguments of type  
1123 “http://www.w3.org/2001/XMLSchema#string” and `AttributeDesignator` selects a **bag** of type  
1124 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-and-  
1125 only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly one  
1126 value.

1127 [f18] The `AttributeDesignator` selects a **bag** of values for the patient-number **subject attribute** in  
1128 the request **context**.

1129 [f25] The second argument of the variable definition is a function specified by the `FunctionId` attribute.  
1130 Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes arguments of type  
1131 “http://www.w3.org/2001/XMLSchema#string” and the `AttributeSelector` selects a **bag** of type  
1132 “http://www.w3.org/2001/XMLSchema#string”, “urn:oasis:names:tc:xacml:1.0:function:string-one-and-  
1133 only” is used. This function guarantees that its argument evaluates to a **bag** containing exactly one  
1134 value.

1135 [f26] The `<AttributeSelector>` element selects a **bag** of values from the **resource** content using a  
1136 free-form XPath expression. In this case, it selects the value of the patient-number in the **resource**.  
1137 Note that the namespace prefixes in the XPath expression are resolved with the standard XML  
1138 namespace declarations.

1139 [f35] **Rule** identifier.

1140 [f36] **Rule effect** declaration. When a **rule** evaluates to ‘True’ it emits the value of the `Effect` attribute.  
1141 This value is then combined with the `Effect` values of other **rules** according to the **rule-combining**  
1142 **algorithm**.

1143 [f37] - [f41] Free form description of the **rule**.

1144 [f42] - [f83] A **rule target** defines a set of **decision requests** that the **rule** is intended to evaluate.

1145 [f43] - [f67] The `<AnyOf>` element contains a **disjunctive sequence** of `<AllOf>` elements. In this  
1146 example, there is just one.

1147 [f44] - [f66] The `<AllOf>` element encloses the **conjunctive sequence** of `Match` elements. In this  
1148 example, there are two.

1149 [f45] - [f53] The first `<Match>` element compares its first and second child elements according to the  
1150 matching function. A match is positive if the value of the first argument matches any of the values  
1151 selected by the second argument. This match compares the **target** namespace of the requested  
1152 document with the value of “urn:example:med:schemas:record”.

1153 [f45] The `MatchId` attribute names the matching function.

1154 [f46] - [f47] Literal **attribute** value to match.

1155 [f48] - [f52] The `<AttributeDesignator>` element selects the **target** namespace from the **resource**  
1156 contained in the request **context**. The **attribute** name is specified by the `AttributeId`.

1157 [f54] - [f65] The second `<Match>` element. This match compares the results of two XPath expressions  
1158 applied to the `<Content>` element of the **resource** category. The second XPath expression is the  
1159 location path to the requested XML element and the first XPath expression is the literal value “md:record”.  
1160 The “xpath-node-match” function evaluates to “True” if the requested XML element is below the  
1161 “md:record” element.

1162 [f68] - [f82] The `<AnyOf>` element contains a **disjunctive sequence** of `<AllOf>` elements. In this case,  
1163 there is just one `<AllOf>` element.

1164 [f69] - [f81] The `<AllOf>` element contains a **conjunctive sequence** of `<Match>` elements. In this case,  
1165 there is just one `<Match>` element.

1166 [f70] - [f80] The <Match> element compares its first and second child elements according to the matching  
1167 function. The match is positive if the value of the first argument matches any of the values selected by  
1168 the second argument. In this case, the value of the action-id **action attribute** in the request **context** is  
1169 compared with the literal value "read".

1170 [f84] - [f86] The <Condition> element. A **condition** must evaluate to "True" for the **rule** to be  
1171 applicable. This **condition** contains a reference to a variable definition defined elsewhere in the **policy**.

#### 1172 4.2.4.2 Rule 2

1173 **Rule 2** illustrates the use of a mathematical function, i.e. the <Apply> element with functionId  
1174 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" to calculate the date of the patient's  
1175 sixteenth birthday. It also illustrates the use of **predicate** expressions, with the functionId  
1176 "urn:oasis:names:tc:xacml:1.0:function:and". This example has one function embedded in the  
1177 <Condition> element and another one referenced in a <VariableDefinition> element.

```
1178 [g1] <?xml version="1.0" encoding="UTF-8"?>
1179 [g2] <Policy
1180 [g3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1181 [g4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1182 [g5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1183 [g6]   xmlns:xf="http://www.w3.org/2005/xpath-functions"
1184 [g7]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1185 [g8]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"
1186 [g9]   Version="1.0"
1187 [g10]  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1188      algorithm:deny-overrides">
1189 [g11]  <PolicyDefaults>
1190 [g12]    <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1191 [g13]  </PolicyDefaults>
1192 [g14]  <Target/>
1193 [g15]  <VariableDefinition VariableId="17590035">
1194 [g16]    <Apply
1195 [g17]      FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal">
1196 [g18]      <Apply
1197 [g19]        FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
1198 [g20]          <AttributeDesignator
1199 [g21]            MustBePresent="false"
1200 [g22]            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
1201 [g23]            AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
1202 [g24]            DataType="http://www.w3.org/2001/XMLSchema#date"/>
1203 [g25]          </Apply>
1204 [g26]        <Apply
1205 [g27]          FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
1206 [g28]            <Apply
1207 [g29]              FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
1208 [g30]                <AttributeSelector
1209 [g31]                  MustBePresent="false"
1210 [g32]                  Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1211 [g33]                  Path="md:record/md:patient/md:patientDoB/text()"
1212 [g34]                  DataType="http://www.w3.org/2001/XMLSchema#date"/>
1213 [g35]                </Apply>
1214 [g36]              <AttributeValue
1215 [g37]                DataType="http://www.w3.org/2001/XMLSchema#yearMonthDuration"
1216 [g38]                >P16Y</AttributeValue>
1217 [g39]            </Apply>
1218 [g40]          </Apply>
1219 [g41]        </VariableDefinition>
1220 [g42]      <Rule
1221 [g43]        RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
1222 [g44]        Effect="Permit">
1223 [g45]        <Description>
1224 [g46]          A person may read any medical record in the
1225 [g47]          http://www.med.example.com/records.xsd namespace
1226 [g48]          for which he or she is the designated parent or guardian,
1227 [g49]          and for which the patient is under 16 years of age
1228 [g50]        </Description>
1229 [g51]        <Target>
1230 [g52]          <AnyOf>
1231 [g53]            <AllOf>
```

```

1232 [g54] <Match
1233 [g55]     MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1234 [g56]     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1235 [g57]     >urn:example:med:schemas:record</AttributeValue>
1236 [g58]     <AttributeDesignator
1237 [g59]     MustBePresent="false"
1238 [g60]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1239 [g61]     AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1240 [g62]     DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1241 [g63] </Match>
1242 [g64] <Match
1243 [g65]     MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1244 [g66]     <AttributeValue
1245 [g67]     DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1246 [g68]     XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1247 [g69]     >md:record</AttributeValue>
1248 [g70]     <AttributeDesignator
1249 [g71]     MustBePresent="false"
1250 [g72]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1251 [g73]     AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1252 [g74]     DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1253 [g75] </Match>
1254 [g76] </AllOf>
1255 [g77] </AnyOf>
1256 [g78] <AnyOf>
1257 [g79]     <AllOf>
1258 [g80]     <Match
1259 [g81]     MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1260 [g82]     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1261 [g83]     >read</AttributeValue>
1262 [g84]     <AttributeDesignator
1263 [g85]     MustBePresent="false"
1264 [g86]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1265 [g87]     AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1266 [g88]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1267 [g89]     </Match>
1268 [g90]     </AllOf>
1269 [g91] </AnyOf>
1270 [g92] </Target>
1271 [g93] <Condition>
1272 [g94]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1273 [g95]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1274 [g96]     <Apply
1275 [g97]     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1276 [g98]     <AttributeDesignator
1277 [g99]     MustBePresent="false"
1278 [g100]     Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1279 [g101]     AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:parent-
1280 guardian-id"
1281 [g102]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1282 [g103]     </Apply>
1283 [g104]     <Apply
1284 [g105]     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1285 [g106]     <AttributeSelector
1286 [g107]     MustBePresent="false"
1287 [g108]     Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1288 [g109]     Path="md:record/md:parentGuardian/md:parentGuardianId/text()"
1289 [g110]     DataType="http://www.w3.org/2001/XMLSchema#string"/>
1290 [g111]     </Apply>
1291 [g112]     </Apply>
1292 [g113]     <VariableReference VariableId="17590035"/>
1293 [g114]     </Apply>
1294 [g115]     </Condition>
1295 [g116] </Rule>
1296 [g117] </Policy>

```

1297 [g15] - [g41] The <VariableDefinition> element contains part of the **condition** (i.e. is the patient  
1298 under 16 years of age?). The patient is under 16 years of age if the current date is less than the date  
1299 computed by adding 16 to the patient's date of birth.

1300 [g16] - [g40] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to compare the two date  
1301 arguments.

1302 [g18] - [g25] The first date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-one-and-only” to  
 1303 ensure that the **bag** of values selected by its argument contains exactly one value of type  
 1304 “http://www.w3.org/2001/XMLSchema#date”.

1305 [g20] The current date is evaluated by selecting the “urn:oasis:names:tc:xacml:1.0:environment:current-  
 1306 date” **environment attribute**.

1307 [g26] - [g39] The second date argument uses “urn:oasis:names:tc:xacml:1.0:function:date-add-  
 1308 yearMonthDuration” to compute the date of the patient’s sixteenth birthday by adding 16 years to the  
 1309 patient’s date of birth. The first of its arguments is of type “http://www.w3.org/2001/XMLSchema#date”  
 1310 and the second is of type “http://www.w3.org/TR/2007/REC-xpath-functions-20070123/#dt-  
 1311 yearMonthDuration”.

1312 [g30] The <AttributeSelector> element selects the patient’s date of birth by taking the XPath  
 1313 expression over the **resource** content.

1314 [g36] - [g38] Year Month Duration of 16 years.

1315 [g51] - [g92] **Rule** declaration and **rule target**. See **Rule** 1 in Section 4.2.4.1 for the detailed explanation  
 1316 of these elements.

1317 [g93] - [g115] The <Condition> element. The **condition** must evaluate to “True” for the **rule** to be  
 1318 applicable. This **condition** evaluates the truth of the statement: the requestor is the designated parent or  
 1319 guardian and the patient is under 16 years of age. It contains one embedded <Apply> element and one  
 1320 referenced <VariableDefinition> element.

1321 [g94] The **condition** uses the “urn:oasis:names:tc:xacml:1.0:function:and” function. This is a Boolean  
 1322 function that takes one or more Boolean arguments (2 in this case) and performs the logical “AND”  
 1323 operation to compute the truth value of the expression.

1324 [g95] - [g112] The first part of the **condition** is evaluated (i.e. is the requestor the designated parent or  
 1325 guardian?). The function is “urn:oasis:names:tc:xacml:1.0:function:string-equal” and it takes two  
 1326 arguments of type “http://www.w3.org/2001/XMLSchema#string”.

1327 [g96] designates the first argument. Since “urn:oasis:names:tc:xacml:1.0:function:string-equal” takes  
 1328 arguments of type “http://www.w3.org/2001/XMLSchema#string”,  
 1329 “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that the **subject attribute**  
 1330 “urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id” in the request **context** contains  
 1331 exactly one value.

1332 [g98] designates the first argument. The value of the **subject attribute**  
 1333 “urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id” is selected from the request **context**  
 1334 using the <AttributeDesignator> element.

1335 [g104] As above, the “urn:oasis:names:tc:xacml:1.0:function:string-one-and-only” is used to ensure that  
 1336 the **bag** of values selected by its argument contains exactly one value of type  
 1337 “http://www.w3.org/2001/XMLSchema#string”.

1338 [g106] The second argument selects the value of the <md:parentGuardianId> element from the  
 1339 **resource** content using the <AttributeSelector> element. This element contains a free-form XPath  
 1340 expression, pointing into the <Content> element of the resource category. Note that all namespace  
 1341 prefixes in the XPath expression are resolved with standard namespace declarations. The  
 1342 AttributeSelector evaluates to the **bag** of values of type  
 1343 “http://www.w3.org/2001/XMLSchema#string”.

1344 [g113] references the <VariableDefinition> element, where the second part of the **condition** is  
 1345 defined.

### 1346 4.2.4.3 Rule 3

1347 **Rule** 3 illustrates the use of an **obligation** expression.

```

1348 [h1] <?xml version="1.0" encoding="UTF-8"?>
1349 [h2] <Policy
1350 [h3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1351 [h4]   xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1352 [h5]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

```

```

1353 [h6] xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
1354 http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
1355 [h7] xmlns:md="http://www.med.example.com/schemas/record.xsd"
1356 [h8] PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:3"
1357 [h9] Version="1.0"
1358 [h10] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1359 algorithm:deny-overrides">
1360 [h11] <Description>
1361 [h12] Policy for any medical record in the
1362 [h13] http://www.med.example.com/schemas/record.xsd namespace
1363 [h14] </Description>
1364 [h15] <PolicyDefaults>
1365 [h16] <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1366 [h17] </PolicyDefaults>
1367 [h18] <Target>
1368 [h19] <AnyOf>
1369 [h20] <AllOf>
1370 [h21] <Match
1371 [h22] MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1372 [h23] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1373 [h24] >urn:example:med:schemas:record</AttributeValue>
1374 [h25] <AttributeDesignator
1375 [h26] MustBePresent="false"
1376 [h27] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1377 [h28] AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1378 [h29] DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1379 [h30] </Match>
1380 [h31] </AllOf>
1381 [h32] </AnyOf>
1382 [h33] </Target>
1383 [h34] <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:3"
1384 [h35] Effect="Permit">
1385 [h36] <Description>
1386 [h37] A physician may write any medical element in a record
1387 [h38] for which he or she is the designated primary care
1388 [h39] physician, provided an email is sent to the patient
1389 [h40] </Description>
1390 [h41] <Target>
1391 [h42] <AnyOf>
1392 [h43] <AllOf>
1393 [h44] <Match
1394 [h45] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1395 [h46] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1396 [h47] >physician</AttributeValue>
1397 [h48] <AttributeDesignator
1398 [h49] MustBePresent="false"
1399 [h50] Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1400 [h51] AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
1401 [h52] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1402 [h53] </Match>
1403 [h54] </AllOf>
1404 [h55] </AnyOf>
1405 [h56] <AnyOf>
1406 [h57] <AllOf>
1407 [h58] <Match
1408 [h59] MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1409 [h60] <AttributeValue
1410 [h61] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1411 [h62] XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1412 [h63] >md:record/md:medical</AttributeValue>
1413 [h64] <AttributeDesignator
1414 [h65] MustBePresent="false"
1415 [h66] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1416 [h67] AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1417 [h68] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1418 [h69] </Match>
1419 [h70] </AllOf>
1420 [h71] </AnyOf>
1421 [h72] <AnyOf>
1422 [h73] <AllOf>
1423 [h74] <Match
1424 [h75] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1425 [h76] <AttributeValue

```

```

1426 [h77]         DataType="http://www.w3.org/2001/XMLSchema#string"
1427 [h78]         >write</AttributeValue>
1428 [h79]         <AttributeDesignator
1429 [h80]             MustBePresent="false"
1430 [h81]             Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1431 [h82]             AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1432 [h83]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1433 [h84]         </Match>
1434 [h85]     </AllOf>
1435 [h86] </AnyOf>
1436 [h87] </Target>
1437 [h88] <Condition>
1438 [h89]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1439 [h90]         <Apply
1440 [h91]             FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1441 [h92]             <AttributeDesignator
1442 [h93]                 MustBePresent="false"
1443 [h94]                 Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1444 [h95]                 AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id"
1445 [h96]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1446 [h97]             </Apply>
1447 [h98]         <Apply
1448 [h99]             FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
1449 [h100]            <AttributeSelector
1450 [h101]                MustBePresent="false"
1451 [h102]                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1452 [h103]                Path="md:record/md:primaryCarePhysician/md:registrationID/text()"
1453 [h104]                DataType="http://www.w3.org/2001/XMLSchema#string"/>
1454 [h105]            </Apply>
1455 [h106]        </Apply>
1456 [h107]    </Condition>
1457 [h108] </Rule>
1458 [h109] <ObligationExpressions>
1459 [h110]     <ObligationExpression
1460 [h111]         ObligationId="urn:oasis:names:tc:xacml:example:obligation:email"
1461 [h112]         FulfillOn="Permit">
1462 [h113]         <AttributeAssignmentExpression
1463 [h114]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:mailto">
1464 [h115]             <AttributeSelector
1465 [h116]                 MustBePresent="true"
1466 [h117]                 Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1467 [h118]                 Path="md:record/md:patient/md:patientContact/md:email"
1468 [h119]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1469 [h120]             </AttributeAssignmentExpression>
1470 [h121]         <AttributeAssignmentExpression
1471 [h122]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
1472 [h123]             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1473 [h124]                 >Your medical record has been accessed by:</AttributeValue>
1474 [h125]             </AttributeAssignmentExpression>
1475 [h126]         <AttributeAssignmentExpression
1476 [h127]             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
1477 [h128]             <AttributeDesignator
1478 [h129]                 MustBePresent="false"
1479 [h130]                 Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1480 [h131]                 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
1481 [h132]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1482 [h133]             </AttributeAssignmentExpression>
1483 [h134]         </ObligationExpression>
1484 [h135]     </ObligationExpressions>
1485 [h136] </Policy>

```

1486 [h2] - [h10] The <Policy> element includes standard namespace declarations as well as **policy** specific  
1487 parameters, such as PolicyId and RuleCombiningAlgId.

1488 [h8] **Policy** identifier. This parameter allows the **policy** to be referenced by a **policy set**.

1489 [h10] The **Rule-combining algorithm** identifies the algorithm for combining the outcomes of **rule**  
1490 evaluation.

1491 [h11] - [h14] Free-form description of the **policy**.

1492 [h18] - [h33] **Policy target**. The **policy target** defines a set of applicable **decision requests**. The  
1493 structure of the <Target> element in the <Policy> is identical to the structure of the <Target>

1494 element in the <Rule>. In this case, the **policy target** is the set of all XML **resources** that conform to  
1495 the namespace “urn:example:med:schemas:record”.

1496 [h34] - [h108] The only <Rule> element included in this <Policy>. Two parameters are specified in the  
1497 **rule** header: RuleId and Effect.

1498 [h41] - [h87] The **rule target** further constrains the **policy target**.

1499 [h44] - [h53] The <Match> element targets the **rule** at **subjects** whose  
1500 “urn:oasis:names:tc:xacml:3.0:example:attribute:role” **subject attribute** is equal to “physician”.

1501 [h58] - [h69] The <Match> element targets the **rule** at **resources** that match the XPath expression  
1502 “md:record/md:medical”.

1503 [h74] - [h84] The <Match> element targets the **rule** at **actions** whose  
1504 “urn:oasis:names:tc:xacml:1.0:action:action-id” **action attribute** is equal to “write”.

1505 [h88] - [h107] The <Condition> element. For the **rule** to be applicable to the **decision request**, the  
1506 **condition** must evaluate to “True”. This **condition** compares the value of the  
1507 “urn:oasis:names:tc:xacml:3.0:example:attribute:physician-id” **subject attribute** with the value of the  
1508 <registrationId> element in the medical record that is being accessed.

1509 [h109] - [h134] The <ObligationExpressions> element. **Obligations** are a set of operations that  
1510 must be performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be  
1511 associated with a “Permit” or “Deny” **authorization decision**. The element contains a single **obligation**  
1512 expression, which will be evaluated into an obligation when the policy is evaluated.

1513 [h110] - [h133] The <ObligationExpression> element consists of the ObligationId attribute, the  
1514 **authorization decision** value for which it must be fulfilled, and a set of **attribute** assignments.

1515 [h110] The ObligationId attribute identifies the **obligation**. In this case, the **PEP** is required to send  
1516 email.

1517 [h111] The FulfillOn attribute defines the **authorization decision** value for which the **obligation**  
1518 derived from the **obligation** expression must be fulfilled. In this case, the **obligation** must be fulfilled  
1519 when **access** is permitted.

1520 [h112] - [h119] The first parameter indicates where the **PEP** will find the email address in the **resource**.  
1521 The **PDP** will evaluate the <AttributeSelector> and return the result to the **PEP** inside the resulting  
1522 **obligation**.

1523 [h120] - [h123] The second parameter contains literal text for the email body.

1524 [h125] - [h132] The third parameter indicates where the **PEP** will find further text for the email body in the  
1525 **resource**. The **PDP** will evaluate the <AttributeDesignator> and return the result to the **PEP** inside  
1526 the resulting **obligation**.

#### 1527 4.2.4.4 Rule 4

1528 **Rule 4** illustrates the use of the “Deny” **Effect** value, and a <Rule> with no <Condition> element.

```

1529 [i1] <?xml version="1.0" encoding="UTF-8"?>
1530 [i2] <Policy
1531 [i3]   xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1532 [i4]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1533 [i5]   xmlns:md="http://www.med.example.com/schemas/record.xsd"
1534 [i6]   PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:4"
1535 [i7]   Version="1.0"
1536 [i8]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
1537 [i9]   algorithm:deny-overrides">
1538 [i9]   <PolicyDefaults>
1539 [i10]     <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
1540 [i11]   </PolicyDefaults>
1541 [i12]   <Target/>
1542 [i13]   <Rule
1543 [i14]     RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:4"
1544 [i15]     Effect="Deny">
1545 [i16]   <Description>
1546 [i17]     An Administrator shall not be permitted to read or write

```

```

1547 [i18] medical elements of a patient record in the
1548 [i19] http://www.med.example.com/records.xsd namespace.
1549 [i20] </Description>
1550 [i21] <Target>
1551 [i22] <AnyOf>
1552 [i23] <AllOf>
1553 [i24] <Match
1554 [i25] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1555 [i26] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1556 [i27] >administrator</AttributeValue>
1557 [i28] <AttributeDesignator
1558 [i29] MustBePresent="false"
1559 [i30] Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
1560 [i31] AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
1561 [i32] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1562 [i33] </Match>
1563 [i34] </AllOf>
1564 [i35] </AnyOf>
1565 [i36] <AnyOf>
1566 [i37] <AllOf>
1567 [i38] <Match
1568 [i39] MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
1569 [i40] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
1570 [i41] >urn:example:med:schemas:record</AttributeValue>
1571 [i42] <AttributeDesignator
1572 [i43] MustBePresent="false"
1573 [i44] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1574 [i45] AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1575 [i46] DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
1576 [i47] </Match>
1577 [i48] <Match
1578 [i49] MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
1579 [i50] <AttributeValue
1580 [i51] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
1581 [i52] XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1582 [i53] >md:record/md:medical</AttributeValue>
1583 [i54] <AttributeDesignator
1584 [i55] MustBePresent="false"
1585 [i56] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1586 [i57] AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
1587 [i58] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
1588 [i59] </Match>
1589 [i60] </AllOf>
1590 [i61] </AnyOf>
1591 [i62] <AnyOf>
1592 [i63] <AllOf>
1593 [i64] <Match
1594 [i65] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1595 [i66] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1596 [i67] >read</AttributeValue>
1597 [i68] <AttributeDesignator
1598 [i69] MustBePresent="false"
1599 [i70] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1600 [i71] AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1601 [i72] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1602 [i73] </Match>
1603 [i74] </AllOf>
1604 [i75] <AllOf>
1605 [i76] <Match
1606 [i77] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1607 [i78] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1608 [i79] >write</AttributeValue>
1609 [i80] <AttributeDesignator
1610 [i81] MustBePresent="false"
1611 [i82] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
1612 [i83] AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
1613 [i84] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1614 [i85] </Match>
1615 [i86] </AllOf>
1616 [i87] </AnyOf>
1617 [i88] </Target>
1618 [i89] </Rule>
1619 [i90] </Policy>

```

1620 [i13] - [i15] The <Rule> element declaration.

1621 [i15] **Rule Effect**. Every **rule** that evaluates to “True” emits the **rule effect** as its value. This **rule**  
 1622 **Effect** is “Deny” meaning that according to this **rule**, **access** must be denied when it evaluates to  
 1623 “True”.

1624 [i16] - [i20] Free form description of the **rule**.

1625 [i21] - [i88] **Rule target**. The **Rule target** defines the set of **decision requests** that are applicable to the  
 1626 **rule**.

1627 [i24] - [i33] The <Match> element targets the **rule** at **subjects** whose  
 1628 “urn:oasis:names:tc:xacml:3.0:example:attribute:role” **subject attribute** is equal to “administrator”.

1629 [i36] - [i61] The <AnyOf> element contains one <AllOf> element, which (in turn) contains two <Match>  
 1630 elements. The **target** matches if the **resource** identified by the request **context** matches both **resource**  
 1631 match criteria.

1632 [i38] - [i47] The first <Match> element targets the **rule** at **resources** whose  
 1633 “urn:oasis:names:tc:xacml:2.0:resource:target-namespace” **resource attribute** is equal to  
 1634 “urn:example:med:schemas:record”.

1635 [i48] - [i59] The second <Match> element targets the **rule** at XML elements that match the XPath  
 1636 expression “/md:record/md:medical”.

1637 [i62] - [i87] The <AnyOf> element contains two <AllOf> elements, each of which contains one <Match>  
 1638 element. The **target** matches if the **action** identified in the request **context** matches either of the **action**  
 1639 match criteria.

1640 [i64] - [i85] The <Match> elements **target** the **rule** at **actions** whose  
 1641 “urn:oasis:names:tc:xacml:1.0:action:action-id” **action attribute** is equal to “read” or “write”.

1642 This **rule** does not have a <Condition> element.

#### 1643 4.2.4.5 Example PolicySet

1644 This section uses the examples of the previous sections to illustrate the process of combining **policies**.  
 1645 The **policy** governing read **access** to medical elements of a record is formed from each of the four **rules**  
 1646 described in Section 4.2.3. In plain language, the combined **rule** is:

- 1647 • Either the requestor is the patient; or
- 1648 • the requestor is the parent or guardian and the patient is under 16; or
- 1649 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1650 • the requestor is not an administrator.

1651 The following **policy set** illustrates the combined **policies**. **Policy 3** is included by reference and **policy**  
 1652 **2** is explicitly included.

```

1653 [j1]    <?xml version="1.0" encoding="UTF-8"?>
1654 [j2]    <PolicySet
1655 [j3]      xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
1656 [j4]      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1657 [j5]      PolicySetId="urn:oasis:names:tc:xacml:3.0:example:policysetid:1"
1658 [j6]      Version="1.0"
1659 [j7]      PolicyCombiningAlgId=
1660 [j8]      "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
1661 [j9]      <Description>
1662 [j10]        Example policy set.
1663 [j11]      </Description>
1664 [j12]      <Target>
1665 [j13]        <AnyOf>
1666 [j14]          <AllOf>
1667 [j15]            <Match
1668 [j16]              MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1669 [j17]                <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
1670 [j18]                  >urn:example:med:schema:records</AttributeValue>
1671 [j19]                <AttributeDesignator
1672 [j20]                  MustBePresent="false"

```

```

1673 [j21]         Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
1674 [j22]         AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
1675 [j23]         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1676 [j24]         </Match>
1677 [j25]         </AllOf>
1678 [j26]         </AnyOf>
1679 [j27]         </Target>
1680 [j28]         <PolicyIdReference>
1681 [j29]           urn:oasis:names:tc:xacml:3.0:example:policyid:3
1682 [j30]         </PolicyIdReference>
1683 [j31]         <Policy
1684 [j32]           PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"
1685 [j33]           RuleCombiningAlgId=
1686 [j34]             "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
1687 [j35]           Version="1.0">
1688 [j36]           <Target/>
1689 [j37]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"
1690 [j38]             Effect="Permit">
1691 [j39]           </Rule>
1692 [j40]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
1693 [j41]             Effect="Permit">
1694 [j42]           </Rule>
1695 [j43]           <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:4"
1696 [j44]             Effect="Deny">
1697 [j45]           </Rule>
1698 [j46]         </Policy>
1699 [j47]         </PolicySet>

```

1700 [j2] - [j8] The <PolicySet> element declaration. Standard XML namespace declarations are included.

1701 [j5] The PolicySetId attribute is used for identifying this **policy set** for possible inclusion in another  
1702 **policy set**.

1703 [j7] - [j8] The **policy-combining algorithm** identifier. **Policies** and **policy sets** in this **policy set** are  
1704 combined according to the specified **policy-combining algorithm** when the **authorization decision** is  
1705 computed.

1706 [j9] - [j11] Free form description of the **policy set**.

1707 [j12] - [j27] The **policy set** <Target> element defines the set of **decision requests** that are applicable to  
1708 this <PolicySet> element.

1709 [j28] - [j30] PolicyIdReference includes a **policy** by id.

1710 [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>
```

```

1760 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>
1761 <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
1762 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="required"/>
1763 <xs:attribute name="MaxDelegationDepth" type="xs:integer" use="optional"/>
1764 </xs:complexType>

```

1765 The <PolicySet> element is of PolicySetType complex type.

1766 The <PolicySet> element contains the following attributes and elements:

1767 PolicySetId [Required]

1768 **Policy set** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to  
1769 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or URI  
1770 scheme. If the **policy set** identifier is in the form of a URL, then it MAY be resolvable.

1771 Version [Required]

1772 The version number of the PolicySet.

1773 PolicyCombiningAlgId [Required]

1774 The identifier of the **policy-combining algorithm** by which the <PolicySet>,  
1775 <CombinerParameters>, <PolicyCombinerParameters> and  
1776 <PolicySetCombinerParameters> components MUST be combined. Standard **policy-**  
1777 **combining algorithms** are listed in Appendix C. Standard **policy-combining algorithm**  
1778 identifiers are listed in Section B.9.

1779 MaxDelegationDepth [Optional]

1780 If present, limits the depth of delegation which is authorized by this **policy set**. See the delegation  
1781 profile [**XACMLAdmin**].

1782 <Description> [Optional]

1783 A free-form description of the **policy set**.

1784 <PolicyIssuer> [Optional]

1785 **Attributes** of the **issuer** of the **policy set**.

1786 <PolicySetDefaults> [Optional]

1787 A set of default values applicable to the **policy set**. The scope of the <PolicySetDefaults>  
1788 element SHALL be the enclosing **policy set**.

1789 <Target> [Required]

1790 The <Target> element defines the applicability of a **policy set** to a set of **decision requests**.

1791 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be computed  
1792 from the <Target> elements of the referenced <Policy> elements, either as an intersection or  
1793 as a union.

1794 <PolicySet> [Any Number]

1795 A **policy set** that is included in this **policy set**.

1796 <Policy> [Any Number]

1797 A **policy** that is included in this **policy set**.

1798 <PolicySetIdReference> [Any Number]

1799 A reference to a **policy set** that MUST be included in this **policy set**. If  
1800 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1801 <PolicyIdReference> [Any Number]

1802 A reference to a **policy** that MUST be included in this **policy set**. If the  
1803 <PolicyIdReference> is a URL, then it MAY be resolvable.

1804 <ObligationExpressions> [Optional]  
 1805       Contains the set of <ObligationExpression> elements. See Section 7.16 for a description of  
 1806       how the set of **obligations** to be returned by the **PDP** shall be determined.

1807 <AdviceExpressions> [Optional]  
 1808       Contains the set of <AdviceExpression> elements. See Section 7.16 for a description of how  
 1809       the set of **advice** to be returned by the **PDP** shall be determined.

1810 <CombinerParameters> [Optional]  
 1811       Contains a sequence of <CombinerParameter> elements.

1812 <PolicyCombinerParameters> [Optional]  
 1813       Contains a sequence of <CombinerParameter> elements that are associated with a particular  
 1814       <Policy> or <PolicyIdReference> element within the <PolicySet>.

1815 <PolicySetCombinerParameters> [Optional]  
 1816       Contains a sequence of <CombinerParameter> elements that are associated with a particular  
 1817       <PolicySet> or <PolicySetIdReference> element within the <PolicySet>.

## 1818 5.2 Element <Description>

1819 The <Description> element contains a free-form description of the <PolicySet>, <Policy>,  
 1820 <Rule> or <Apply> element. The <Description> element is of xs:string simple type.

1821 

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

## 1822 5.3 Element <PolicyIssuer>

1823 The <PolicyIssuer> element contains **attributes** describing the issuer of the **policy** or **policy set**.  
 1824 The use of the **policy** issuer element is defined in a separate administration profile [**XACMLAdmin**]. A  
 1825 PDP which does not implement the administration profile **MUST** report an error or return an Indeterminate  
 1826 result if it encounters this element.

1827 

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

1828 

```
<xs:complexType name="PolicyIssuerType">
```

1829 

```
  <xs:sequence>
```

1830 

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

1831 

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

1832 

```
  </xs:sequence>
```

1833 

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

1834 The <PolicyIssuer> element is of PolicyIssuerType complex type.

1835 The <PolicyIssuer> element contains the following elements:

1836 <Content> [Optional]

1837       Free form XML describing the issuer. See Section 5.45.

1838 <Attribute> [Zero to many]

1839       An **attribute** of the issuer. See Section 5.46.

## 1840 5.4 Element <PolicySetDefaults>

1841 The <PolicySetDefaults> element SHALL specify default values that apply to the <PolicySet>  
 1842 element.

1843 

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

1844 

```
<xs:complexType name="DefaultsType">
```

1845 

```
  <xs:sequence>
```

1846  
1847  
1848  
1849  
1850

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

1851 <PolicySetDefaults> element is of DefaultsType complex type.

1852 The <PolicySetDefaults> element contains the following elements:

1853 <XPathVersion> [Optional]

1854 Default XPath version.

## 1855 5.5 Element <XPathVersion>

1856 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by  
1857 <AttributeSelector> elements and XPath-based functions in the **policy set** or **policy**.

1858

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

1859 The URI for the XPath 1.0 specification is "http://www.w3.org/TR/1999/REC-xpath-19991116".

1860 The URI for the XPath 2.0 specification is "http://www.w3.org/TR/2007/REC-xpath20-20070123".

1861 The <XPathVersion> element is REQUIRED if the XACML enclosing **policy set** or **policy** contains  
1862 <AttributeSelector> elements or XPath-based functions.

## 1863 5.6 Element <Target>

1864 The <Target> element identifies the set of **decision requests** that the parent element is intended to  
1865 evaluate. The <Target> element SHALL appear as a child of a <PolicySet> and <Policy> element  
1866 and MAY appear as a child of a <Rule> element.

1867 The <Target> element SHALL contain a **conjunctive sequence** of <AnyOf> elements. For the parent  
1868 of the <Target> element to be applicable to the **decision request**, there MUST be at least one positive  
1869 match between each <AnyOf> element of the <Target> element and the corresponding section of the  
1870 <Request> element.

1871  
1872  
1873  
1874  
1875  
1876

```
<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>
```

1877 The <Target> element is of TargetType complex type.

1878 The <Target> element contains the following elements:

1879 <AnyOf> [Zero to Many]

1880 Matching specification for **attributes** in the **context**. If this element is missing, then the **target**  
1881 SHALL match all **contexts**.

## 1882 5.7 Element <AnyOf>

1883 The <AnyOf> element SHALL contain a **disjunctive sequence** of <AllOf> elements.

1884  
1885  
1886  
1887  
1888  
1889

```
<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>
```

1890 The <AnyOf> element is of AnyOfType complex type.

1891 The <AnyOf> element contains the following elements:

1892 <AllOf> [One to Many, Required]

1893 See Section 5.8.

## 1894 5.8 Element <AllOf>

1895 The <AllOf> element SHALL contain a **conjunctive sequence** of <Match> elements.

```
1896 <xs:element name="AllOf" type="xacml:AllOfType"/>
1897 <xs:complexType name="AllOfType">
1898   <xs:sequence minOccurs="1" maxOccurs="unbounded">
1899     <xs:element ref="xacml:Match"/>
1900   </xs:sequence>
1901 </xs:complexType>
```

1902 The <AllOf> element is of AllOfType complex type.

1903 The <AllOf> element contains the following elements:

1904 <Match> [One to Many]

1905 A **conjunctive sequence** of individual matches of the **attributes** in the request **context** and the  
1906 embedded **attribute** values. See Section 5.9.

## 1907 5.9 Element <Match>

1908 The <Match> element SHALL identify a set of entities by matching **attribute** values in an  
1909 <Attributes> element of the request **context** with the embedded **attribute** value.

```
1910 <xs:element name="Match" type="xacml:MatchType"/>
1911 <xs:complexType name="MatchType">
1912   <xs:sequence>
1913     <xs:element ref="xacml:AttributeValue"/>
1914     <xs:choice>
1915       <xs:element ref="xacml:AttributeDesignator"/>
1916       <xs:element ref="xacml:AttributeSelector"/>
1917     </xs:choice>
1918   </xs:sequence>
1919   <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
1920 </xs:complexType>
```

1921 The <Match> element is of MatchType complex type.

1922 The <Match> element contains the following attributes and elements:

1923 MatchId [Required]

1924 Specifies a matching function. The value of this attribute MUST be of type xs:anyURI with legal  
1925 values documented in Section 7.6.

1926 <AttributeValue> [Required]

1927 Embedded **attribute** value.

1928 <AttributeDesignator> [Required choice]

1929 MAY be used to identify one or more **attribute** values in an <Attributes> element of the  
1930 request **context**.

1931 <AttributeSelector> [Required choice]

1932 MAY be used to identify one or more **attribute** values in a <Content> element of the request  
1933 **context**.

## 1934 5.10 Element <PolicySetIdReference>

1935 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element by id.  
1936 If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet> element.  
1937 However, the mechanism for resolving a **policy set** reference to the corresponding **policy set** is outside  
1938 the scope of this specification.

```
1939 <xs:element name="PolicySetIdReference" type="xacml:IdReferenceType"/>
1940 <xs:complexType name="IdReferenceType">
1941   <xs:simpleContent>
1942     <xs:extension base="xs:anyURI">
1943       <xs:attribute name="xacml:Version"
1944         type="xacml:VersionMatchType" use="optional"/>
1945       <xs:attribute name="xacml:EarliestVersion"
1946         type="xacml:VersionMatchType" use="optional"/>
1947       <xs:attribute name="xacml:LatestVersion"
1948         type="xacml:VersionMatchType" use="optional"/>
1949     </xs:extension>
1950   </xs:simpleContent>
1951 </xs:complexType>
```

1952 Element <PolicySetIdReference> is of xacml:IdReferenceType complex type.

1953 IdReferenceType extends the xs:anyURI type with the following attributes:

1954 Version [Optional]

1955 Specifies a matching expression for the version of the **policy set** referenced.

1956 EarliestVersion [Optional]

1957 Specifies a matching expression for the earliest acceptable version of the **policy set** referenced.

1958 LatestVersion [Optional]

1959 Specifies a matching expression for the latest acceptable version of the **policy set** referenced.

1960 The matching operation is defined in Section 5.13. Any combination of these attributes MAY be present  
1961 in a <PolicySetIdReference>. The referenced **policy set** MUST match all expressions. If none of  
1962 these attributes is present, then any version of the **policy set** is acceptable. In the case that more than  
1963 one matching version can be obtained, then the most recent one SHOULD be used.

## 1964 5.11 Element <PolicyIdReference>

1965 The <PolicyIdReference> element SHALL be used to reference a <Policy> element by id. If  
1966 <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy> element. However, the  
1967 mechanism for resolving a **policy** reference to the corresponding **policy** is outside the scope of this  
1968 specification.

```
1969 <xs:element name="PolicyIdReference" type="xacml:IdReferenceType"/>
```

1970 Element <PolicyIdReference> is of xacml:IdReferenceType complex type (see Section 5.10) .

## 1971 5.12 Simple type VersionType

1972 Elements of this type SHALL contain the version number of the **policy** or **policy set**.

```
1973 <xs:simpleType name="VersionType">
1974   <xs:restriction base="xs:string">
1975     <xs:pattern value="(\d+\.)*\d+"/>
1976   </xs:restriction>
1977 </xs:simpleType>
```

1978 The version number is expressed as a sequence of decimal numbers, each separated by a period (.).  
1979 'd+' represents a sequence of one or more decimal digits.

## 1980 5.13 Simple type VersionMatchType

1981 Elements of this type SHALL contain a restricted regular expression matching a version number (see  
1982 Section 5.12). The expression SHALL match versions of a referenced **policy** or **policy set** that are  
1983 acceptable for inclusion in the referencing **policy** or **policy set**.

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

1989 A version match is '.'-separated, like a version string. A number represents a direct numeric match. A '\*'  
1990 means that any single number is valid. A '+' means that any number, and any subsequent numbers, are  
1991 valid. In this manner, the following four patterns would all match the version string '1.2.3': '1.2.3', '1.\*.3',  
1992 '1.2.\*' and '1.+'

## 1993 5.14 Element <Policy>

1994 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

1995 A <Policy> element may be evaluated, in which case the evaluation procedure defined in Section 7.11  
1996 SHALL be used.

1997 The main components of this element are the <Target>, <Rule>, <CombinerParameters>,  
1998 <RuleCombinerParameters>, <ObligationExpressions> and <AdviceExpressions>  
1999 elements and the RuleCombiningAlgId attribute.

2000 A <Policy> element MAY contain a <PolicyIssuer> element. The interpretation of the  
2001 <PolicyIssuer> element is explained in the separate administrative **policy** profile [XACMLAdmin].

2002 The <Target> element defines the applicability of the <Policy> element to a set of **decision requests**.  
2003 If the <Target> element within the <Policy> element matches the request **context**, then the  
2004 <Policy> element MAY be used by the **PDP** in making its **authorization decision**. See Section 7.11.

2005 The <Policy> element includes a sequence of choices between <VariableDefinition> and  
2006 <Rule> elements.

2007 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the  
2008 RuleCombiningAlgId attribute.

2009 The <ObligationExpressions> element contains a set of **obligation** expressions that MUST be  
2010 evaluated into **obligations** by the **PDP** and the resulting **obligations** MUST be fulfilled by the **PEP** in  
2011 conjunction with the **authorization decision**. If the **PEP** does not understand, or cannot fulfill, any of the  
2012 **obligations**, then it MUST act according to the PEP bias. See Section 7.2 and 7.16.

2013 The <AdviceExpressions> element contains a set of **advice** expressions that MUST be evaluated into  
2014 **advice** by the **PDP**. The resulting **advice** MAY be safely ignored by the **PEP** in conjunction with the  
2015 **authorization decision**. See Section 7.16.

```
2016 <xs:element name="Policy" type="xacml:PolicyType"/>  
2017 <xs:complexType name="PolicyType">  
2018   <xs:sequence>  
2019     <xs:element ref="xacml:Description" minOccurs="0"/>  
2020     <xs:element ref="xacml:PolicyIssuer" minOccurs="0"/>  
2021     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>  
2022     <xs:element ref="xacml:Target"/>  
2023     <xs:choice maxOccurs="unbounded">  
2024       <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>  
2025       <xs:element ref="xacml:RuleCombinerParameters" minOccurs="0"/>  
2026       <xs:element ref="xacml:VariableDefinition"/>  
2027       <xs:element ref="xacml:Rule"/>  
2028     </xs:choice>  
2029     <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
```

```

2030     <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
2031 </xs:sequence>
2032 <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2033 <xs:attribute name="Version" type="xacml:VersionType" use="required"/>
2034 <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2035 <xs:attribute name="MaxDelegationDepth" type="xs:integer" use="optional"/>
2036 </xs:complexType>

```

2037 The <Policy> element is of PolicyType complex type.

2038 The <Policy> element contains the following attributes and elements:

2039 PolicyId [Required]

2040 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to the  
2041 **PDP** have the same identifier. This MAY be achieved by following a predefined URN or URI  
2042 scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2043 Version [Required]

2044 The version number of the **Policy**.

2045 RuleCombiningAlgId [Required]

2046 The identifier of the **rule-combining algorithm** by which the <Policy>,  
2047 <CombinerParameters> and <RuleCombinerParameters> components MUST be  
2048 combined. Standard **rule-combining algorithms** are listed in Appendix C. Standard **rule-**  
2049 **combining algorithm** identifiers are listed in Section B.9.

2050 MaxDelegationDepth [Optional]

2051 If present, limits the depth of delegation which is authorized by this **policy**. See the delegation  
2052 profile [XACMLAdmin].

2053 <Description> [Optional]

2054 A free-form description of the **policy**. See Section 5.2.

2055 <PolicyIssuer> [Optional]

2056 **Attributes** of the **issuer** of the **policy**.

2057 <PolicyDefaults> [Optional]

2058 Defines a set of default values applicable to the **policy**. The scope of the <PolicyDefaults>  
2059 element SHALL be the enclosing **policy**.

2060 <CombinerParameters> [Optional]

2061 A sequence of parameters to be used by the **rule-combining algorithm**.

2062 <RuleCombinerParameters> [Optional]

2063 A sequence of parameters to be used by the **rule-combining algorithm**.

2064 <Target> [Required]

2065 The <Target> element defines the applicability of a <Policy> to a set of **decision requests**.

2066 The <Target> element MAY be declared by the creator of the <Policy> element, or it MAY be  
2067 computed from the <Target> elements of the referenced <Rule> elements either as an  
2068 intersection or as a union.

2069 <VariableDefinition> [Any Number]

2070 Common function definitions that can be referenced from anywhere in a **rule** where an  
2071 expression can be found.

2072 <Rule> [Any Number]

2073 A sequence of **rules** that MUST be combined according to the `RuleCombiningAlgId` attribute.  
2074 **Rules** whose `<Target>` elements and conditions match the **decision request** MUST be  
2075 considered. **Rules** whose `<Target>` elements or conditions do not match the **decision request**  
2076 SHALL be ignored.

2077 `<ObligationExpressions>` [Optional]

2078 A **conjunctive sequence** of **obligation** expressions which MUST be evaluated into **obligations**  
2079 by the PDP. The corresponding **obligations** MUST be fulfilled by the **PEP** in conjunction with  
2080 the **authorization decision**. See Section 7.16 for a description of how the set of **obligations** to  
2081 be returned by the **PDP** SHALL be determined. See section 7.2 about enforcement of  
2082 **obligations**.

2083 `<AdviceExpressions>` [Optional]

2084 A **conjunctive sequence** of **advice** expressions which MUST be evaluated into **advice** by the **PDP**.  
2085 The corresponding **advice** provide supplementary information to the **PEP** in conjunction with the  
2086 **authorization decision**. See Section 7.16 for a description of how the set of **advice** to be  
2087 returned by the **PDP** SHALL be determined.

## 2088 5.15 Element `<PolicyDefaults>`

2089 The `<PolicyDefaults>` element SHALL specify default values that apply to the `<Policy>` element.

```
2090 <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>  
2091 <xs:complexType name="DefaultsType">  
2092 <xs:sequence>  
2093 <xs:choice>  
2094 <xs:element ref="xacml:XPathVersion" />  
2095 </xs:choice>  
2096 </xs:sequence>  
2097 </xs:complexType>
```

2098 `<PolicyDefaults>` element is of `DefaultsType` complex type.

2099 The `<PolicyDefaults>` element contains the following elements:

2100 `<XPathVersion>` [Optional]

2101 Default XPath version.

## 2102 5.16 Element `<CombinerParameters>`

2103 The `<CombinerParameters>` element conveys parameters for a **policy-** or **rule-combining algorithm**.

2104 If multiple `<CombinerParameters>` elements occur within the same **policy** or **policy set**, they SHALL  
2105 be considered equal to one `<CombinerParameters>` element containing the concatenation of all the  
2106 sequences of `<CombinerParameters>` contained in all the aforementioned `<CombinerParameters>`  
2107 elements, such that the order of occurrence of the `<CombinerParameters>` elements is preserved in the  
2108 concatenation of the `<CombinerParameter>` elements.

2109 Note that none of the combining algorithms specified in XACML 3.0 is parameterized.

```
2110 <xs:element name="CombinerParameters" type="xacml:CombinerParametersType"/>  
2111 <xs:complexType name="CombinerParametersType">  
2112 <xs:sequence>  
2113 <xs:element ref="xacml:CombinerParameter" minOccurs="0"  
2114 maxOccurs="unbounded"/>  
2115 </xs:sequence>  
2116 </xs:complexType>
```

2117 The `<CombinerParameters>` element is of `CombinerParametersType` complex type.

2118 The `<CombinerParameters>` element contains the following elements:

2119 <CombinerParameter> [Any Number]  
2120       A single parameter. See Section 5.17.  
2121 Support for the <CombinerParameters> element is optional.

## 2122 5.17 Element <CombinerParameter>

2123 The <CombinerParameter> element conveys a single parameter for a *policy*- or *rule-combining*  
2124 *algorithm*.

```
2125 <xs:element name="CombinerParameter" type="xacml:CombinerParameterType"/>  
2126 <xs:complexType name="CombinerParameterType">  
2127   <xs:sequence>  
2128     <xs:element ref="xacml:AttributeValue"/>  
2129   </xs:sequence>  
2130   <xs:attribute name="ParameterName" type="xs:string" use="required"/>  
2131 </xs:complexType>
```

2132 The <CombinerParameter> element is of CombinerParameterType complex type.

2133 The <CombinerParameter> element contains the following attributes:

2134 ParameterName [Required]

2135       The identifier of the parameter.

2136 <AttributeValue> [Required]

2137       The value of the parameter.

2138 Support for the <CombinerParameter> element is optional.

## 2139 5.18 Element <RuleCombinerParameters>

2140 The <RuleCombinerParameters> element conveys parameters associated with a particular *rule*  
2141 within a *policy* for a *rule-combining algorithm*.

2142 Each <RuleCombinerParameters> element MUST be associated with a *rule* contained within the  
2143 same *policy*. If multiple <RuleCombinerParameters> elements reference the same *rule*, they SHALL  
2144 be considered equal to one <RuleCombinerParameters> element containing the concatenation of all  
2145 the sequences of <CombinerParameters> contained in all the aforementioned  
2146 <RuleCombinerParameters> elements, such that the order of occurrence of the  
2147 <RuleCominberParameters> elements is preserved in the concatenation of the  
2148 <CombinerParameter> elements.

2149 Note that none of the *rule-combining algorithms* specified in XACML 3.0 is parameterized.

```
2150 <xs:element name="RuleCombinerParameters"  
2151   type="xacml:RuleCombinerParametersType"/>  
2152 <xs:complexType name="RuleCombinerParametersType">  
2153   <xs:complexContent>  
2154     <xs:extension base="xacml:CombinerParametersType">  
2155       <xs:attribute name="RuleIdRef" type="xs:string"  
2156         use="required"/>  
2157     </xs:extension>  
2158   </xs:complexContent>  
2159 </xs:complexType>
```

2160 The <RuleCombinerParameters> element contains the following attribute:

2161 RuleIdRef [Required]

2162       The identifier of the <Rule> contained in the *policy*.

2163 Support for the <RuleCombinerParameters> element is optional, only if support for combiner  
2164 parameters is not implemented.

## 2165 5.19 Element <PolicyCombinerParameters>

2166 The <PolicyCombinerParameters> element conveys parameters associated with a particular *policy*  
2167 within a *policy set* for a *policy-combining algorithm*.

2168 Each <PolicyCombinerParameters> element MUST be associated with a *policy* contained within the  
2169 same *policy set*. If multiple <PolicyCombinerParameters> elements reference the same *policy*,  
2170 they SHALL be considered equal to one <PolicyCombinerParameters> element containing the  
2171 concatenation of all the sequences of <CombinerParameters> contained in all the aforementioned  
2172 <PolicyCombinerParameters> elements, such that the order of occurrence of the  
2173 <PolicyCominberParameters> elements is preserved in the concatenation of the  
2174 <CombinerParameter> elements.

2175 Note that none of the *policy-combining algorithms* specified in XACML 3.0 is parameterized.

```
2176 <xs:element name="PolicyCombinerParameters"  
2177 type="xacml:PolicyCombinerParametersType"/>  
2178 <xs:complexType name="PolicyCombinerParametersType">  
2179   <xs:complexContent>  
2180     <xs:extension base="xacml:CombinerParametersType">  
2181       <xs:attribute name="PolicyIdRef" type="xs:anyURI"  
2182 use="required"/>  
2183     </xs:extension>  
2184   </xs:complexContent>  
2185 </xs:complexType>
```

2186 The <PolicyCombinerParameters> element is of PolicyCombinerParametersType complex  
2187 type.

2188 The <PolicyCombinerParameters> element contains the following attribute:

2189 PolicyIdRef [Required]

2190 The identifier of a <Policy> or the value of a <PolicyIdReference> contained in the *policy*  
2191 *set*.

2192 Support for the <PolicyCombinerParameters> element is optional, only if support for combiner  
2193 parameters is not implemented.

## 2194 5.20 Element <PolicySetCombinerParameters>

2195 The <PolicySetCombinerParameters> element conveys parameters associated with a particular  
2196 *policy set* within a *policy set* for a *policy-combining algorithm*.

2197 Each <PolicySetCombinerParameters> element MUST be associated with a *policy set* contained  
2198 within the same *policy set*. If multiple <PolicySetCombinerParameters> elements reference the  
2199 same *policy set*, they SHALL be considered equal to one <PolicySetCombinerParameters>  
2200 element containing the concatenation of all the sequences of <CombinerParameters> contained in all  
2201 the aforementioned <PolicySetCombinerParameters> elements, such that the order of occurrence  
2202 of the <PolicySetCominberParameters> elements is preserved in the concatenation of the  
2203 <CombinerParameter> elements.

2204 Note that none of the *policy-combining algorithms* specified in XACML 3.0 is parameterized.

```
2205 <xs:element name="PolicySetCombinerParameters"  
2206 type="xacml:PolicySetCombinerParametersType"/>  
2207 <xs:complexType name="PolicySetCombinerParametersType">  
2208   <xs:complexContent>  
2209     <xs:extension base="xacml:CombinerParametersType">  
2210       <xs:attribute name="PolicySetIdRef" type="xs:anyURI"  
2211 use="required"/>  
2212     </xs:extension>  
2213   </xs:complexContent>  
2214 </xs:complexType>
```

2215 The <PolicySetCombinerParameters> element is of PolicySetCombinerParametersType  
 2216 complex type.

2217 The <PolicySetCombinerParameters> element contains the following attribute:

2218 PolicySetIdRef [Required]

2219       The identifier of a <PolicySet> or the value of a <PolicySetIdReference> contained in the  
 2220       *policy set*.

2221 Support for the <PolicySetCombinerParameters> element is optional, only if support for combiner  
 2222 parameters is not implemented.

## 2223 5.21 Element <Rule>

2224 The <Rule> element SHALL define the individual *rules* in the *policy*. The main components of this  
 2225 element are the <Target>, <Condition>, <ObligationExpressions> and  
 2226 <AdviceExpressions> elements and the Effect attribute.

2227 A <Rule> element may be evaluated, in which case the evaluation procedure defined in Section 7.10  
 2228 SHALL be used.

```

2229 <xs:element name="Rule" type="xacml:RuleType"/>
2230 <xs:complexType name="RuleType">
2231   <xs:sequence>
2232     <xs:element ref="xacml:Description" minOccurs="0"/>
2233     <xs:element ref="xacml:Target" minOccurs="0"/>
2234     <xs:element ref="xacml:Condition" minOccurs="0"/>
2235     <xs:element ref="xacml:ObligationExpressions" minOccurs="0"/>
2236     <xs:element ref="xacml:AdviceExpressions" minOccurs="0"/>
2237   </xs:sequence>
2238   <xs:attribute name="RuleId" type="xs:string" use="required"/>
2239   <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>
2240 </xs:complexType>
  
```

2241 The <Rule> element is of RuleType complex type.

2242 The <Rule> element contains the following attributes and elements:

2243 RuleId [Required]

2244       A string identifying this *rule*.

2245 Effect [Required]

2246       **Rule effect.** The value of this attribute is either “Permit” or “Deny”.

2247 <Description> [Optional]

2248       A free-form description of the *rule*.

2249 <Target> [Optional]

2250       Identifies the set of *decision requests* that the <Rule> element is intended to evaluate. If this  
 2251       element is omitted, then the *target* for the <Rule> SHALL be defined by the <Target> element  
 2252       of the enclosing <Policy> element. See Section 7.7 for details.

2253 <Condition> [Optional]

2254       A *predicate* that MUST be satisfied for the *rule* to be assigned its Effect value.

2255 <ObligationExpressions> [Optional]

2256       A *conjunctive sequence* of *obligation* expressions which MUST be evaluated into *obligations*  
 2257       by the PDP. The corresponding *obligations* MUST be fulfilled by the *PEP* in conjunction with  
 2258       the *authorization decision*. See Section 7.16 for a description of how the set of *obligations* to  
 2259       be returned by the *PDP* SHALL be determined. See section 7.2 about enforcement of  
 2260       *obligations*.

2261 <AdviceExpressions> [Optional]  
2262 A **conjunctive sequence** of **advice** expressions which MUST be evaluated into **advice** by the **PDP**.  
2263 The corresponding **advice** provide supplementary information to the **PEP** in conjunction with the  
2264 **authorization decision**. See Section 7.16 for a description of how the set of **advice** to be  
2265 returned by the **PDP** SHALL be determined.

## 2266 5.22 Simple type EffectType

2267 The EffectType simple type defines the values allowed for the Effect attribute of the <Rule> element  
2268 and for the FulfillOn attribute of the <ObligationExpression> and <AdviceExpression>  
2269 elements.

```
2270 <xs:simpleType name="EffectType">  
2271   <xs:restriction base="xs:string">  
2272     <xs:enumeration value="Permit"/>  
2273     <xs:enumeration value="Deny"/>  
2274   </xs:restriction>  
2275 </xs:simpleType>
```

## 2276 5.23 Element <VariableDefinition>

2277 The <VariableDefinition> element SHALL be used to define a value that can be referenced by a  
2278 <VariableReference> element. The name supplied for its VariableId attribute SHALL NOT occur  
2279 in the VariableId attribute of any other <VariableDefinition> element within the encompassing  
2280 **policy**. The <VariableDefinition> element MAY contain undefined <VariableReference>  
2281 elements, but if it does, a corresponding <VariableDefinition> element MUST be defined later in  
2282 the encompassing **policy**. <VariableDefinition> elements MAY be grouped together or MAY be  
2283 placed close to the reference in the encompassing **policy**. There MAY be zero or more references to  
2284 each <VariableDefinition> element.

```
2285 <xs:element name="VariableDefinition" type="xacml:VariableDefinitionType"/>  
2286 <xs:complexType name="VariableDefinitionType">  
2287   <xs:sequence>  
2288     <xs:element ref="xacml:Expression"/>  
2289   </xs:sequence>  
2290   <xs:attribute name="VariableId" type="xs:string" use="required"/>  
2291 </xs:complexType>
```

2292 The <VariableDefinition> element is of VariableDefinitionType complex type. The  
2293 <VariableDefinition> element has the following elements and attributes:

2294 <Expression> [Required]  
2295 Any element of ExpressionType complex type.  
2296 VariableId [Required]  
2297 The name of the variable definition.

## 2298 5.24 Element <VariableReference>

2299 The <VariableReference> element is used to reference a value defined within the same  
2300 encompassing <Policy> element. The <VariableReference> element SHALL refer to the  
2301 <VariableDefinition> element by string equality on the value of their respective VariableId  
2302 attributes. One and only one <VariableDefinition> MUST exist within the same encompassing  
2303 <Policy> element to which the <VariableReference> refers. There MAY be zero or more  
2304 <VariableReference> elements that refer to the same <VariableDefinition> element.

```
2305 <xs:element name="VariableReference" type="xacml:VariableReferenceType"  
2306 substitutionGroup="xacml:Expression"/>  
2307 <xs:complexType name="VariableReferenceType">
```

```

2308     <xs:complexContent>
2309         <xs:extension base="xacml:ExpressionType">
2310             <xs:attribute name="VariableId" type="xs:string"
2311                 use="required"/>
2312         </xs:extension>
2313     </xs:complexContent>
2314 </xs:complexType>

```

2315 The <VariableReference> element is of the VariableReferenceType complex type, which is of  
2316 the ExpressionType complex type and is a member of the <Expression> element substitution group.  
2317 The <VariableReference> element MAY appear any place where an <Expression> element occurs  
2318 in the schema.

2319 The <VariableReference> element has the following attribute:

2320 VariableId [Required]

2321 The name used to refer to the value defined in a <VariableDefinition> element.

## 2322 5.25 Element <Expression>

2323 The <Expression> element is not used directly in a **policy**. The <Expression> element signifies that  
2324 an element that extends the ExpressionType and is a member of the <Expression> element  
2325 substitution group SHALL appear in its place.

```

2326 <xs:element name="Expression" type="xacml:ExpressionType" abstract="true"/>
2327 <xs:complexType name="ExpressionType" abstract="true"/>

```

2328 The following elements are in the <Expression> element substitution group:

2329 <Apply>, <AttributeSelector>, <AttributeValue>, <Function>, <VariableReference> and  
2330 <AttributeDesignator>.

## 2331 5.26 Element <Condition>

2332 The <Condition> element is a Boolean function over **attributes** or functions of **attributes**.

```

2333 <xs:element name="Condition" type="xacml:ConditionType"/>
2334 <xs:complexType name="ConditionType">
2335     <xs:sequence>
2336         <xs:element ref="xacml:Expression"/>
2337     </xs:sequence>
2338 </xs:complexType>

```

2339 The <Condition> contains one <Expression> element, with the restriction that the <Expression>  
2340 return data-type MUST be "http://www.w3.org/2001/XMLSchema#boolean". Evaluation of the  
2341 <Condition> element is described in Section 7.9.

## 2342 5.27 Element <Apply>

2343 The <Apply> element denotes application of a function to its arguments, thus encoding a function call.  
2344 The <Apply> element can be applied to any combination of the members of the <Expression>  
2345 element substitution group. See Section 5.25.

```

2346 <xs:element name="Apply" type="xacml:ApplyType"
2347     substitutionGroup="xacml:Expression"/>
2348 <xs:complexType name="ApplyType">
2349     <xs:complexContent>
2350         <xs:extension base="xacml:ExpressionType">
2351             <xs:sequence>
2352                 <xs:element ref="xacml:Description" minOccurs="0"/>
2353                 <xs:element ref="xacml:Expression" minOccurs="0"
2354                     maxOccurs="unbounded"/>

```

```

2355         </xs:sequence>
2356         <xs:attribute name="FunctionId" type="xs:anyURI"
2357             use="required"/>
2358     </xs:extension>
2359 </xs:complexContent>
2360 </xs:complexType>

```

2361 The <Apply> element is of ApplyType complex type.

2362 The <Apply> element contains the following attributes and elements:

2363 FunctionId [Required]

2364 The identifier of the function to be applied to the arguments. XACML-defined functions are  
2365 described in Appendix A.3.

2366 <Description> [Optional]

2367 A free-form description of the <Apply> element.

2368 <Expression> [Optional]

2369 Arguments to the function, which may include other functions.

## 2370 5.28 Element <Function>

2371 The <Function> element SHALL be used to name a function as an argument to the function defined by  
2372 the parent <Apply> element.

```

2373 <xs:element name="Function" type="xacml:FunctionType"
2374     substitutionGroup="xacml:Expression"/>
2375 <xs:complexType name="FunctionType">
2376     <xs:complexContent>
2377         <xs:extension base="xacml:ExpressionType">
2378             <xs:attribute name="FunctionId" type="xs:anyURI"
2379                 use="required"/>
2380         </xs:extension>
2381     </xs:complexContent>
2382 </xs:complexType>

```

2383 The <Function> element is of FunctionType complex type.

2384 The <Function> element contains the following attribute:

2385 FunctionId [Required]

2386 The identifier of the function.

## 2387 5.29 Element <AttributeDesignator>

2388 The <AttributeDesignator> element retrieves a **bag** of values for a **named attribute** from the  
2389 request **context**. A **named attribute** SHALL be considered present if there is at least one **attribute** that  
2390 matches the criteria set out below.

2391 The <AttributeDesignator> element SHALL return a **bag** containing all the **attribute** values that are  
2392 matched by the **named attribute**. In the event that no matching **attribute** is present in the **context**, the  
2393 MustBePresent attribute governs whether this element returns an empty **bag** or "Indeterminate". See  
2394 Section 7.3.5.

2395 The <AttributeDesignator> MAY appear in the <Match> element and MAY be passed to the  
2396 <Apply> element as an argument.

2397 The <AttributeDesignator> element is of the AttributeDesignatorType complex type.

```

2398 <xs:complexType name="AttributeDesignatorType">
2399     <xs:complexContent>
2400         <xs:extension base="xacml:ExpressionType">

```

```

2401     <xs:attribute name="Category" type="xs:anyURI"
2402         use="required"/>
2403     <xs:attribute name="AttributeId" type="xs:anyURI"
2404         use="required"/>
2405     <xs:attribute name="DataType" type="xs:anyURI"
2406         use="required"/>
2407     <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2408     <xs:attribute name="MustBePresent" type="xs:boolean"
2409         use="required"/>
2410     </xs:extension>
2411 </xs:complexContent>
2412 </xs:complexType>

```

2413 A **named attribute** SHALL match an **attribute** if the values of their respective `Category`,  
2414 `AttributeId`, `DataType` and `Issuer` attributes match. The attribute designator's `Category` MUST  
2415 match, by URI equality, the `Category` of the `<Attributes>` element in which the **attribute** is present.  
2416 The attribute designator's `AttributeId` MUST match, by URI equality, the `AttributeId` of the  
2417 attribute. The attribute designator's `DataType` MUST match, by URI equality, the `DataType` of the same  
2418 **attribute**.

2419 If the `Issuer` attribute is present in the attribute designator, then it MUST match, using the  
2420 "urn:oasis:names:tc:xacml:1.0:function:string-equal" function, the `Issuer` of the same **attribute**. If the  
2421 `Issuer` is not present in the attribute designator, then the matching of the **attribute** to the **named**  
2422 **attribute** SHALL be governed by `AttributeId` and `DataType` attributes alone.

2423 The `<AttributeDesignatorType>` contains the following attributes:

2424 `Category` [Required]

2425 This attribute SHALL specify the `Category` with which to match the **attribute**.

2426 `AttributeId` [Required]

2427 This attribute SHALL specify the `AttributeId` with which to match the **attribute**.

2428 `DataType` [Required]

2429 The **bag** returned by the `<AttributeDesignator>` element SHALL contain values of this data-  
2430 type.

2431 `Issuer` [Optional]

2432 This attribute, if supplied, SHALL specify the `Issuer` with which to match the **attribute**.

2433 `MustBePresent` [Required]

2434 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the event  
2435 the **named attribute** is absent from the request **context**. See Section 7.3.5. Also see Sections  
2436 7.17.2 and 7.17.3.

## 2437 5.30 Element `<AttributeSelector>`

2438 The `<AttributeSelector>` element produces a **bag** of unnamed and uncategorized **attribute** values.  
2439 The values shall be constructed from the node(s) selected by applying the XPath expression given by the  
2440 element's `Path` attribute to the XML content indicated by the element's `Category` attribute. Support for  
2441 the `<AttributeSelector>` element is OPTIONAL.

2442 See section 7.3.7 for details of `<AttributeSelector>` evaluation.

```

2443 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"
2444     substitutionGroup="xacml:Expression"/>
2445 <xs:complexType name="AttributeSelectorType">
2446     <xs:complexContent>
2447         <xs:extension base="xacml:ExpressionType">
2448             <xs:attribute name="Category" type="xs:anyURI"
2449                 use="required"/>

```

```

2450         <xs:attribute name="ContextSelectorId" type="xs:anyURI"
2451             use="optional"/>
2452         <xs:attribute name="Path" type="xs:string"
2453             use="required"/>
2454         <xs:attribute name="DataType" type="xs:anyURI"
2455             use="required"/>
2456         <xs:attribute name="MustBePresent" type="xs:boolean"
2457             use="required"/>
2458     </xs:extension>
2459 </xs:complexContent>
2460 </xs:complexType>

```

2461 The <AttributeSelector> element is of AttributeSelectorType complex type.

2462 The <AttributeSelector> element has the following attributes:

2463 Category [Required]

2464 This attribute SHALL specify the **attributes** category of the <Content> element containing the  
2465 XML from which nodes will be selected. It also indicates the **attributes** category containing the  
2466 applicable ContextSelectorId attribute, if the element includes a ContextSelectorId xml  
2467 attribute.

2468 ContextSelectorId [Optional]

2469 This attribute refers to the **attribute** (by its AttributeId) in the request **context** in the category  
2470 given by the Category attribute. The referenced **attribute** MUST have data type  
2471 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression, and must select a single node in the  
2472 <Content> element. The XPathCategory attribute of the referenced **attribute** MUST be equal  
2473 to the Category attribute of the **attribute selector**.

2474 Path [Required]

2475 This attribute SHALL contain an XPath expression to be evaluated against the specified XML  
2476 content. See Section 7.3.7 for details of the XPath evaluation during <AttributeSelector>  
2477 processing.

2478 DataType [Required]

2479 The attribute specifies the datatype of the values returned from the evaluation of this  
2480 <AttributeSelector> element.

2481 MustBePresent [Required]

2482 This attribute governs whether the element returns "Indeterminate" or an empty **bag** in the event  
2483 the XPath expression selects no node. See Section 7.3.5. Also see Sections 7.17.2 and 7.17.3.

## 2484 5.31 Element <AttributeValue>

2485 The <AttributeValue> element SHALL contain a literal **attribute** value.

```

2486 <xs:element name="AttributeValue" type="xacml:AttributeValueType"
2487     substitutionGroup="xacml:Expression"/>
2488 <xs:complexType name="AttributeValueType" mixed="true">
2489     <xs:complexContent mixed="true">
2490         <xs:extension base="xacml:ExpressionType">
2491             <xs:sequence>
2492                 <xs:any namespace="##any" processContents="lax"
2493                     minOccurs="0" maxOccurs="unbounded"/>
2494             </xs:sequence>
2495             <xs:attribute name="DataType" type="xs:anyURI"
2496                 use="required"/>
2497             <xs:anyAttribute namespace="##any" processContents="lax"/>
2498         </xs:extension>
2499     </xs:complexContent>

```

2500 `</xs:complexType>`

2501 The `<AttributeValue>` element is of `AttributeValueType` complex type.

2502 The `<AttributeValue>` element has the following attributes:

2503 `DataType` [Required]

2504 The data-type of the *attribute* value.

### 2505 **5.32 Element `<Obligations>`**

2506 The `<Obligations>` element SHALL contain a set of `<Obligation>` elements.

```
2507 <xs:element name="Obligations" type="xacml:ObligationsType"/>
2508 <xs:complexType name="ObligationsType">
2509   <xs:sequence>
2510     <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>
2511   </xs:sequence>
2512 </xs:complexType>
```

2513 The `<Obligations>` element is of `ObligationsType` complexType.

2514 The `<Obligations>` element contains the following element:

2515 `<Obligation>` [One to Many]

2516 A sequence of *obligations*. See Section 5.34.

### 2517 **5.33 Element `<AssociatedAdvice>`**

2518 The `<AssociatedAdvice>` element SHALL contain a set of `<Advice>` elements.

```
2519 <xs:element name="AssociatedAdvice" type="xacml:AssociatedAdviceType"/>
2520 <xs:complexType name="AssociatedAdviceType">
2521   <xs:sequence>
2522     <xs:element ref="xacml:Advice" maxOccurs="unbounded"/>
2523   </xs:sequence>
2524 </xs:complexType>
```

2525 The `<AssociatedAdvice>` element is of `AssociatedAdviceType` complexType.

2526 The `<AssociatedAdvice>` element contains the following element:

2527 `<Advice>` [One to Many]

2528 A sequence of *advice*. See Section 5.35.

### 2529 **5.34 Element `<Obligation>`**

2530 The `<Obligation>` element SHALL contain an identifier for the *obligation* and a set of *attributes* that  
2531 form arguments of the action defined by the *obligation*.

```
2532 <xs:element name="Obligation" type="xacml:ObligationType"/>
2533 <xs:complexType name="ObligationType">
2534   <xs:sequence>
2535     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2536       maxOccurs="unbounded"/>
2537   </xs:sequence>
2538   <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2539 </xs:complexType>
```

2540 The `<Obligation>` element is of `ObligationType` complexType. See Section 7.16 for a description  
2541 of how the set of *obligations* to be returned by the *PDP* is determined.

2542 The `<Obligation>` element contains the following elements and attributes:

2543 `ObligationId` [Required]

2544 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the **PEP**.

2545 <AttributeAssignment> [Optional]

2546 **Obligation** arguments assignment. The values of the **obligation** arguments SHALL be  
2547 interpreted by the **PEP**.

### 2548 5.35 Element <Advice>

2549 The <Advice> element SHALL contain an identifier for the **advice** and a set of **attributes** that form  
2550 arguments of the supplemental information defined by the **advice**.

```
2551 <xs:element name="Advice" type="xacml:AdviceType"/>
2552 <xs:complexType name="AdviceType">
2553   <xs:sequence>
2554     <xs:element ref="xacml:AttributeAssignment" minOccurs="0"
2555     maxOccurs="unbounded"/>
2556   </xs:sequence>
2557   <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>
2558 </xs:complexType>
```

2559 The <Advice> element is of AdviceType complexType. See Section 7.16 for a description of how the  
2560 set of **advice** to be returned by the **PDP** is determined.

2561 The <Advice> element contains the following elements and attributes:

2562 AdviceId [Required]

2563 **Advice** identifier. The value of the **advice** identifier MAY be interpreted by the **PEP**.

2564 <AttributeAssignment> [Optional]

2565 **Advice** arguments assignment. The values of the **advice** arguments MAY be interpreted by the  
2566 **PEP**.

### 2567 5.36 Element <AttributeAssignment>

2568 The <AttributeAssignment> element is used for including arguments in **obligation** and **advice**  
2569 expressions. It SHALL contain an AttributeId and the corresponding **attribute** value, by extending  
2570 the AttributeValueType type definition. The <AttributeAssignment> element MAY be used in  
2571 any way that is consistent with the schema syntax, which is a sequence of <xs:any> elements. The  
2572 value specified SHALL be understood by the **PEP**, but it is not further specified by XACML. See Section  
2573 7.16. Section 4.2.4.3 provides a number of examples of arguments included in **obligation** expressions.

```
2574 <xs:element name="AttributeAssignment" type="xacml:AttributeAssignmentType"/>
2575 <xs:complexType name="AttributeAssignmentType" mixed="true">
2576   <xs:complexContent>
2577     <xs:extension base="xacml:AttributeValueType">
2578       <xs:attribute name="AttributeId" type="xs:anyURI"
2579       use="required"/>
2580       <xs:attribute name="Category" type="xs:anyURI"
2581       use="optional"/>
2582       <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2583     </xs:extension>
2584   </xs:complexContent>
2585 </xs:complexType>
```

2586 The <AttributeAssignment> element is of AttributeAssignmentType complex type.

2587 The <AttributeAssignment> element contains the following attributes:

2588 AttributeId [Required]

2589 The **attribute** Identifier.

2590 Category [Optional]

2591 An optional category of the **attribute**. If this attribute is missing, the **attribute** has no category.  
2592 The **PEP** SHALL interpret the significance and meaning of any `Category` attribute. Non-  
2593 normative note: an expected use of the category is to disambiguate **attributes** which are relayed  
2594 from the request.

2595 Issuer [Optional]

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

### 2599 5.37 Element <ObligationExpressions>

2600 The <ObligationExpressions> element SHALL contain a set of <ObligationExpression>  
2601 elements.

```
2602 <xs:element name="ObligationExpressions"  
2603     type="xacml:ObligationExpressionsType"/>  
2604 <xs:complexType name="ObligationExpressionsType">  
2605     <xs:sequence>  
2606         <xs:element ref="xacml:ObligationExpression" maxOccurs="unbounded"/>  
2607     </xs:sequence>  
2608 </xs:complexType>
```

2609 The <ObligationExpressions> element is of `ObligationExpressionsType` `complexType`.

2610 The <ObligationExpressions> element contains the following element:

2611 <ObligationExpression> [One to Many]

2612 A sequence of **obligations** expressions. See Section 5.39.

### 2613 5.38 Element <AdviceExpressions>

2614 The <AdviceExpressions> element SHALL contain a set of <AdviceExpression> elements.

```
2615 <xs:element name="AdviceExpressions" type="xacml:AdviceExpressionsType"/>  
2616 <xs:complexType name="AdviceExpressionsType">  
2617     <xs:sequence>  
2618         <xs:element ref="xacml:AdviceExpression" maxOccurs="unbounded"/>  
2619     </xs:sequence>  
2620 </xs:complexType>
```

2621 The <AdviceExpressions> element is of `AdviceExpressionsType` `complexType`.

2622 The <AdviceExpressions> element contains the following element:

2623 <AdviceExpression> [One to Many]

2624 A sequence of **advice** expressions. See Section 5.40.

### 2625 5.39 Element <ObligationExpression>

2626 The <ObligationExpression> element evaluates to an **obligation** and SHALL contain an identifier  
2627 for an **obligation** and a set of expressions that form arguments of the action defined by the **obligation**.  
2628 The `FulfillOn` attribute SHALL indicate the **effect** for which this **obligation** must be fulfilled by the  
2629 **PEP**.

```
2630 <xs:element name="ObligationExpression"  
2631     type="xacml:ObligationExpressionType"/>  
2632 <xs:complexType name="ObligationExpressionType">  
2633     <xs:sequence>  
2634         <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"  
2635             maxOccurs="unbounded"/>  
2636     </xs:sequence>
```

```

2637 <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2638 <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
2639 </xs:complexType>

```

2640 The <ObligationExpression> element is of ObligationExpressionType complexType. See  
 2641 Section 7.16 for a description of how the set of **obligations** to be returned by the **PDP** is determined.

2642 The <ObligationExpression> element contains the following elements and attributes:

2643 ObligationId [Required]

2644 **Obligation** identifier. The value of the **obligation** identifier SHALL be interpreted by the **PEP**.

2645 FulfillOn [Required]

2646 The **effect** for which this **obligation** must be fulfilled by the **PEP**.

2647 <AttributeAssignmentExpression> [Optional]

2648 **Obligation** arguments in the form of expressions. The expressions SHALL be evaluated by the  
 2649 PDP to constant <AttributeValue> elements or **bags**, which shall be the attribute  
 2650 assignments in the <Obligation> returned to the PEP. If an  
 2651 <AttributeAssignmentExpression> evaluates to an atomic **attribute** value, then there  
 2652 MUST be one resulting <AttributeAssignment> which MUST contain this single **attribute**  
 2653 value. If the <AttributeAssignmentExpression> evaluates to a **bag**, then there MUST be a  
 2654 resulting <AttributeAssignment> for each of the values in the **bag**. If the **bag** is empty, there  
 2655 shall be no <AttributeAssignment> from this <AttributeAssignmentExpression>.The  
 2656 values of the **obligation** arguments SHALL be interpreted by the **PEP**.

## 2657 5.40 Element <AdviceExpression>

2658 The <AdviceExpression> element evaluates to an **advice** and SHALL contain an identifier for an  
 2659 **advice** and a set of expressions that form arguments of the supplemental information defined by the  
 2660 **advice**. The AppliesTo attribute SHALL indicate the **effect** for which this **advice** must be provided to  
 2661 the **PEP**.

```

2662 <xs:element name="AdviceExpression" type="xacml:AdviceExpressionType"/>
2663 <xs:complexType name="AdviceExpressionType">
2664 <xs:sequence>
2665 <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"
2666 maxOccurs="unbounded"/>
2667 </xs:sequence>
2668 <xs:attribute name="AdviceId" type="xs:anyURI" use="required"/>
2669 <xs:attribute name="AppliesTo" type="xacml:EffectType" use="required"/>
2670 </xs:complexType>

```

2671 The <AdviceExpression> element is of AdviceExpressionType complexType. See Section 7.16  
 2672 for a description of how the set of **advice** to be returned by the **PDP** is determined.

2673 The <AdviceExpression> element contains the following elements and attributes:

2674 AdviceId [Required]

2675 **Advice** identifier. The value of the **advice** identifier MAY be interpreted by the **PEP**.

2676 AppliesTo [Required]

2677 The **effect** for which this **advice** must be provided to the **PEP**.

2678 <AttributeAssignmentExpression> [Optional]

2679 **Advice** arguments in the form of expressions. The expressions SHALL be evaluated by the PDP  
 2680 to constant <AttributeValue> elements or **bags**, which shall be the attribute assignments in  
 2681 the <Advice> returned to the PEP. If an <AttributeAssignmentExpression> evaluates to  
 2682 an atomic **attribute** value, then there MUST be one resulting <AttributeAssignment> which  
 2683 MUST contain this single **attribute** value. If the <AttributeAssignmentExpression>

2684 evaluates to a **bag**, then there MUST be a resulting <AttributeAssignment> for each of the  
2685 values in the **bag**. If the **bag** is empty, there shall be no <AttributeAssignment> from this  
2686 <AttributeAssignmentExpression>. The values of the **advice** arguments MAY be  
2687 interpreted by the **PEP**.

## 2688 5.41 Element <AttributeAssignmentExpression>

2689 The <AttributeAssignmentExpression> element is used for including arguments in **obligations**  
2690 and **advice**. It SHALL contain an AttributeId and an expression which SHALL be evaluated into the  
2691 corresponding **attribute** value. The value specified SHALL be understood by the **PEP**, but it is not further  
2692 specified by XACML. See Section 7.16. Section 4.2.4.3 provides a number of examples of arguments  
2693 included in **obligations**.

```
2694 <xs:element name="AttributeAssignmentExpression"  
2695     type="xacml:AttributeAssignmentExpressionType"/>  
2696 <xs:complexType name="AttributeAssignmentExpressionType">  
2697     <xs:sequence>  
2698         <xs:element ref="xacml:Expression"/>  
2699     </xs:sequence>  
2700     <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2701     <xs:attribute name="Category" type="xs:anyURI" use="optional"/>  
2702     <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
2703 </xs:complexType>
```

2704 The <AttributeAssignmentExpression> element is of AttributeAssignmentExpressionType  
2705 complex type.

2706 The <AttributeAssignmentExpression> element contains the following attributes:

2707 <Expression> [Required]

2708 The expression which evaluates to a constant **attribute** value or a bag of zero or more attribute  
2709 values. See section 5.25.

2710 AttributeId [Required]

2711 The **attribute** identifier. The value of the AttributeId attribute in the resulting  
2712 <AttributeAssignment> element MUST be equal to this value.

2713 Category [Optional]

2714 An optional category of the **attribute**. If this attribute is missing, the **attribute** has no category.  
2715 The value of the Category attribute in the resulting <AttributeAssignment> element MUST be  
2716 equal to this value.

2717 Issuer [Optional]

2718 An optional issuer of the **attribute**. If this attribute is missing, the **attribute** has no issuer. The  
2719 value of the Issuer attribute in the resulting <AttributeAssignment> element MUST be equal to  
2720 this value.

## 2721 5.42 Element <Request>

2722 The <Request> element is an abstraction layer used by the **policy** language. For simplicity of  
2723 expression, this document describes **policy** evaluation in terms of operations on the **context**. However a  
2724 conforming **PDP** is not required to actually instantiate the **context** in the form of an XML document. But,  
2725 any system conforming to the XACML specification MUST produce exactly the same **authorization**  
2726 **decisions** as if all the inputs had been transformed into the form of an <Request> element.

2727 The <Request> element contains <Attributes> elements. There may be multiple <Attributes>  
2728 elements with the same Category attribute if the **PDP** implements the multiple decision profile, see  
2729 **[Multi]**. Under other conditions, it is a syntax error if there are multiple <Attributes> elements with the  
2730 same Category (see Section 7.17.2 for error codes).

```

2731 <xs:element name="Request" type="xacml:RequestType"/>
2732 <xs:complexType name="RequestType">
2733   <xs:sequence>
2734     <xs:element ref="xacml:RequestDefaults" minOccurs="0"/>
2735     <xs:element ref="xacml:Attributes" maxOccurs="unbounded"/>
2736     <xs:element ref="xacml:MultiRequests" minOccurs="0"/>
2737   </xs:sequence>
2738   <xs:attribute name="ReturnPolicyIdList" type="xs:boolean" use="required"/>
2739   <xs:attribute name="CombinedDecision" type="xs:boolean" use="required" />
2740 </xs:complexType>

```

2741 The <Request> element is of RequestType complex type.

2742 The <Request> element contains the following elements and attributes:

2743 ReturnPolicyIdList [Required]

2744 This attribute is used to request that the **PDP** return a list of all fully applicable **policies** and **policy sets** which were used in the decision as a part of the decision response.

2746 CombinedDecision [Required]

2747 This attribute is used to request that the **PDP** combines multiple decisions into a single decision.  
 2748 The use of this attribute is specified in [Multi]. If the **PDP** does not implement the relevant  
 2749 functionality in [Multi], then the **PDP** must return an Indeterminate with a status code of  
 2750 urn:oasis:names:tc:xacml:1.0:status:processing-error if it receives a request with this attribute set  
 2751 to "true".

2752 <RequestDefaults> [Optional]

2753 Contains default values for the request, such as XPath version. See section 5.43.

2754 <Attributes> [One to Many]

2755 Specifies information about **attributes** of the request **context** by listing a sequence of  
 2756 <Attribute> elements associated with an **attribute** category. One or more <Attributes>  
 2757 elements are allowed. Different <Attributes> elements with different categories are used to  
 2758 represent information about the **subject**, **resource**, **action**, **environment** or other categories of  
 2759 the **access** request.

2760 <MultiRequests> [Optional]

2761 Lists multiple **request contexts** by references to the <Attributes> elements. Implementation  
 2762 of this element is optional. The semantics of this element is defined in [Multi]. If the  
 2763 implementation does not implement this element, it MUST return an Indeterminate result if it  
 2764 encounters this element. See section 5.50.

## 2765 5.43 Element <RequestDefaults>

2766 The <RequestDefaults> element SHALL specify default values that apply to the <Request> element.

```

2767 <xs:element name="RequestDefaults" type="xacml:RequestDefaultsType"/>
2768 <xs:complexType name="RequestDefaultsType">
2769   <xs:sequence>
2770     <xs:choice>
2771       <xs:element ref="xacml:XPathVersion"/>
2772     </xs:choice>
2773   </xs:sequence>
2774 </xs:complexType>

```

2775 <RequestDefaults> element is of RequestDefaultsType complex type.

2776 The <RequestDefaults> element contains the following elements:

2777 <XPathVersion> [Optional]

2778 Default XPath version for XPath expressions occurring in the request.

## 2779 5.44 Element <Attributes>

2780 The <Attributes> element specifies **attributes** of a **subject**, **resource**, **action**, **environment** or  
2781 another category by listing a sequence of <Attribute> elements associated with the category.

```
2782 <xs:element name="Attributes" type="xacml:AttributesType"/>
2783 <xs:complexType name="AttributesType">
2784   <xs:sequence>
2785     <xs:element ref="xacml:Content" minOccurs="0"/>
2786     <xs:element ref="xacml:Attribute" minOccurs="0"
2787       maxOccurs="unbounded"/>
2788   </xs:sequence>
2789   <xs:attribute name="Category" type="xs:anyURI" use="required"/>
2790   <xs:attribute ref="xml:id" use="optional"/>
2791 </xs:complexType><xs:complexType name="SubjectType">
```

2792 The <Attributes> element is of AttributesType complex type.

2793 The <Attributes> element contains the following elements and attributes:

2794 Category [Required]

2795 This attribute indicates which **attribute** category the contained **attributes** belong to. The  
2796 Category attribute is used to differentiate between **attributes** of **subject**, **resource**, **action**,  
2797 **environment** or other categories.

2798 xml:id [Optional]

2799 This attribute provides a unique identifier for this <Attributes> element. See [XMLid] It is  
2800 primarily intended to be referenced in multiple requests. See [Multi].

2801 <Content> [Optional]

2802 Specifies additional sources of **attributes** in free form XML document format which can be  
2803 referenced using <AttributeSelector> elements.

2804 <Attribute> [Any Number]

2805 A sequence of **attributes** that apply to the category of the request.

## 2806 5.45 Element <Content>

2807 The <Content> element is a notional placeholder for additional **attributes**, typically the content of the  
2808 **resource**.

```
2809 <xs:element name="Content" type="xacml:ContentType"/>
2810 <xs:complexType name="ContentType" mixed="true">
2811   <xs:sequence>
2812     <xs:any namespace="##any" processContents="lax"/>
2813   </xs:sequence>
2814 </xs:complexType>
```

2815 The <Content> element is of ContentType complex type.

2816 The <Content> element has exactly one arbitrary type child element.

## 2817 5.46 Element <Attribute>

2818 The <Attribute> element is the central abstraction of the request **context**. It contains **attribute** meta-  
2819 data and one or more **attribute** values. The **attribute** meta-data comprises the **attribute** identifier and  
2820 the **attribute** issuer. <AttributeDesignator> elements in the **policy** MAY refer to **attributes** by  
2821 means of this meta-data.

```
2822 <xs:element name="Attribute" type="xacml:AttributeType"/>
2823 <xs:complexType name="AttributeType">
2824   <xs:sequence>
```

```

2825     <xs:element ref="xacml:AttributeValue" maxOccurs="unbounded"/>
2826 </xs:sequence>
2827 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
2828 <xs:attribute name="Issuer" type="xs:string" use="optional"/>
2829 <xs:attribute name="IncludeInResult" type="xs:boolean" use="required"/>
2830 </xs:complexType>

```

2831 The <Attribute> element is of AttributeType complex type.

2832 The <Attribute> element contains the following attributes and elements:

2833 AttributeId [Required]

2834       The **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly  
2835       used **attributes**. See Appendix B.

2836 Issuer [Optional]

2837       The **Attribute** issuer. For example, this attribute value MAY be an x500Name that binds to a  
2838       public key, or it may be some other identifier exchanged out-of-band by issuing and relying  
2839       parties.

2840 IncludeInResult [Default: false]

2841       Whether to include this **attribute** in the result. This is useful to correlate requests with their  
2842       responses in case of multiple requests.

2843 <AttributeValue> [One to Many]

2844       One or more **attribute** values. Each **attribute** value MAY have contents that are empty, occur  
2845       once or occur multiple times.

## 2846 5.47 Element <Response>

2847 The <Response> element is an abstraction layer used by the **policy** language. Any proprietary system  
2848 using the XACML specification MUST transform an XACML **context** <Response> element into the form  
2849 of its **authorization decision**.

2850 The <Response> element encapsulates the **authorization decision** produced by the **PDP**. It includes a  
2851 sequence of one or more results, with one <Result> element per requested **resource**. Multiple results  
2852 MAY be returned by some implementations, in particular those that support the XACML Profile for  
2853 Requests for Multiple Resources [**Multi**]. Support for multiple results is OPTIONAL.

```

2854 <xs:element name="Response" type="xacml:ResponseType"/>
2855 <xs:complexType name="ResponseType">
2856 <xs:sequence>
2857 <xs:element ref="xacml:Result" maxOccurs="unbounded"/>
2858 </xs:sequence>
2859 </xs:complexType>

```

2860 The <Response> element is of ResponseType complex type.

2861 The <Response> element contains the following elements:

2862 <Result> [One to Many]

2863       An **authorization decision** result. See Section 5.48.

## 2864 5.48 Element <Result>

2865 The <Result> element represents an **authorization decision** result. It MAY include a set of  
2866 **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or cannot fulfill an  
2867 **obligation**, then the action of the PEP is determined by its bias, see section 7.1. It MAY include a set of  
2868 **advice** with supplemental information which MAY be safely ignored by the **PEP**.

```

2869 <xs:complexType name="ResultType">
2870 <xs:sequence>

```

```

2871     <xs:element ref="xacml:Decision"/>
2872     <xs:element ref="xacml:Status" minOccurs="0"/>
2873     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2874     <xs:element ref="xacml:AssociatedAdvice" minOccurs="0"/>
2875     <xs:element ref="xacml:Attributes" minOccurs="0"
2876         maxOccurs="unbounded"/>
2877     <xs:element ref="xacml:PolicyIdentifierList" minOccurs="0"/>
2878 </xs:sequence>
2879 </xs:complexType>

```

2880 The <Result> element is of ResultType complex type.

2881 The <Result> element contains the following attributes and elements:

2882 <Decision> [Required]

2883 The **authorization decision**: “Permit”, “Deny”, “Indeterminate” or “NotApplicable”.

2884 <Status> [Optional]

2885 Indicates whether errors occurred during evaluation of the **decision request**, and optionally,  
2886 information about those errors. If the <Response> element contains <Result> elements whose  
2887 <Status> elements are all identical, and the <Response> element is contained in a protocol  
2888 wrapper that can convey status information, then the common status information MAY be placed  
2889 in the protocol wrapper and this <Status> element MAY be omitted from all <Result>  
2890 elements.

2891 <Obligations> [Optional]

2892 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand or cannot  
2893 fulfill an **obligation**, then the action of the PEP is determined by its bias, see section 7.2. See  
2894 Section 7.16 for a description of how the set of **obligations** to be returned by the **PDP** is  
2895 determined.

2896 <AssociatedAdvice> [Optional]

2897 A list of **advice** that provide supplemental information to the **PEP**. If the **PEP** does not  
2898 understand an **advice**, the PEP may safely ignore the **advice**. See Section 7.16 for a description  
2899 of how the set of **advice** to be returned by the **PDP** is determined.

2900 <Attributes> [Optional]

2901 A list of **attributes** that were part of the request. The choice of which **attributes** are included here  
2902 is made with the IncludeInResult attribute of the <Attribute> elements of the request. See  
2903 section 5.46.

2904 <PolicyIdentifierList> [Optional]

2905 If the ReturnPolicyIdList attribute in the <Request> is true (see section 5.42), a **PDP** that  
2906 implements this optional feature MUST return a list of all **policies** which were found to be fully  
2907 applicable. That is, all **policies** where both the <Target> matched and the <Condition>  
2908 evaluated to true, whether or not the <Effect> was the same or different from the <Decision>.

## 2909 5.49 Element <PolicyIdentifierList>

2910 The <PolicyIdentifierList> element contains a list of **policy** and **policy set** identifiers of **policies**  
2911 which have been applicable to a request. The list is unordered.

```

2912 <xs:element name="PolicyIdentifierList"
2913     type="xacml:PolicyIdentifierListType"/>
2914 <xs:complexType name="PolicyIdentifierListType">
2915     <xs:choice minOccurs="0" maxOccurs="unbounded">
2916         <xs:element ref="xacml:PolicyIdReference"/>
2917         <xs:element ref="xacml:PolicySetIdReference"/>
2918     </xs:choice>

```

2919 `</xs:complexType>`

2920 The `<PolicyIdentifierList>` element is of `PolicyIdentifierListType` complex type.

2921 The `<PolicyIdentifierList>` element contains the following elements.

2922 `<PolicyIdReference>` [Any number]

2923 The identifier and version of a **policy** which was applicable to the request. See section 5.11. The  
2924 `<PolicyIdReference>` element MUST use the `Version` attribute to specify the version and  
2925 MUST NOT use the `LatestVersion` or `EarliestVersion` attributes.

2926 `<PolicySetIdReference>` [Any number]

2927 The identifier and version of a **policy set** which was applicable to the request. See section 5.10.  
2928 The `<PolicySetIdReference>` element MUST use the `Version` attribute to specify the  
2929 version and MUST NOT use the `LatestVersion` or `EarliestVersion` attributes.

## 2930 **5.50 Element `<MultiRequests>`**

2931 The `<MultiRequests>` element contains a list of requests by reference to `<Attributes>` elements in  
2932 the enclosing `<Request>` element. The semantics of this element are defined in **[Multi]**. Support for this  
2933 element is optional. If an implementation does not support this element, but receives it, the  
2934 implementation MUST generate an "Indeterminate" response.

```
2935 <xs:element name="MultiRequests" type="xacml:MultiRequestsType"/>
2936 <xs:complexType name="MultiRequestsType">
2937   <xs:sequence>
2938     <xs:element ref="xacml:RequestReference" maxOccurs="unbounded"/>
2939   </xs:sequence>
2940 </xs:complexType>
```

2941 The `<MultiRequests>` element contains the following elements.

2942 `<RequestReference>` [one to many]

2943 Defines a request instance by reference to `<Attributes>` elements in the enclosing  
2944 `<Request>` element. See section 5.51.

## 2945 **5.51 Element `<RequestReference>`**

2946 The `<RequestReference>` element defines an instance of a request in terms of references to  
2947 `<Attributes>` elements. The semantics of this element are defined in **[Multi]**. Support for this element  
2948 is optional.

```
2949 <xs:element name="RequestReference" type="xacml:RequestReference"/>
2950 <xs:complexType name="RequestReferenceType">
2951   <xs:sequence>
2952     <xs:element ref="xacml:AttributesReference" maxOccurs="unbounded"/>
2953   </xs:sequence>
2954 </xs:complexType>
```

2955 The `<RequestReference>` element contains the following elements.

2956 `<AttributesReference>` [one to many]

2957 A reference to an `<Attributes>` element in the enclosing `<Request>` element. See section  
2958 5.52.

## 2959 **5.52 Element `<AttributesReference>`**

2960 The `<AttributesReference>` element makes a reference to an `<Attributes>` element. The  
2961 meaning of this element is defined in **[Multi]**. Support for this element is optional.

```
2962 <xs:element name="AttributesReference" type="xacml:AttributesReference"/>
```

```
2963 <xs:complexType name="AttributesReferenceType">
2964   <xs:attribute name="ReferenceId" type="xs:IDREF" use="required" />
2965 </xs:complexType>
```

2966 The <AttributesReference> element contains the following attributes.

2967 ReferenceId [required]

2968 A reference to the `xml:id` attribute of an <Attributes> element in the enclosing <Request>  
2969 element.

## 2970 5.53 Element <Decision>

2971 The <Decision> element contains the result of *policy* evaluation.

```
2972 <xs:element name="Decision" type="xacml:DecisionType"/>
2973 <xs:simpleType name="DecisionType">
2974   <xs:restriction base="xs:string">
2975     <xs:enumeration value="Permit"/>
2976     <xs:enumeration value="Deny"/>
2977     <xs:enumeration value="Indeterminate"/>
2978     <xs:enumeration value="NotApplicable"/>
2979   </xs:restriction>
2980 </xs:simpleType>
```

2981 The <Decision> element is of `DecisionType` simple type.

2982 The values of the <Decision> element have the following meanings:

2983 "Permit": the requested **access** is permitted.

2984 "Deny": the requested **access** is denied.

2985 "Indeterminate": the **PDP** is unable to evaluate the requested **access**. Reasons for such inability  
2986 include: missing **attributes**, network errors while retrieving **policies**, division by zero during  
2987 **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc.

2988 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

## 2989 5.54 Element <Status>

2990 The <Status> element represents the status of the **authorization decision** result.

```
2991 <xs:element name="Status" type="xacml:StatusType"/>
2992 <xs:complexType name="StatusType">
2993   <xs:sequence>
2994     <xs:element ref="xacml:StatusCode"/>
2995     <xs:element ref="xacml:StatusMessage" minOccurs="0"/>
2996     <xs:element ref="xacml:StatusDetail" minOccurs="0"/>
2997   </xs:sequence>
2998 </xs:complexType>
```

2999 The <Status> element is of `StatusType` complex type.

3000 The <Status> element contains the following elements:

3001 <StatusCode> [Required]

3002 Status code.

3003 <StatusMessage> [Optional]

3004 A status message describing the status code.

3005 <StatusDetail> [Optional]

3006 Additional status information.

## 3007 5.55 Element <StatusCode>

3008 The <StatusCode> element contains a major status code value and an optional sequence of minor  
3009 status codes.

```
3010 <xs:element name="StatusCode" type="xacml:StatusCodeType"/>  
3011 <xs:complexType name="StatusCodeType">  
3012   <xs:sequence>  
3013     <xs:element ref="xacml:StatusCode" minOccurs="0"/>  
3014   </xs:sequence>  
3015   <xs:attribute name="Value" type="xs:anyURI" use="required"/>  
3016 </xs:complexType>
```

3017 The <StatusCode> element is of StatusCodeType complex type.

3018 The <StatusCode> element contains the following attributes and elements:

3019 Value [Required]

3020 See Section B.8 for a list of values.

3021 <StatusCode> [Any Number]

3022 Minor status code. This status code qualifies its parent status code.

## 3023 5.56 Element <StatusMessage>

3024 The <StatusMessage> element is a free-form description of the status code.

```
3025 <xs:element name="StatusMessage" type="xs:string"/>
```

3026 The <StatusMessage> element is of xs:string type.

## 3027 5.57 Element <StatusDetail>

3028 The <StatusDetail> element qualifies the <Status> element with additional information.

```
3029 <xs:element name="StatusDetail" type="xacml:StatusDetailType"/>  
3030 <xs:complexType name="StatusDetailType">  
3031   <xs:sequence>  
3032     <xs:any namespace="##any" processContents="lax" minOccurs="0"  
3033       maxOccurs="unbounded"/>  
3034   </xs:sequence>  
3035 </xs:complexType>
```

3036 The <StatusDetail> element is of StatusDetailType complex type.

3037 The <StatusDetail> element allows arbitrary XML content.

3038 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the following  
3039 XACML-defined <StatusCode> values and includes a <StatusDetail> element, then the following  
3040 rules apply.

3041 urn:oasis:names:tc:xacml:1.0:status:ok

3042 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “ok” status value.

3043 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

3044 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a  
3045 <StatusDetail> element containing one or more <MissingAttributeDetail> elements.

3046 urn:oasis:names:tc:xacml:1.0:status:syntax-error

3047 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the “syntax-error” status  
3048 value. A syntax error may represent either a problem with the **policy** being used or with the request  
3049 **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

3050 urn:oasis:names:tc:xacml:1.0:status:processing-error

3051 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the “processing-error” status  
3052 value. This status code indicates an internal problem in the **PDP**. For security reasons, the **PDP** MAY  
3053 choose to return no further information to the **PEP**. In the case of a divide-by-zero error or other  
3054 computational error, the **PDP** MAY return a <StatusMessage> describing the nature of the error.

## 3055 5.58 Element <MissingAttributeDetail>

3056 The <MissingAttributeDetail> element conveys information about **attributes** required for **policy**  
3057 evaluation that were missing from the request **context**.

```
3058 <xs:element name="MissingAttributeDetail"  
3059 type="xacml:MissingAttributeDetailType"/>  
3060 <xs:complexType name="MissingAttributeDetailType">  
3061 <xs:sequence>  
3062 <xs:element ref="xacml:AttributeValue" minOccurs="0"  
3063 maxOccurs="unbounded"/>  
3064 </xs:sequence>  
3065 <xs:attribute name="Category" type="xs:anyURI" use="required"/>  
3066 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
3067 <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
3068 <xs:attribute name="Issuer" type="xs:string" use="optional"/>  
3069 </xs:complexType>
```

3070 The <MissingAttributeDetail> element is of MissingAttributeDetailType complex type.

3071 The <MissingAttributeDetail> element contains the following attributes and elements:

3072 <AttributeValue> [Optional]

3073 The required value of the missing **attribute**.

3074 Category [Required]

3075 The category identifier of the missing **attribute**.

3076 AttributeId [Required]

3077 The identifier of the missing **attribute**.

3078 DataType [Required]

3079 The data-type of the missing **attribute**.

3080 Issuer [Optional]

3081 This attribute, if supplied, SHALL specify the required Issuer of the missing **attribute**.

3082 If the **PDP** includes <AttributeValue> elements in the <MissingAttributeDetail> element, then  
3083 this indicates the acceptable values for that **attribute**. If no <AttributeValue> elements are included,  
3084 then this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list of  
3085 **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the missing  
3086 values or **attributes** will be sufficient to satisfy the **policy**.

3087

## 6 XPath 2.0 definitions

3088 The XPath 2.0 specification leaves a number of aspects of behavior implementation defined. This section  
3089 defines how XPath 2.0 SHALL behave when hosted in XACML.

3090 <http://www.w3.org/TR/2007/REC-xpath20-20070123/#id-impl-defined-items> defines the following items:

- 3091 1. The version of Unicode that is used to construct expressions.  
3092 XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.
- 3093 2. The statically-known collations.  
3094 XACML leaves this implementation defined.
- 3095 3. The implicit timezone.  
3096 XACML defined the implicit time zone as UTC.
- 3097 4. The circumstances in which warnings are raised, and the ways in which warnings are handled.  
3098 XACML leaves this implementation defined.
- 3099 5. The method by which errors are reported to the external processing environment.  
3100 An XPath error causes an XACML Indeterminate value in the element where the XPath error  
3101 occurs. The StatusCode value SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-error".  
3102 Implementations MAY provide additional details about the error in the response or by some other  
3103 means.
- 3104 6. Whether the implementation is based on the rules of XML 1.0 or 1.1.  
3105 XACML is based on XML 1.0.
- 3106 7. Whether the implementation supports the namespace axis.  
3107 XACML leaves this implementation defined. It is RECOMMENDED that users of XACML do not  
3108 make use of the namespace axis.
- 3109 8. Any static typing extensions supported by the implementation, if the Static Typing Feature is  
3110 supported.  
3111 XACML leaves this implementation defined.

3112

3113 <http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/#implementation-defined> defines the  
3114 following items:

- 3115 1. Support for additional user-defined or implementation-defined types is implementation-defined.  
3116 It is RECOMMENDED that implementations of XACML do not define any additional types and it is  
3117 RECOMMENDED that users of XACML do not make user of any additional types.
- 3118 2. Some typed values in the data model are undefined. Attempting to access an undefined property  
3119 is always an error. Behavior in these cases is implementation-defined and the host language is  
3120 responsible for determining the result.  
3121 An XPath error causes an XACML Indeterminate value in the element where the XPath error  
3122 occurs. The StatusCode value SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-error".  
3123 Implementations MAY provide additional details about the error in the response or by some other  
3124 means.

3125

3126 <http://www.w3.org/TR/2007/REC-xpath-functions-20070123/#impl-def> defines the following items:

- 3127 1. The destination of the trace output is implementation-defined.  
3128 XACML leaves this implementation defined.
- 3129 2. For xs:integer operations, implementations that support limited-precision integer operations must  
3130 either raise an error [err:FOAR0002] or provide an implementation-defined mechanism that  
3131 allows users to choose between raising an error and returning a result that is modulo the largest  
3132 representable integer value.  
3133 XACML leaves this implementation defined. If an implementation chooses to raise an error, the

- 3134            StatusCode value SHALL be “urn:oasis:names:tc:xacml:1.0:status:processing-error”.
- 3135            Implementations MAY provide additional details about the error in the response or by some other
- 3136            means.
- 3137            3. For xs:decimal values the number of digits of precision returned by the numeric operators is
- 3138            implementation-defined.
- 3139            XACML leaves this implementation defined.
- 3140            4. If the number of digits in the result of a numeric operation exceeds the number of digits that the
- 3141            implementation supports, the result is truncated or rounded in an implementation-defined manner.
- 3142            XACML leaves this implementation defined.
- 3143            5. It is implementation-defined which version of Unicode is supported.
- 3144            XACML leaves this implementation defined. It is RECOMMENDED that the latest version is used.
- 3145            6. For fn:normalize-unicode, conforming implementations must support normalization form "NFC"
- 3146            and may support normalization forms "NFD", "NFKC", "NFKD", "FULLY-NORMALIZED". They
- 3147            may also support other normalization forms with implementation-defined semantics.
- 3148            XACML leaves this implementation defined.
- 3149            7. The ability to decompose strings into collation units suitable for substring matching is an
- 3150            implementation-defined property of a collation.
- 3151            XACML leaves this implementation defined.
- 3152            8. All minimally conforming processors must support year values with a minimum of 4 digits (i.e.,
- 3153            YYYY) and a minimum fractional second precision of 1 millisecond or three digits (i.e., s.sss).
- 3154            However, conforming processors may set larger implementation-defined limits on the maximum
- 3155            number of digits they support in these two situations.
- 3156            XACML leaves this implementation defined, and it is RECOMMENDED that users of XACML do
- 3157            not expect greater limits and precision.
- 3158            9. The result of casting a string to xs:decimal, when the resulting value is not too large or too small
- 3159            but nevertheless has too many decimal digits to be accurately represented, is implementation-
- 3160            defined.
- 3161            XACML leaves this implementation defined.
- 3162            10. Various aspects of the processing provided by fn:doc are implementation-defined.
- 3163            Implementations may provide external configuration options that allow any aspect of the
- 3164            processing to be controlled by the user.
- 3165            XACML leaves this implementation defined.
- 3166            11. The manner in which implementations provide options to weaken the stable characteristic of
- 3167            fn:collection and fn:doc are implementation-defined.
- 3168            XACML leaves this implementation defined.

---

## 3169 7 Functional requirements

3170 This section specifies certain functional requirements that are not directly associated with the production  
3171 or consumption of a particular XACML element.

### 3172 7.1 Unicode issues

#### 3173 7.1.1 Normalization

3174 In Unicode, some equivalent characters can be represented by more than one different Unicode  
3175 character sequence. See [CMF]. The process of converting Unicode strings into equivalent character  
3176 sequences is called "normalization" [UAX15]. Some operations, such as string comparison, are sensitive  
3177 to normalization. An operation is normalization-sensitive if its output(s) are different depending on the  
3178 state of normalization of the input(s); if the output(s) are textual, they are deemed different only if they  
3179 would remain different were they to be normalized.

3180 For more information on normalization see [CM].

3181 An XACML implementation MUST behave as if each normalization-sensitive operation normalizes input  
3182 strings into Unicode Normalization Form C ("NFC"). An implementation MAY use some other form of  
3183 internal processing (such as using a non-Unicode, "legacy" character encoding) as long as the externally  
3184 visible results are identical to this specification.

#### 3185 7.1.2 Version of Unicode

3186 The version of Unicode used by XACML is implementation defined. It is RECOMMENDED that the latest  
3187 version is used. Also note security issues in section 9.3.

### 3188 7.2 Policy enforcement point

3189 This section describes the requirements for the *PEP*.

3190 An application functions in the role of the *PEP* if it guards *access* to a set of *resources* and asks the  
3191 *PDP* for an *authorization decision*. The *PEP* MUST abide by the *authorization decision* as described  
3192 in one of the following sub-sections

3193 In any case any *advice* in the *decision* may be safely ignored by the *PEP*.

#### 3194 7.2.1 Base PEP

3195 If the *decision* is "Permit", then the *PEP* SHALL permit *access*. If *obligations* accompany the *decision*,  
3196 then the *PEP* SHALL permit *access* only if it understands and it can and will discharge those  
3197 *obligations*.

3198 If the *decision* is "Deny", then the *PEP* SHALL deny *access*. If *obligations* accompany the *decision*,  
3199 then the *PEP* shall deny *access* only if it understands, and it can and will discharge those *obligations*.

3200 If the *decision* is "Not Applicable", then the *PEP*'s behavior is undefined.

3201 If the *decision* is "Indeterminate", then the *PEP*'s behavior is undefined.

#### 3202 7.2.2 Deny-biased PEP

3203 If the *decision* is "Permit", then the *PEP* SHALL permit *access*. If *obligations* accompany the *decision*,  
3204 then the *PEP* SHALL permit *access* only if it understands and it can and will discharge those  
3205 *obligations*.

3206 All other *decisions* SHALL result in the denial of *access*.

3207 Note: other actions, e.g. consultation of additional *PDPs*, reformulation/resubmission of  
3208 the *decision request*, etc., are not prohibited.

### 3209 7.2.3 Permit-biased PEP

3210 If the *decision* is "Deny", then the *PEP* SHALL deny *access*. If *obligations* accompany the *decision*,  
3211 then the *PEP* shall deny *access* only if it understands, and it can and will discharge those *obligations*.

3212 All other *decisions* SHALL result in the permission of *access*.

3213 Note: other actions, e.g. consultation of additional *PDPs*, reformulation/resubmission of  
3214 the *decision request*, etc., are not prohibited.

## 3215 7.3 Attribute evaluation

3216 *Attributes* are represented in the request *context* by the *context handler*, regardless of whether or not  
3217 they appeared in the original *decision request*, and are referred to in the *policy* by attribute designators  
3218 and attribute selectors. A *named attribute* is the term used for the criteria that the specific attribute  
3219 designators use to refer to particular *attributes* in the <Attributes> elements of the request *context*.

### 3220 7.3.1 Structured attributes

3221 <AttributeValue> elements MAY contain an instance of a structured XML data-type, for example  
3222 <ds:KeyInfo>. XACML 3.0 supports several ways for comparing the contents of such elements.

- 3223 1. In some cases, such elements MAY be compared using one of the XACML string functions, such  
3224 as "string-regexp-match", described below. This requires that the element be given the data-type  
3225 "http://www.w3.org/2001/XMLSchema#string". For example, a structured data-type that is  
3226 actually a ds:KeyInfo/KeyName would appear in the *Context* as:

```
3227 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">  
3228 <lt;ds:KeyName>jhibbert-key</ds:KeyName>  
3229 </AttributeValue>
```

3230 In general, this method will not be adequate unless the structured data-type is quite simple.

- 3231 2. The structured *attribute* MAY be made available in the <Content> element of the appropriate  
3232 *attribute* category and an <AttributeSelector> element MAY be used to select the contents  
3233 of a leaf sub-element of the structured data-type by means of an XPath expression. That value  
3234 MAY then be compared using one of the supported XACML functions appropriate for its primitive  
3235 data-type. This method requires support by the *PDP* for the optional XPath expressions feature.
- 3236 3. 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 any node in  
3238 the structured data-type by means of an XPath expression. This node MAY then be compared  
3239 using one of the XPath-based functions described in Section A.3.15. This method requires  
3240 support by the *PDP* for the optional XPath expressions and XPath functions features.

3241 See also Section 7.3.

### 3242 7.3.2 Attribute bags

3243 XACML defines implicit collections of its data-types. XACML refers to a collection of values that are of a  
3244 single data-type as a *bag*. *Bags* of data-types are needed because selections of nodes from an XML  
3245 *resource* or XACML request *context* may return more than one value.

3246 The <AttributeSelector> element uses an XPath expression to specify the selection of data from  
3247 free form XML. The result of an XPath expression is termed a node-set, which contains all the nodes  
3248 from the XML content that match the *predicate* in the XPath expression. Based on the various indexing  
3249 functions provided in the XPath specification, it SHALL be implied that a resultant node-set is the  
3250 collection of the matching nodes. XACML also defines the <AttributeDesignator> element to have  
3251 the same matching methodology for *attributes* in the XACML request *context*.

3252 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be no  
3253 notion of a **bag** containing **bags**, or a **bag** containing values of differing types; i.e., a **bag** in XACML  
3254 SHALL contain only values that are of the same data-type.

### 3255 7.3.3 Multivalued attributes

3256 If a single <Attribute> element in a request **context** contains multiple <AttributeValue> child  
3257 elements, then the **bag** of values resulting from evaluation of the <Attribute> element MUST be  
3258 identical to the **bag** of values that results from evaluating a **context** in which each <AttributeValue>  
3259 element appears in a separate <Attribute> element, each carrying identical meta-data.

### 3260 7.3.4 Attribute Matching

3261 A **named attribute** includes specific criteria with which to match **attributes** in the **context**. An **attribute**  
3262 specifies a **Category**, **AttributeId** and **DataType**, and a **named attribute** also specifies the  
3263 **Issuer**. A **named attribute** SHALL match an **attribute** if the values of their respective **Category**,  
3264 **AttributeId**, **DataType** and optional **Issuer** attributes match. The **Category** of the **named**  
3265 **attribute** MUST match, by URI equality, the **Category** of the corresponding **context attribute**. The  
3266 **AttributeId** of the **named attribute** MUST match, by URI equality, the **AttributeId** of the  
3267 corresponding **context attribute**. The **DataType** of the **named attribute** MUST match, by URI equality,  
3268 the **DataType** of the corresponding **context attribute**. If **Issuer** is supplied in the **named attribute**,  
3269 then it MUST match, using the urn:oasis:names:tc:xacml:1.0:function:string-equal function, the **Issuer** of  
3270 the corresponding **context attribute**. If **Issuer** is not supplied in the **named attribute**, then the  
3271 matching of the **context attribute** to the **named attribute** SHALL be governed by **AttributeId** and  
3272 **DataType** alone, regardless of the presence, absence, or actual value of **Issuer** in the corresponding  
3273 **context attribute**. In the case of an attribute selector, the matching of the **attribute** to the **named**  
3274 **attribute** SHALL be governed by the XPath expression and **DataType**.

### 3275 7.3.5 Attribute Retrieval

3276 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**. The  
3277 **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document, but the  
3278 **context handler** is responsible for obtaining and supplying the requested values by whatever means it  
3279 deems appropriate. The **context handler** SHALL return the values of **attributes** that match the attribute  
3280 designator or attribute selector and form them into a **bag** of values with the specified data-type. If no  
3281 **attributes** from the request **context** match, then the **attribute** SHALL be considered missing. If the  
3282 **attribute** is missing, then **MustBePresent** governs whether the attribute designator or attribute selector  
3283 returns an empty **bag** or an "Indeterminate" result. If **MustBePresent** is "False" (default value), then a  
3284 missing **attribute** SHALL result in an empty **bag**. If **MustBePresent** is "True", then a missing **attribute**  
3285 SHALL result in "Indeterminate". This "Indeterminate" result SHALL be handled in accordance with the  
3286 specification of the encompassing expressions, **rules**, **policies** and **policy sets**. If the result is  
3287 "Indeterminate", then the **AttributeId**, **DataType** and **Issuer** of the **attribute** MAY be listed in the  
3288 **authorization decision** as described in Section 7.15. However, a **PDP** MAY choose not to return such  
3289 information for security reasons.

### 3290 7.3.6 Environment Attributes

3291 Standard **environment attributes** are listed in Section B.7. If a value for one of these **attributes** is  
3292 supplied in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the  
3293 **context handler** SHALL supply a value. In the case of date and time **attributes**, the supplied value  
3294 SHALL have the semantics of the "date and time that apply to the **decision request**".

### 3295 7.3.7 AttributeSelector evaluation

3296 An <AttributeSelector> element will be evaluated according to the following processing model.

3297

3298 NOTE: It is not necessary for an implementation to actually follow these steps. It is only  
3299 necessary to produce results identical to those that would be produced by following these  
3300 steps.

- 3301 1. Construct an XML data structure suitable for xpath processing from the <Content> element in  
3302 the **attributes** category given by the *Category* attribute. The data structure shall be constructed  
3303 so that the document node of this structure contains a single document element which  
3304 corresponds to the single child element of the <Content> element. The constructed data  
3305 structure shall be equivalent to one that would result from parsing a stand-alone XML document  
3306 consisting of the contents of the <Content> element (including any comment and processing-  
3307 instruction markup). Namespace declarations which are not “visibly utilized”, as defined by [exc-  
3308 c14n], MAY not be present and MUST NOT be utilized by the XPath expression in step 3. The  
3309 data structure must meet the requirements of the applicable xpath version.
- 3310 2. Select a context node for xpath processing from this data structure. If there is a  
3311 *ContextSelectorId* attribute, the context node shall be the node selected by applying the  
3312 XPath expression given in the **attribute** value of the designated **attribute** (in the **attributes**  
3313 category given by the <AttributeSelector> *Category* attribute). It shall be an error if this  
3314 evaluation returns no node or more than one node, in which case the return value MUST be an  
3315 “Indeterminate” with a status code “urn:oasis:names:tc:xacml:1.0:status:syntax-error”. If there is  
3316 no *ContextSelectorId*, the document node of the data structure shall be the context node.
- 3317 3. Evaluate the XPath expression given in the *Path* attribute against the xml data structure, using  
3318 the context node selected in the previous step. It shall be an error if this evaluation returns  
3319 anything other than a sequence of nodes (possibly empty), in which case the  
3320 <AttributeSelector> MUST return “Indeterminate” with a status code  
3321 “urn:oasis:names:tc:xacml:1.0:status:syntax-error”.
- 3322 4. If the data type is a primitive data type, convert the text value of each selected node to the  
3323 desired data type, as specified in the *DataType* attribute. Each value shall be constructed using  
3324 the appropriate constructor function from [XF] Section 5 listed below, corresponding to the  
3325 specified data type.

3326  
3327 xs:string()  
3328 xs:boolean()  
3329 xs:integer()  
3330 xs:double()  
3331 xs:dateTime()  
3332 xs:date()  
3333 xs:time()  
3334 xs:hexBinary()  
3335 xs:base64Binary()  
3336 xs:anyURI()  
3337 xs:yearMonthDuration()  
3338 xs:dayTimeDuration()  
3339

3340 If the *DataType* is not one of the primitive types listed above, then the return values shall be  
3341 constructed from the nodeset in a manner specified by the of the particular *DataType* extension  
3342 specification. If the data type extension does not specify an appropriate constructor function, then  
3343 the <AttributeSelector> MUST return “Indeterminate” with a status code  
3344 “urn:oasis:names:tc:xacml:1.0:status:syntax-error”.

3345  
3346 If an error occurs when converting the values returned by the XPath expression to the specified  
3347 *DataType*, then the result of the <AttributeSelector> MUST be “Indeterminate”, with a  
3348 status code “urn:oasis:names:tc:xacml:1.0:status:processing-error”

## 3349 7.4 Expression evaluation

3350 XACML specifies expressions in terms of the elements listed below, of which the <Apply> and  
3351 <Condition> elements recursively compose greater expressions. Valid expressions SHALL be type  
3352 correct, which means that the types of each of the elements contained within <Apply> elements SHALL  
3353 agree with the respective argument types of the function that is named by the `FunctionId` attribute.  
3354 The resultant type of the <Apply> element SHALL be the resultant type of the function, which MAY be  
3355 narrowed to a primitive data-type, or a **bag** of a primitive data-type, by type-unification. XACML defines  
3356 an evaluation result of "Indeterminate", which is said to be the result of an invalid expression, or an  
3357 operational error occurring during the evaluation of the expression.

3358 XACML defines these elements to be in the substitution group of the <Expression> element:

- 3359 • <xacml:AttributeValue>
- 3360 • <xacml:AttributeDesignator>
- 3361 • <xacml:AttributeSelector>
- 3362 • <xacml:Apply>
- 3363 • <xacml:Condition>
- 3364 • <xacml:Function>
- 3365 • <xacml:VariableReference>

## 3366 7.5 Arithmetic evaluation

3367 IEEE 754 [IEEE754] specifies how to evaluate arithmetic functions in a context, which specifies defaults  
3368 for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all integer and  
3369 double functions relying on the Extended Default Context, enhanced with double precision:

3370 flags - all set to 0

3371 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap enabler,  
3372 which SHALL be set to 1

3373 precision - is set to the designated double precision

3374 rounding - is set to round-half-even (IEEE 854 §4.1)

## 3375 7.6 Match evaluation

3376 The **attribute** matching element <Match> appears in the <Target> element of **rules**, **policies** and  
3377 **policy sets**.

3378 This element represents a Boolean expression over **attributes** of the request **context**. A matching  
3379 element contains a `MatchId` attribute that specifies the function to be used in performing the match  
3380 evaluation, an <AttributeValue> and an <AttributeDesignator> or <AttributeSelector>  
3381 element that specifies the **attribute** in the **context** that is to be matched against the specified value.

3382 The `MatchId` attribute SHALL specify a function that takes two arguments, returning a result type of  
3383 "<http://www.w3.org/2001/XMLSchema#boolean>". The **attribute** value specified in the matching element  
3384 SHALL be supplied to the `MatchId` function as its first argument. An element of the **bag** returned by the  
3385 <AttributeDesignator> or <AttributeSelector> element SHALL be supplied to the `MatchId`  
3386 function as its second argument, as explained below. The `DataType` of the <AttributeValue>  
3387 SHALL match the data-type of the first argument expected by the `MatchId` function. The `DataType` of  
3388 the <AttributeDesignator> or <AttributeSelector> element SHALL match the data-type of the  
3389 second argument expected by the `MatchId` function.

3390 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the  
3391 `MatchId` attribute, and those functions MAY use data-types that are also extensions, so long as the  
3392 extension function returns a Boolean result and takes two single base types as its inputs. The function

3393 used as the value for the `MatchId` attribute SHOULD be easily indexable. Use of non-indexable or  
 3394 complex functions may prevent efficient evaluation of **decision requests**.

3395 The evaluation semantics for a matching element is as follows. If an operational error were to occur while  
 3396 evaluating the `<AttributeDesignator>` or `<AttributeSelector>` element, then the result of the  
 3397 entire expression SHALL be "Indeterminate". If the `<AttributeDesignator>` or  
 3398 `<AttributeSelector>` element were to evaluate to an empty **bag**, then the result of the expression  
 3399 SHALL be "False". Otherwise, the `MatchId` function SHALL be applied between the  
 3400 `<AttributeValue>` and each element of the **bag** returned from the `<AttributeDesignator>` or  
 3401 `<AttributeSelector>` element. If at least one of those function applications were to evaluate to  
 3402 "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the function  
 3403 applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally, if all function  
 3404 applications evaluate to "False", then the result of the entire expression SHALL be "False".

3405 It is also possible to express the semantics of a **target** matching element in a **condition**. For instance,  
 3406 the **target** match expression that compares a "**subject-name**" starting with the name "John" can be  
 3407 expressed as follows:

```

3408 <Match
3409   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
3410     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
3411       John.*
3412     </AttributeValue>
3413     <AttributeDesignator
3414       Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
3415 subject"
3416       AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
3417       DataType="http://www.w3.org/2001/XMLSchema#string"/>
3418   </Match>
  
```

3419 Alternatively, the same match semantics can be expressed as an `<Apply>` element in a **condition** by  
 3420 using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```

3421 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
3422   <Function
3423     FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match"/>
3424   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
3425     John.*
3426   </AttributeValue>
3427   <AttributeDesignator
3428     Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
3429 subject"
3430     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
3431     DataType="http://www.w3.org/2001/XMLSchema#string"/>
3432 </Apply>
  
```

## 3433 7.7 Target evaluation

3434 An empty **target** matches any request. Otherwise the **target** value SHALL be "Match" if all the AnyOf  
 3435 specified in the **target** match values in the request **context**. Otherwise, if any one of the AnyOf specified  
 3436 in the **target** is "No Match", then the **target** SHALL be "No Match". Otherwise, the **target** SHALL be  
 3437 "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"

3438 Table 1 Target match table

3439 The AnyOf SHALL match values in the request **context** if at least one of their <AllOf> elements  
 3440 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"

3441 Table 2 AnyOf match table

3442 An AllOf SHALL match a value in the request **context** if the value of all its <Match> elements is "True".  
 3443 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"

3444 Table 3 AllOf match table

3445 **7.8 VariableReference Evaluation**

3446 The <VariableReference> element references a single <VariableDefinition> element contained  
 3447 within the same <Policy> element. A <VariableReference> that does not reference a particular  
 3448 <VariableDefinition> element within the encompassing <Policy> element is called an undefined  
 3449 reference. **Policies** with undefined references are invalid.

3450 In any place where a <VariableReference> occurs, it has the effect as if the text of the  
 3451 <Expression> element defined in the <VariableDefinition> element replaces the  
 3452 <VariableReference> element. Any evaluation scheme that preserves this semantic is acceptable.  
 3453 For instance, the expression in the <VariableDefinition> element may be evaluated to a particular  
 3454 value and cached for multiple references without consequence. (I.e. the value of an <Expression>  
 3455 element remains the same for the entire **policy** evaluation.) This characteristic is one of the benefits of  
 3456 XACML being a declarative language.

3457 A variable reference containing circular references is invalid. The PDP MUST detect circular references  
 3458 either at policy loading time or during runtime evaluation. If the PDP detects a circular reference during  
 3459 runtime the variable reference evaluates to "Indeterminate" with status code  
 3460 urn:oasis:names:tc:xacml:1.0:status:processing-error.

3461 **7.9 Condition evaluation**

3462 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to "True".  
 3463 Its value SHALL be "False" if the <Condition> element evaluates to "False". The **condition** value  
 3464 SHALL be "Indeterminate", if the expression contained in the <Condition> element evaluates to  
 3465 "Indeterminate."

3466 **7.10 Rule evaluation**

3467 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves separate  
 3468 evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 4.

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

3469 *Table 4 Rule truth table.*

3470 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable" or  
 3471 "Indeterminate", respectively, regardless of the value of the **condition**. For these cases, therefore, the  
 3472 **condition** need not be evaluated.

3473 If the **target** value is "Match", or there is no **target** in the **rule**, and the **condition** value is "True", then the  
 3474 **effect** specified in the enclosing <Rule> element SHALL determine the **rule's** value.

## 3475 7.11 Policy evaluation

3476 The value of a **policy** SHALL be determined only by its contents, considered in relation to the contents of  
 3477 the request **context**. A **policy's** value SHALL be determined by evaluation of the **policy's target** and  
 3478 **rules**.

3479 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**  
 3480 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the **policy's**  
 3481 **rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to "No-match", then  
 3482 the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to "Indeterminate", then the  
 3483 value of the **policy** SHALL be "Indeterminate".

3484 The **policy** truth table is shown in Table 5.

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

3485 *Table 5 Policy truth table*

3486 A **rules** value of "At least one **rule** value is its **Effect**" means either that the <Rule> element is absent, or  
 3487 one or more of the **rules** contained in the **policy** is applicable to the **decision request** (i.e., it returns the  
 3488 value of its "**Effect**"; see Section 7.10). A **rules** value of "All **rule** values are 'NotApplicable'" SHALL be  
 3489 used if no **rule** contained in the **policy** is applicable to the request and if no **rule** contained in the **policy**  
 3490 returns a value of "Indeterminate". If no **rule** contained in the **policy** is applicable to the request, but one  
 3491 or more **rule** returns a value of "Indeterminate", then the **rules** SHALL evaluate to "At least one **rule** value  
 3492 is 'Indeterminate'".

3493 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be "NotApplicable" or  
 3494 "Indeterminate", respectively, regardless of the value of the **rules**. For these cases, therefore, the **rules**  
 3495 need not be evaluated.

3496 If the **target** value is "Match" and the **rule** value is "At least one **rule** value is its **Effect**" or "At least one  
 3497 **rule** value is 'Indeterminate'", then the **rule-combining algorithm** specified in the **policy** SHALL  
 3498 determine the **policy** value.

3499 Note that none of the **rule-combining algorithms** defined by XACML 3.0 take parameters. However,  
 3500 non-standard combining algorithms MAY take parameters. In such a case, the values of these  
 3501 parameters associated with the **rules**, MUST be taken into account when evaluating the **policy**. The  
 3502 parameters and their types should be defined in the specification of the combining algorithm. If the  
 3503 implementation supports combiner parameters and if combiner parameters are present in a **policy**, then  
 3504 the parameter values MUST be supplied to the combining algorithm implementation.

## 3505 7.12 Policy Set evaluation

3506 The value of a **policy set** SHALL be determined by its contents, considered in relation to the contents of  
 3507 the request **context**. A **policy set**'s value SHALL be determined by evaluation of the **policy set**'s **target**,  
 3508 **policies**, and **policy sets**, according to the specified **policy-combining algorithm**.

3509 The **policy set**'s **target** SHALL be evaluated to determine the applicability of the **policy set**. If the **target**  
 3510 evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of the **policy**  
 3511 **set**'s **policies** and **policy sets**, according to the specified **policy-combining algorithm**. If the **target**  
 3512 evaluates to "No-match", then the value of the **policy set** shall be "NotApplicable". If the **target** evaluates  
 3513 to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

3514 The **policy set** truth table is shown in Table 6.

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

3515 Table 6 Policy set truth table

3516 A **policies** value of "At least one **policy** value is its Decision" SHALL be used if there are no contained or  
 3517 referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets** contained in or  
 3518 referenced by the **policy set** is applicable to the **decision request** (i.e., returns a value determined by its  
 3519 combining algorithm) A **policies** value of "All **policy** values are 'NotApplicable'" SHALL be used if no  
 3520 **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request and if no  
 3521 **policy** or **policy set** contained in or referenced by the **policy set** returns a value of "Indeterminate". If no  
 3522 **policy** or **policy set** contained in or referenced by the **policy set** is applicable to the request but one or  
 3523 more **policy** or **policy set** returns a value of "Indeterminate", then the **policies** SHALL evaluate to "At  
 3524 least one **policy** value is 'Indeterminate'".

3525 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be "NotApplicable"  
 3526 or "Indeterminate", respectively, regardless of the value of the **policies**. For these cases, therefore, the  
 3527 **policies** need not be evaluated.

3528 If the **target** value is “Match” and the **policies** value is “At least one **policy** value is its Decision” or “At  
3529 least one **policy** value is ‘Indeterminate’”, then the **policy-combining algorithm** specified in the **policy**  
3530 **set** SHALL determine the **policy set** value.

3531 Note that none of the **policy-combining algorithms** defined by XACML 3.0 take parameters. However,  
3532 non-standard combining algorithms MAY take parameters. In such a case, the values of these  
3533 parameters associated with the **policies**, MUST be taken into account when evaluating the **policy set**.  
3534 The parameters and their types should be defined in the specification of the combining algorithm. If the  
3535 implementation supports combiner parameters and if combiner parameters are present in a **policy**, then  
3536 the parameter values MUST be supplied to the combining algorithm implementation.

## 3537 7.13 PolicySetIdReference and PolicyIdReference evaluation

3538 A policy set id reference or a policy id reference is evaluated by resolving the reference and evaluating  
3539 the referenced policy set or policy.

3540 If resolving the reference fails, the reference evaluates to “Indeterminate” with status code  
3541 urn:oasis:names:tc:xacml:1.0:status:processing-error.

3542 A policy set id reference or a policy id reference containing circular references is invalid. The PDP MUST  
3543 detect circular references either at policy loading time or during runtime evaluation. If the PDP detects a  
3544 circular reference during runtime the reference evaluates to “Indeterminate” with status code  
3545 urn:oasis:names:tc:xacml:1.0:status:processing-error.

## 3546 7.14 Hierarchical resources

3547 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document). XACML  
3548 provides several optional mechanisms for supporting hierarchical **resources**. These are described in the  
3549 XACML Profile for Hierarchical Resources [**Hier**] and in the XACML Profile for Requests for Multiple  
3550 Resources [**Multi**].

## 3551 7.15 Authorization decision

3552 In relation to a particular **decision request**, the **PDP** is defined by a **policy-combining algorithm** and a  
3553 set of **policies** and/or **policy sets**. The **PDP** SHALL return a response **context** as if it had evaluated a  
3554 single **policy set** consisting of this **policy-combining algorithm** and the set of **policies** and/or **policy**  
3555 **sets**.

3556 The **PDP** MUST evaluate the **policy set** as specified in Sections 5 and 7. The **PDP** MUST return a  
3557 response **context**, with one <Decision> element of value "Permit", "Deny", "Indeterminate" or  
3558 "NotApplicable".

3559 If the **PDP** cannot make a **decision**, then an "Indeterminate" <Decision> element SHALL be returned.

## 3560 7.16 Obligations and advice

3561 A **rule**, **policy**, or **policy set** may contain one or more **obligation** or **advice** expressions. When such a  
3562 **rule**, **policy**, or **policy set** is evaluated, the **obligation** or **advice** expression SHALL be evaluated to an  
3563 **obligation** or **advice** respectively, which SHALL be passed up to the next level of evaluation (the  
3564 enclosing or referencing **policy**, **policy set**, or **authorization decision**) only if the **effect** of the **rule**,  
3565 **policy**, or **policy set** being evaluated matches the value of the FulfillOn attribute of the **obligation** or  
3566 the AppliesTo attribute of the **advice**. If any of the **attribute** assignment expressions in an **obligation**  
3567 or **advice** expression with a matching FulfillOn or AppliesTo attribute evaluates to “Indeterminate”,  
3568 then the whole **rule**, **policy**, or **policy set** SHALL be “Indeterminate”. If the FulfillOn or AppliesTo  
3569 attribute does not match the result of the combining algorithm or the **rule** evaluation, then any  
3570 indeterminate in an **obligation** or **advice** expression has no effect.

3571 As a consequence of this procedure, no **obligations** or **advice** SHALL be returned to the **PEP** if the **rule**,  
3572 **policies**, or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is  
3573 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **rule**, **policy**, or **policy**  
3574 **set** does not match the **decision** resulting from evaluating an enclosing **policy set**.

3575 If the *PDP*'s evaluation is viewed as a tree of *rules*, *policy sets* and *policies*, each of which returns  
3576 "Permit" or "Deny", then the set of *obligations* and *advice* returned by the *PDP* to the *PEP* will include  
3577 only the *obligations* and *advice* associated with those paths where the *effect* at each level of evaluation  
3578 is the same as the *effect* being returned by the *PDP*. In situations where any lack of determinism is  
3579 unacceptable, a deterministic combining algorithm, such as ordered-deny-overrides, should be used.  
3580 Also see Section 7.2.

## 3581 7.17 Exception handling

3582 XACML specifies behavior for the *PDP* in the following situations.

### 3583 7.17.1 Unsupported functionality

3584 If the *PDP* attempts to evaluate a *policy set* or *policy* that contains an optional element type or function  
3585 that the *PDP* does not support, then the *PDP* SHALL return a <Decision> value of "Indeterminate". If a  
3586 <StatusCode> element is also returned, then its value SHALL be  
3587 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and  
3588 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported function.

### 3589 7.17.2 Syntax and type errors

3590 If a *policy* that contains invalid syntax is evaluated by the XACML *PDP* at the time a *decision request* is  
3591 received, then the result of that *policy* SHALL be "Indeterminate" with a *StatusCode* value of  
3592 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3593 If a *policy* that contains invalid static data-types is evaluated by the XACML *PDP* at the time a *decision*  
3594 *request* is received, then the result of that *policy* SHALL be "Indeterminate" with a *StatusCode* value of  
3595 "urn:oasis:names:tc:xacml:1.0:status:processing-error".

### 3596 7.17.3 Missing attributes

3597 The absence of matching *attributes* in the request *context* for any of the attribute designators attribute or  
3598 selectors that are found in the *policy* will result in an enclosing <AllOf> element to return a value of  
3599 "Indeterminate", if the designator or selector has the *MustBePresent* XML attribute set to true, as  
3600 described in Sections 5.29 and 5.30 and may result in a <Decision> element containing the  
3601 "Indeterminate" value. If, in this case, and a status code is supplied, then the value

3602 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute"

3603 SHALL be used, to indicate that more information is needed in order for a definitive *decision* to be  
3604 rendered. In this case, the <Status> element MAY list the names and data-types of any *attributes* that  
3605 are needed by the *PDP* to refine its *decision* (see Section 5.58). A *PEP* MAY resubmit a refined request  
3606 *context* in response to a <Decision> element contents of "Indeterminate" with a status code of

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

3608 by adding *attribute* values for the *attribute* names that were listed in the previous response. When the  
3609 *PDP* returns a <Decision> element contents of "Indeterminate", with a status code of

3610 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute",

3611 it MUST NOT list the names and data-types of any *attribute* for which values were supplied in the original  
3612 request. Note, this requirement forces the *PDP* to eventually return an *authorization decision* of  
3613 "Permit", "Deny", or "Indeterminate" with some other status code, in response to successively-refined  
3614 requests.

---

## 3615 8 XACML extensibility points (non-normative)

3616 This section describes the points within the XACML model and schema where extensions can be added.

### 3617 8.1 Extensible XML attribute types

3618 The following XML attributes have values that are URIs. These may be extended by the creation of new  
3619 URIs associated with new semantics for these attributes.

3620 `Category,`

3621 `AttributeId,`

3622 `DataType,`

3623 `FunctionId,`

3624 `MatchId,`

3625 `ObligationId,`

3626 `AdviceId,`

3627 `PolicyCombiningAlgId,`

3628 `RuleCombiningAlgId,`

3629 `StatusCode,`

3630 `SubjectCategory.`

3631 See Section 5 for definitions of these *attribute* types.

### 3632 8.2 Structured attributes

3633 `<AttributeValue>` elements MAY contain an instance of a structured XML data-type. Section 7.3.1  
3634 describes a number of standard techniques to identify data items within such a structured *attribute*.  
3635 Listed here are some additional techniques that require XACML extensions.

- 3636 1. For a given structured data-type, a community of XACML users MAY define new *attribute*  
3637 identifiers for each leaf sub-element of the structured data-type that has a type conformant with  
3638 one of the XACML-defined primitive data-types. Using these new *attribute* identifiers, the *PEPs*  
3639 or *context handlers* used by that community of users can flatten instances of the structured  
3640 data-type into a sequence of individual `<Attribute>` elements. Each such `<Attribute>`  
3641 element can be compared using the XACML-defined functions. Using this method, the structured  
3642 data-type itself never appears in an `<AttributeValue>` element.
- 3643 2. A community of XACML users MAY define a new function that can be used to compare a value of  
3644 the structured data-type against some other value. This method may only be used by *PDPs* that  
3645 support the new function.

---

## 3646 9 Security and privacy considerations (non- 3647 normative)

3648 This section identifies possible security and privacy compromise scenarios that should be considered  
3649 when implementing an XACML-based system. The section is informative only. It is left to the  
3650 implementer to decide whether these compromise scenarios are practical in their environment and to  
3651 select appropriate safeguards.

### 3652 9.1 Threat model

3653 We assume here that the adversary has access to the communication channel between the XACML  
3654 actors and is able to interpret, insert, delete, and modify messages or parts of messages.

3655 Additionally, an actor may use information from a former message maliciously in subsequent transactions.  
3656 It is further assumed that **rules** and **policies** are only as reliable as the actors that create and use them.  
3657 Thus it is incumbent on each actor to establish appropriate trust in the other actors upon which it relies.  
3658 Mechanisms for trust establishment are outside the scope of this specification.

3659 The messages that are transmitted between the actors in the XACML model are susceptible to attack by  
3660 malicious third parties. Other points of vulnerability include the **PEP**, the **PDP**, and the **PAP**. While some  
3661 of these entities are not strictly within the scope of this specification, their compromise could lead to the  
3662 compromise of **access control** enforced by the **PEP**.

3663 It should be noted that there are other components of a distributed system that may be compromised,  
3664 such as an operating system and the domain-name system (DNS) that are outside the scope of this  
3665 discussion of threat models. Compromise in these components may also lead to a policy violation.

3666 The following sections detail specific compromise scenarios that may be relevant to an XACML system.

#### 3667 9.1.1 Unauthorized disclosure

3668 XACML does not specify any inherent mechanisms to protect the confidentiality of the messages  
3669 exchanged between actors. Therefore, an adversary could observe the messages in transit. Under  
3670 certain security **policies**, disclosure of this information is a violation. Disclosure of **attributes** or the types  
3671 of **decision requests** that a **subject** submits may be a breach of privacy policy. In the commercial  
3672 sector, the consequences of unauthorized disclosure of personal data may range from embarrassment to  
3673 the custodian, to imprisonment and/or large fines in the case of medical or financial data.

3674 Unauthorized disclosure is addressed by confidentiality safeguards.

#### 3675 9.1.2 Message replay

3676 A message replay attack is one in which the adversary records and replays legitimate messages between  
3677 XACML actors. This attack may lead to denial of service, the use of out-of-date information or  
3678 impersonation.

3679 Prevention of replay attacks requires the use of message freshness safeguards.

3680 Note that encryption of the message does not mitigate a replay attack since the message is simply  
3681 replayed and does not have to be understood by the adversary.

#### 3682 9.1.3 Message insertion

3683 A message insertion attack is one in which the adversary inserts messages in the sequence of messages  
3684 between XACML actors.

3685 The solution to a message insertion attack is to use mutual authentication and message sequence  
3686 integrity safeguards between the actors. It should be noted that just using SSL mutual authentication is  
3687 not sufficient. This only proves that the other party is the one identified by the **subject** of the X.509

3688 certificate. In order to be effective, it is necessary to confirm that the certificate **subject** is authorized to  
3689 send the message.

#### 3690 9.1.4 Message deletion

3691 A message deletion attack is one in which the adversary deletes messages in the sequence of messages  
3692 between XACML actors. Message deletion may lead to denial of service. However, a properly designed  
3693 XACML system should not render an incorrect **authorization decision** as a result of a message deletion  
3694 attack.

3695 The solution to a message deletion attack is to use message sequence integrity safeguards between the  
3696 actors.

#### 3697 9.1.5 Message modification

3698 If an adversary can intercept a message and change its contents, then they may be able to alter an  
3699 **authorization decision**. A message integrity safeguard can prevent a successful message modification  
3700 attack.

#### 3701 9.1.6 NotApplicable results

3702 A result of "NotApplicable" means that the **PDP** could not locate a **policy** whose **target** matched the  
3703 information in the **decision request**. In general, it is highly recommended that a "Deny" **effect policy** be  
3704 used, so that when a **PDP** would have returned "NotApplicable", a result of "Deny" is returned instead.

3705 In some security models, however, such as those found in many web servers, an **authorization decision**  
3706 of "NotApplicable" is treated as equivalent to "Permit". There are particular security considerations that  
3707 must be taken into account for this to be safe. These are explained in the following paragraphs.

3708 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the **policy** to  
3709 match elements in the **decision request** be closely aligned with the data syntax used by the applications  
3710 that will be submitting the **decision request**. A failure to match will result in "NotApplicable" and be  
3711 treated as "Permit". So an unintended failure to match may allow unintended **access**.

3712 Commercial http responders allow a variety of syntaxes to be treated equivalently. The "%" can be used  
3713 to represent characters by hex value. The URL path "/./" provides multiple ways of specifying the same  
3714 value. Multiple character sets may be permitted and, in some cases, the same printed character can be  
3715 represented by different binary values. Unless the matching algorithm used by the **policy** is sophisticated  
3716 enough to catch these variations, unintended **access** may be permitted.

3717 It may be safe to treat "NotApplicable" as "Permit" only in a closed environment where all applications that  
3718 formulate a **decision request** can be guaranteed to use the exact syntax expected by the **policies**. In a  
3719 more open environment, where **decision requests** may be received from applications that use any legal  
3720 syntax, it is strongly recommended that "NotApplicable" NOT be treated as "Permit" unless matching  
3721 **rules** have been very carefully designed to match all possible applicable inputs, regardless of syntax or  
3722 type variations. Note, however, that according to Section 7.2, a **PEP** must deny **access** unless it  
3723 receives an explicit "Permit" **authorization decision**.

#### 3724 9.1.7 Negative rules

3725 A negative **rule** is one that is based on a **predicate** not being "True". If not used with care, negative  
3726 **rules** can lead to policy violations, therefore some authorities recommend that they not be used.  
3727 However, negative **rules** can be extremely efficient in certain cases, so XACML has chosen to include  
3728 them. Nevertheless, it is recommended that they be used with care and avoided if possible.

3729 A common use for negative **rules** is to deny **access** to an individual or subgroup when their membership  
3730 in a larger group would otherwise permit them **access**. For example, we might want to write a **rule** that  
3731 allows all vice presidents to see the unpublished financial data, except for Joe, who is only a ceremonial  
3732 vice president and can be indiscreet in his communications. If we have complete control over the  
3733 administration of **subject attributes**, a superior approach would be to define "Vice President" and  
3734 "Ceremonial Vice President" as distinct groups and then define **rules** accordingly. However, in some

3735 environments this approach may not be feasible. (It is worth noting in passing that referring to individuals  
3736 in **rules** does not scale well. Generally, shared **attributes** are preferred.)

3737 If not used with care, negative **rules** can lead to policy violations in two common cases: when **attributes**  
3738 are suppressed and when the base group changes. An example of suppressed **attributes** would be if we  
3739 have a **policy** that **access** should be permitted, unless the **subject** is a credit risk. If it is possible that  
3740 the **attribute** of being a credit risk may be unknown to the **PDP** for some reason, then unauthorized  
3741 **access** may result. In some environments, the **subject** may be able to suppress the publication of  
3742 **attributes** by the application of privacy controls, or the server or repository that contains the information  
3743 may be unavailable for accidental or intentional reasons.

3744 An example of a changing base group would be if there is a **policy** that everyone in the engineering  
3745 department may change software source code, except for secretaries. Suppose now that the department  
3746 was to merge with another engineering department and the intent is to maintain the same **policy**.  
3747 However, the new department also includes individuals identified as administrative assistants, who ought  
3748 to be treated in the same way as secretaries. Unless the **policy** is altered, they will unintentionally be  
3749 permitted to change software source code. Problems of this type are easy to avoid when one individual  
3750 administers all **policies**, but when administration is distributed, as XACML allows, this type of situation  
3751 must be explicitly guarded against.

## 3752 9.1.8 Denial of service

3753 A denial of service attack is one in which the adversary overloads an XACML actor with excessive  
3754 computations or network traffic such that legitimate users cannot access the services provided by the  
3755 actor.

3756 The urn:oasis:names:tc:xacml:3.0:function:access-permitted function may lead to hard to predict behavior  
3757 in the **PDP**. It is possible that the function is invoked during the recursive invocations of the **PDP** such that  
3758 loops are formed. Such loops may in some cases lead to large numbers of requests to be generated  
3759 before the **PDP** can detect the loop and abort evaluation. Such loops could cause a denial of service at  
3760 the **PDP**, either because of a malicious **policy** or because of a mistake in a **policy**.

## 3761 9.2 Safeguards

### 3762 9.2.1 Authentication

3763 Authentication provides the means for one party in a transaction to determine the identity of the other  
3764 party in the transaction. Authentication may be in one direction, or it may be bilateral.

3765 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the  
3766 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an adversary  
3767 could provide false or invalid **authorization decisions**, leading to a policy violation.

3768 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust to  
3769 determine what, if any, sensitive data should be passed. One should keep in mind that even simple  
3770 "Permit" or "Deny" responses could be exploited if an adversary were allowed to make unlimited requests  
3771 to a **PDP**.

3772 Many different techniques may be used to provide authentication, such as co-located code, a private  
3773 network, a VPN, or digital signatures. Authentication may also be performed as part of the  
3774 communication protocol used to exchange the **contexts**. In this case, authentication may be performed  
3775 either at the message level or at the session level.

### 3776 9.2.2 Policy administration

3777 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects** may  
3778 use this information to determine how to gain unauthorized **access**.

3779 To prevent this threat, the repository used for the storage of **policies** may itself require **access control**.  
3780 In addition, the <Status> element should be used to return values of missing **attributes** only when  
3781 exposure of the identities of those **attributes** will not compromise security.

## 3782 9.2.3 Confidentiality

3783 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired  
3784 recipients and not by anyone else who encounters the message while it is in transit. There are two areas  
3785 in which confidentiality should be considered: one is confidentiality during transmission; the other is  
3786 confidentiality within a <Policy> element.

### 3787 9.2.3.1 Communication confidentiality

3788 In some environments it is deemed good practice to treat all data within an **access control** system as  
3789 confidential. In other environments, **policies** may be made freely available for distribution, inspection,  
3790 and audit. The idea behind keeping **policy** information secret is to make it more difficult for an adversary  
3791 to know what steps might be sufficient to obtain unauthorized **access**. Regardless of the approach  
3792 chosen, the security of the **access control** system should not depend on the secrecy of the **policy**.

3793 Any security considerations related to transmitting or exchanging XACML <Policy> elements are  
3794 outside the scope of the XACML standard. While it is important to ensure that the integrity and  
3795 confidentiality of <Policy> elements is maintained when they are exchanged between two parties, it is  
3796 left to the implementers to determine the appropriate mechanisms for their environment.

3797 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL. Using a  
3798 point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points is  
3799 compromised.

### 3800 9.2.3.2 Statement level confidentiality

3801 In some cases, an implementation may want to encrypt only parts of an XACML <Policy> element.

3802 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used to  
3803 encrypt all or parts of an XML document. This specification is recommended for use with XACML.

3804 It should go without saying that if a repository is used to facilitate the communication of cleartext (i.e.,  
3805 unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to store this  
3806 sensitive data.

## 3807 9.2.4 Policy integrity

3808 The XACML **policy** used by the **PDP** to evaluate the request **context** is the heart of the system.  
3809 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of the  
3810 **policy**. One is to ensure that <Policy> elements have not been altered since they were originally  
3811 created by the **PAP**. The other is to ensure that <Policy> elements have not been inserted or deleted  
3812 from the set of **policies**.

3813 In many cases, both aspects can be achieved by ensuring the integrity of the actors and implementing  
3814 session-level mechanisms to secure the communication between actors. The selection of the appropriate  
3815 mechanisms is left to the implementers. However, when **policy** is distributed between organizations to  
3816 be acted on at a later time, or when the **policy** travels with the protected **resource**, it would be useful to  
3817 sign the **policy**. In these cases, the XML Signature Syntax and Processing standard from W3C is  
3818 recommended to be used with XACML.

3819 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures should  
3820 not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not request a **policy**  
3821 based on who signed it or whether or not it has been signed (as such a basis for selection would, itself,  
3822 be a matter of policy). However, the **PDP** must verify that the key used to sign the **policy** is one  
3823 controlled by the purported **issuer** of the **policy**. The means to do this are dependent on the specific  
3824 signature technology chosen and are outside the scope of this document.

## 3825 9.2.5 Policy identifiers

3826 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure that  
3827 these are unique. Confusion between identifiers could lead to misidentification of the **applicable policy**.

3828 This specification is silent on whether a **PAP** must generate a new identifier when a **policy** is modified or  
3829 may use the same identifier in the modified **policy**. This is a matter of administrative practice. However,  
3830 care must be taken in either case. If the identifier is reused, there is a danger that other **policies** or  
3831 **policy sets** that reference it may be adversely affected. Conversely, if a new identifier is used, these  
3832 other **policies** may continue to use the prior **policy**, unless it is deleted. In either case the results may  
3833 not be what the **policy** administrator intends.

3834 If a **PDP** is provided with **policies** from distinct sources which might not be fully trusted, as in the use of  
3835 the administration profile [**XACMLAdmin**], there is a concern that someone could intentionally publish a  
3836 **policy** with an id which collides with another **policy**. This could cause **policy** references that point to the  
3837 wrong **policy**, and may cause other unintended consequences in an implementation which is predicated  
3838 upon having unique **policy** identifiers.

3839 If this issue is a concern it is RECOMMENDED that distinct **policy** issuers or sources are assigned  
3840 distinct namespaces for **policy** identifiers. One method is to make sure that the **policy** identifier begins  
3841 with a string which has been assigned to the particular **policy** issuer or source. The remainder of the  
3842 **policy** identifier is an issuer-specific unique part. For instance, Alice from Example Inc. could be assigned  
3843 the **policy** identifiers which begin with `http://example.com/xacml/policyId/alice/`. The **PDP** or another  
3844 trusted component can then verify that the authenticated source of the **policy** is Alice at Example Inc, or  
3845 otherwise reject the **policy**. Anyone else will be unable to publish **policies** with identifiers which collide  
3846 with the **policies** of Alice.

## 3847 9.2.6 Trust model

3848 Discussions of authentication, integrity and confidentiality safeguards necessarily assume an underlying  
3849 trust model: how can one actor come to believe that a given key is uniquely associated with a specific,  
3850 identified actor so that the key can be used to encrypt data for that actor or verify signatures (or other  
3851 integrity structures) from that actor? Many different types of trust models exist, including strict  
3852 hierarchies, distributed authorities, the Web, the bridge, and so on.

3853 It is worth considering the relationships between the various actors of the **access control** system in terms  
3854 of the interdependencies that do and do not exist.

- 3855 • None of the entities of the authorization system are dependent on the **PEP**. They may collect data  
3856 from it, (for example authentication data) but are responsible for verifying it themselves.
- 3857 • The correct operation of the system depends on the ability of the **PEP** to actually enforce **policy**  
3858 **decisions**.
- 3859 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the **PDP** is  
3860 supplied with the correct inputs. Other than that, the **PDP** does not depend on the **PEP**.
- 3861 • The **PDP** depends on the **PAP** to supply appropriate **policies**. The **PAP** is not dependent on other  
3862 components.

## 3863 9.2.7 Privacy

3864 It is important to be aware that any transactions that occur with respect to **access control** may reveal  
3865 private information about the actors. For example, if an XACML **policy** states that certain data may only  
3866 be read by **subjects** with "Gold Card Member" status, then any transaction in which a **subject** is  
3867 permitted **access** to that data leaks information to an adversary about the **subject's** status. Privacy  
3868 considerations may therefore lead to encryption and/or to **access control** requirements surrounding the  
3869 enforcement of XACML **policy** instances themselves: confidentiality-protected channels for the  
3870 request/response protocol messages, protection of **subject attributes** in storage and in transit, and so  
3871 on.

3872 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope of  
3873 XACML. The **decision** regarding whether, how, and when to deploy such mechanisms is left to the  
3874 implementers associated with the environment.

3875 **9.3 Unicode security issues**

3876 There are many security considerations related to use of Unicode. An XACML implementation SHOULD  
3877 follow the advice given in the relevant version of **[UTR36]**.

---

3878 **10 Conformance**

3879 **10.1 Introduction**

3880 The XACML specification addresses the following aspect of conformance:

3881 The XACML specification defines a number of functions, etc. that have somewhat special applications,  
3882 therefore they are not required to be implemented in an implementation that claims to conform with the  
3883 OASIS standard.

3884 **10.2 Conformance tables**

3885 This section lists those portions of the specification that **MUST** be included in an implementation of a **PDP**  
3886 that claims to conform to XACML v3.0. A set of test cases has been created to assist in this process.  
3887 These test cases can be located from the OASIS XACML TC Web page. The site hosting the test cases  
3888 contains a full description of the test cases and how to execute them.

3889 Note: "M" means mandatory-to-implement. "O" means optional.

3890 The implementation **MUST** follow sections 5, 6, 7, A, B and C where they apply to implemented items in  
3891 the following tables.

3892 **10.2.1 Schema elements**

3893 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

3894 **10.2.2 Identifier Prefixes**

3895 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

3896 **10.2.3 Algorithms**

3897 The implementation MUST include the **rule-** and **policy-combining algorithms** associated with the  
 3898 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

## 3899 10.2.4 Status Codes

3900 Implementation support for the <StatusCode> element is optional, but if the element is supported, then  
3901 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

## 3902 10.2.5 Attributes

3903 The implementation MUST support the **attributes** associated with the following identifiers as specified by  
3904 XACML. If values for these **attributes** are not present in the **decision request**, then their values MUST  
3905 be supplied by the **context handler**. So, unlike most other **attributes**, their semantics are not  
3906 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

## 3907 10.2.6 Identifiers

3908 The implementation MUST use the **attributes** associated with the following identifiers in the way XACML  
3909 has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that uses XACML,  
3910 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

3911 **10.2.7 Data-types**

3912 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

3913 **10.2.8 Functions**

3914 The implementation MUST properly process those functions associated with the identifiers marked with  
3915 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

3916 **10.2.9 Identifiers planned for future deprecation**

3917 These identifiers are associated with previous versions of XACML and newer alternatives exist in XACML  
3918 3.0. They are planned to be deprecated at some unspecified point in the future. It is RECOMMENDED  
3919 that these identifiers not be used in new policies and requests.

3920 The implementation MUST properly process those features associated with the identifiers marked with an  
3921 "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

3922

3923  
3924  
3925  
3926  
3927  
3928  
3929  
  
3930  
  
3931  
3932  
3933  
3934  
3935  
  
3936  
3937  
3938  
3939  
3940  
3941  
3942  
3943  
3944  
3945  
3946  
3947  
3948  
3949  
3950  
3951  
3952  
3953  
3954  
3955  
3956  
3957  
3958  
3959  
3960  
3961  
3962

---

## A. Data-types and functions (normative)

### A.1 Introduction

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

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

### A.2 Data-types

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

- <http://www.w3.org/2001/XMLSchema#string>
- <http://www.w3.org/2001/XMLSchema#boolean>
- <http://www.w3.org/2001/XMLSchema#integer>
- <http://www.w3.org/2001/XMLSchema#double>
- <http://www.w3.org/2001/XMLSchema#time>
- <http://www.w3.org/2001/XMLSchema#date>
- <http://www.w3.org/2001/XMLSchema#dateTime>
- <http://www.w3.org/2001/XMLSchema#anyURI>
- <http://www.w3.org/2001/XMLSchema#hexBinary>
- <http://www.w3.org/2001/XMLSchema#base64Binary>
- <http://www.w3.org/2001/XMLSchema#dayTimeDuration>
- <http://www.w3.org/2001/XMLSchema#yearMonthDuration>
- <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>
- <urn:oasis:names:tc:xacml:2.0:data-type:ipAddress>
- <urn:oasis:names:tc:xacml:2.0:data-type:dnsName>
- <urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression>

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

An XACML *PDP* SHALL be capable of converting string representations into various primitive data-types. For doubles, XACML SHALL use the conversions described in [IEEE754].

XACML defines four data-types representing identifiers for *subjects* or *resources*; these are:

- “urn:oasis:names:tc:xacml:1.0:data-type:x500Name”,
- “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”
- “urn:oasis:names:tc:xacml:2.0:data-type:ipAddress” and
- “urn:oasis:names:tc:xacml:2.0:data-type:dnsName”

These types appear in several standard applications, such as TLS/SSL and electronic mail.

#### X.500 directory name

3963 The "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" primitive type represents an ITU-T Rec.  
3964 X.520 Distinguished Name. The valid syntax for such a name is described in IETF RFC 2253  
3965 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished  
3966 Names".

#### 3967 **RFC 822 name**

3968 The "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" primitive type represents an electronic  
3969 mail address. The valid syntax for such a name is described in IETF RFC 2821, Section 4.1.2,  
3970 Command Argument Syntax, under the term "Mailbox".

#### 3971 **IP address**

3972 The "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" primitive type represents an IPv4 or IPv6  
3973 network address, with optional mask and optional port or port range. The syntax SHALL be:

3974 ipAddress = address [ "/" mask ] [ ":" [ portrange ] ]

3975 For an IPv4 address, the address and mask are formatted in accordance with the syntax for a  
3976 "host" in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", section 3.2.

3977 For an IPv6 address, the address and mask are formatted in accordance with the syntax for an  
3978 "ipv6reference" in IETF RFC 2732 "Format for Literal IPv6 Addresses in URL's". (Note that an  
3979 IPv6 address or mask, in this syntax, is enclosed in literal "[" "]" brackets.)

#### 3980 **DNS name**

3981 The "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" primitive type represents a Domain  
3982 Name Service (DNS) host name, with optional port or port range. The syntax SHALL be:

3983 dnsName = hostname [ ":" portrange ]

3984 The hostname is formatted in accordance with IETF RFC 2396 "Uniform Resource Identifiers  
3985 (URI): Generic Syntax", section 3.2, except that a wildcard "\*" may be used in the left-most  
3986 component of the hostname to indicate "any subdomain" under the domain specified to its right.

3987 For both the "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" and  
3988 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName" data-types, the port or port range syntax  
3989 SHALL be

3990 portrange = portnumber | "-"portnumber | portnumber "-" [portnumber]

3991 where "portnumber" is a decimal port number. If the port number is of the form "-x", where "x" is  
3992 a port number, then the range is all ports numbered "x" and below. If the port number is of the  
3993 form "x-", then the range is all ports numbered "x" and above. [This syntax is taken from the Java  
3994 SocketPermission.]

#### 3995 **XPath expression**

3996 The "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression" primitive type represents an  
3997 XPath expression selects over the XML in a <Content> element. The syntax is defined by the  
3998 XPath W3C recommendation. The content of this data type also includes the context in which  
3999 namespaces prefixes in the expression are resolved, which distinguishes it from a plain string and  
4000 the XACML **attribute** category of the <Content> element to which it applies. When the value is  
4001 encoded in an <AttributeValue> element, the namespace context is given by the  
4002 <AttributeValue> element and an XML attribute called XPathCategory gives the category of  
4003 the <Content> element where the expression applies.

4004 The XPath expression MUST be evaluated in a context which is equivalent of a stand alone XML  
4005 document with the only child of the <Content> element as the document element. Namespace  
4006 declarations which are not "visibly utilized", as defined by [exc-c14n], MAY not be present and  
4007 MUST NOT be utilized by the XPath expression. The context node of the XPath expression is the  
4008 document node of this stand alone document.

## 4009 A.3 Functions

4010 XACML specifies the following functions. If an argument of one of these functions were to evaluate to  
4011 "Indeterminate", then the function SHALL be set to "Indeterminate".

### 4012 A.3.1 Equality predicates

4013 The following functions are the equality functions for the various primitive types. Each function for a  
4014 particular data-type follows a specified standard convention for that data-type.

- 4015 • urn:oasis:names:tc:xacml:1.0:function:string-equal  
4016     This function SHALL take two arguments of data-type  
4017     "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4018     "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if  
4019     the value of both of its arguments are of equal length and each string is determined to be equal.  
4020     Otherwise, it SHALL return "False". The comparison SHALL use Unicode codepoint collation, as  
4021     defined for the identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by **[XF]**.
- 4022 • urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case  
4023     This function SHALL take two arguments of data-type  
4024     "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4025     "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be "True" if and only if the  
4026     two strings are equal as defined by urn:oasis:names:tc:xacml:1.0:function:string-equal after they  
4027     have both been converted to lower case with urn:oasis:names:tc:xacml:1.0:function:string-  
4028     normalize-to-lower-case.
- 4029 • urn:oasis:names:tc:xacml:1.0:function:boolean-equal  
4030     This function SHALL take two arguments of data-type  
4031     "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return an  
4032     "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if  
4033     the arguments are equal. Otherwise, it SHALL return "False".
- 4034 • urn:oasis:names:tc:xacml:1.0:function:integer-equal  
4035     This function SHALL take two arguments of data-type  
4036     "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an  
4037     "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if  
4038     the two arguments represent the same number.
- 4039 • urn:oasis:names:tc:xacml:1.0:function:double-equal  
4040     This function SHALL take two arguments of data-type  
4041     "http://www.w3.org/2001/XMLSchema#double" and SHALL return an  
4042     "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on doubles  
4043     according to IEEE 754 **[IEEE754]**.
- 4044 • urn:oasis:names:tc:xacml:1.0:function:date-equal  
4045     This function SHALL take two arguments of data-type  
4046     "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4047     "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to  
4048     the "op:date-equal" function **[XF]** Section 10.4.9.
- 4049 • urn:oasis:names:tc:xacml:1.0:function:time-equal  
4050     This function SHALL take two arguments of data-type  
4051     "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
4052     "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to  
4053     the "op:time-equal" function **[XF]** Section 10.4.12.
- 4054 • urn:oasis:names:tc:xacml:1.0:function:dateTime-equal

- 4055 This function SHALL take two arguments of data-type  
4056 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
4057 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according to  
4058 the "op:dateTime-equal" function [XF] Section 10.4.6.
- 4059 • urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal  
4060 This function SHALL take two arguments of data-type  
4061 "http://www.w3.org/2001/XMLSchema#dayTimeDuration" and SHALL return an  
4062 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
4063 according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical  
4064 representation of each argument MUST be converted to a value expressed in fractional seconds  
4065 [XF] Section 10.3.2.
  - 4066 • urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal  
4067 This function SHALL take two arguments of data-type  
4068 "http://www.w3.org/2001/XMLSchema#yearMonthDuration" and SHALL return an  
4069 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
4070 according to the "op:duration-equal" function [XF] Section 10.4.5. Note that the lexical  
4071 representation of each argument MUST be converted to a value expressed in fractional seconds  
4072 [XF] Section 10.3.2.
  - 4073 • urn:oasis:names:tc:xacml:1.0:function:anyURI-equal  
4074 This function SHALL take two arguments of data-type  
4075 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an  
4076 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if and only if  
4077 the values of the two arguments are equal on a codepoint-by-codepoint basis.
  - 4078 • urn:oasis:names:tc:xacml:1.0:function:x500Name-equal  
4079 This function SHALL take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name"  
4080 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if  
4081 and only if each Relative Distinguished Name (RDN) in the two arguments matches. Otherwise,  
4082 it SHALL return "False". Two RDNs shall be said to match if and only if the result of the following  
4083 operations is "True".
    - 4084 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory Access  
4085 Protocol (v3): UTF-8 String Representation of Distinguished Names".
    - 4086 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute  
4087 ValuePairs in that RDN in ascending order when compared as octet strings (described in  
4088 ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
    - 4089 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key  
4090 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section 4.1.2.4  
4091 "Issuer".
  - 4092 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal  
4093 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-  
4094 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It  
4095 SHALL return "True" if and only if the two arguments are equal. Otherwise, it SHALL return  
4096 "False". An RFC822 name consists of a local-part followed by "@" followed by a domain-part.  
4097 The local-part is case-sensitive, while the domain-part (which is usually a DNS host name) is not  
4098 case-sensitive. Perform the following operations:
    - 4099 1. Normalize the domain-part of each argument to lower case
    - 4100 2. Compare the expressions by applying the function  
4101 "urn:oasis:names:tc:xacml:1.0:function:string-equal" to the normalized arguments.
  - 4102 • urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal

4103 This function SHALL take two arguments of data-type  
4104 "http://www.w3.org/2001/XMLSchema#hexBinary" and SHALL return an  
4105 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet sequences  
4106 represented by the value of both arguments have equal length and are equal in a conjunctive,  
4107 point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function.  
4108 Otherwise, it SHALL return "False". The conversion from the string representation to an octet  
4109 sequence SHALL be as specified in [XS] Section 3.2.15.

- 4110 • urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal

4111 This function SHALL take two arguments of data-type  
4112 "http://www.w3.org/2001/XMLSchema#base64Binary" and SHALL return an  
4113 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the octet sequences  
4114 represented by the value of both arguments have equal length and are equal in a conjunctive,  
4115 point-wise, comparison using the "urn:oasis:names:tc:xacml:1.0:function:integer-equal" function.  
4116 Otherwise, it SHALL return "False". The conversion from the string representation to an octet  
4117 sequence SHALL be as specified in [XS] Section 3.2.16.

### 4118 A.3.2 Arithmetic functions

4119 All of the following functions SHALL take two arguments of the specified data-type, integer, or double,  
4120 and SHALL return an element of integer or double data-type, respectively. However, the "add" and  
4121 "multiply" functions MAY take more than two arguments. Each function evaluation operating on doubles  
4122 SHALL proceed as specified by their logical counterparts in IEEE 754 [IEEE754]. For all of these  
4123 functions, if any argument is "Indeterminate", then the function SHALL evaluate to "Indeterminate". In the  
4124 case of the divide functions, if the divisor is zero, then the function SHALL evaluate to "Indeterminate".

- 4125 • urn:oasis:names:tc:xacml:1.0:function:integer-add  
4126 This function MUST accept two or more arguments.

- 4127 • urn:oasis:names:tc:xacml:1.0:function:double-add  
4128 This function MUST accept two or more arguments.

- 4129 • urn:oasis:names:tc:xacml:1.0:function:integer-subtract

- 4130 • urn:oasis:names:tc:xacml:1.0:function:double-subtract

- 4131 • urn:oasis:names:tc:xacml:1.0:function:integer-multiply  
4132 This function MUST accept two or more arguments.

- 4133 • urn:oasis:names:tc:xacml:1.0:function:double-multiply  
4134 This function MUST accept two or more arguments.

- 4135 • urn:oasis:names:tc:xacml:1.0:function:integer-divide

- 4136 • urn:oasis:names:tc:xacml:1.0:function:double-divide

- 4137 • urn:oasis:names:tc:xacml:1.0:function:integer-mod

4138 The following functions SHALL take a single argument of the specified data-type. The round and floor  
4139 functions SHALL take a single argument of data-type "http://www.w3.org/2001/XMLSchema#double" and  
4140 return a value of the data-type "http://www.w3.org/2001/XMLSchema#double".

- 4141 • urn:oasis:names:tc:xacml:1.0:function:integer-abs

- 4142 • urn:oasis:names:tc:xacml:1.0:function:double-abs

- 4143 • urn:oasis:names:tc:xacml:1.0:function:round

- 4144 • urn:oasis:names:tc:xacml:1.0:function:floor

### 4145 A.3.3 String conversion functions

4146 The following functions convert between values of the data-type  
4147 "http://www.w3.org/2001/XMLSchema#string" primitive types.

- 4148 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-space
- 4149     This function SHALL take one argument of data-type
- 4150     “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by stripping off all
- 4151     leading and trailing white space characters. The whitespace characters are defined in the
- 4152     metasymbol S (Production 3) of **[XML]**.
- 4153 • urn:oasis:names:tc:xacml:1.0:function:string-normalize-to-lower-case
- 4154     This function SHALL take one argument of data-type
- 4155     “http://www.w3.org/2001/XMLSchema#string” and SHALL normalize the value by converting each
- 4156     upper case character to its lower case equivalent. Case mapping shall be done as specified for
- 4157     the fn:lower-case function in **[XF]** with no tailoring for particular languages or environments.

### 4158 **A.3.4 Numeric data-type conversion functions**

4159 The following functions convert between the data-type “http://www.w3.org/2001/XMLSchema#integer”  
4160 and” http://www.w3.org/2001/XMLSchema#double” primitive types.

- 4161 • urn:oasis:names:tc:xacml:1.0:function:double-to-integer
- 4162     This function SHALL take one argument of data-type
- 4163     “http://www.w3.org/2001/XMLSchema#double” and SHALL truncate its numeric value to a whole
- 4164     number and return an element of data-type “http://www.w3.org/2001/XMLSchema#integer”.
- 4165 • urn:oasis:names:tc:xacml:1.0:function:integer-to-double
- 4166     This function SHALL take one argument of data-type
- 4167     “http://www.w3.org/2001/XMLSchema#integer” and SHALL promote its value to an element of
- 4168     data-type “http://www.w3.org/2001/XMLSchema#double” with the same numeric value. If the
- 4169     integer argument is outside the range which can be represented by a double, the result SHALL
- 4170     be Indeterminate, with the status code “urn:oasis:names:tc:xacml:1.0:status:processing-error”.

### 4171 **A.3.5 Logical functions**

4172 This section contains the specification for logical functions that operate on arguments of data-type  
4173 “http://www.w3.org/2001/XMLSchema#boolean”.

- 4174 • urn:oasis:names:tc:xacml:1.0:function:or
- 4175     This function SHALL return "False" if it has no arguments and SHALL return "True" if at least one
- 4176     of its arguments evaluates to "True". The order of evaluation SHALL be from first argument to
- 4177     last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",
- 4178     leaving the rest of the arguments unevaluated.
- 4179 • urn:oasis:names:tc:xacml:1.0:function:and
- 4180     This function SHALL return "True" if it has no arguments and SHALL return "False" if one of its
- 4181     arguments evaluates to "False". The order of evaluation SHALL be from first argument to last.
- 4182     The evaluation SHALL stop with a result of "False" if any argument evaluates to "False", leaving
- 4183     the rest of the arguments unevaluated.
- 4184 • urn:oasis:names:tc:xacml:1.0:function:n-of
- 4185     The first argument to this function SHALL be of data-type
- 4186     http://www.w3.org/2001/XMLSchema#integer. The remaining arguments SHALL be of data-type
- 4187     http://www.w3.org/2001/XMLSchema#boolean. The first argument specifies the minimum
- 4188     number of the remaining arguments that MUST evaluate to "True" for the expression to be
- 4189     considered "True". If the first argument is 0, the result SHALL be "True". If the number of
- 4190     arguments after the first one is less than the value of the first argument, then the expression
- 4191     SHALL result in "Indeterminate". The order of evaluation SHALL be: first evaluate the integer
- 4192     value, and then evaluate each subsequent argument. The evaluation SHALL stop and return
- 4193     "True" if the specified number of arguments evaluate to "True". The evaluation of arguments
- 4194     SHALL stop if it is determined that evaluating the remaining arguments will not satisfy the
- 4195     requirement.

- 4196 • urn:oasis:names:tc:xacml:1.0:function:not  
4197 This function SHALL take one argument of data-type  
4198 "http://www.w3.org/2001/XMLSchema#boolean". If the argument evaluates to "True", then the  
4199 result of the expression SHALL be "False". If the argument evaluates to "False", then the result  
4200 of the expression SHALL be "True".

4201 Note: When evaluating and, or, or n-of, it MAY NOT be necessary to attempt a full evaluation of each  
4202 argument in order to determine whether the evaluation of the argument would result in "Indeterminate".  
4203 Analysis of the argument regarding the availability of its **attributes**, or other analysis regarding errors,  
4204 such as "divide-by-zero", may render the argument error free. Such arguments occurring in the  
4205 expression in a position after the evaluation is stated to stop need not be processed.

### 4206 A.3.6 Numeric comparison functions

4207 These functions form a minimal set for comparing two numbers, yielding a Boolean result. For doubles  
4208 they SHALL comply with the rules governed by IEEE 754 [IEEE754].

- 4209 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
- 4210 • urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
- 4211 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than
- 4212 • urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
- 4213 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than
- 4214 • urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal
- 4215 • urn:oasis:names:tc:xacml:1.0:function:double-less-than
- 4216 • urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal

### 4217 A.3.7 Date and time arithmetic functions

4218 These functions perform arithmetic operations with date and time.

- 4219 • urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration  
4220 This function SHALL take two arguments, the first SHALL be of data-type  
4221 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be of data-type  
4222 "http://www.w3.org/2001/XMLSchema#dayTimeDuration". It SHALL return a result of  
4223 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by  
4224 adding the second argument to the first argument according to the specification of adding  
4225 durations to date and time [XS] Appendix E.
- 4226 • urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration  
4227 This function SHALL take two arguments, the first SHALL be a  
4228 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a  
4229 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of  
4230 "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL return the value by  
4231 adding the second argument to the first argument according to the specification of adding  
4232 durations to date and time [XS] Appendix E.
- 4233 • urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration  
4234 This function SHALL take two arguments, the first SHALL be a  
4235 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a  
4236 "http://www.w3.org/2001/XMLSchema#dayTimeDuration". It SHALL return a result of  
4237 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive duration,  
4238 then this function SHALL return the value by adding the corresponding negative duration, as per  
4239 the specification [XS] Appendix E. If the second argument is a negative duration, then the result  
4240 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-  
4241 dayTimeDuration" had been applied to the corresponding positive duration.

- 4242 • urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration
  - 4243 This function SHALL take two arguments, the first SHALL be a
  - 4244 "http://www.w3.org/2001/XMLSchema#dateTime" and the second SHALL be a
  - 4245 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
  - 4246 "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument is a positive duration,
  - 4247 then this function SHALL return the value by adding the corresponding negative duration, as per
  - 4248 the specification [XS] Appendix E. If the second argument is a negative duration, then the result
  - 4249 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-
  - 4250 yearMonthDuration" had been applied to the corresponding positive duration.
- 4251 • urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration
  - 4252 This function SHALL take two arguments, the first SHALL be a
  - 4253 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
  - 4254 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
  - 4255 "http://www.w3.org/2001/XMLSchema#date". This function SHALL return the value by adding the
  - 4256 second argument to the first argument according to the specification of adding duration to date
  - 4257 [XS] Appendix E.
- 4258 • urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration
  - 4259 This function SHALL take two arguments, the first SHALL be a
  - 4260 "http://www.w3.org/2001/XMLSchema#date" and the second SHALL be a
  - 4261 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". It SHALL return a result of
  - 4262 "http://www.w3.org/2001/XMLSchema#date". If the second argument is a positive duration, then
  - 4263 this function SHALL return the value by adding the corresponding negative duration, as per the
  - 4264 specification [XS] Appendix E. If the second argument is a negative duration, then the result
  - 4265 SHALL be as if the function "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration"
  - 4266 had been applied to the corresponding positive duration.

### 4267 A.3.8 Non-numeric comparison functions

4268 These functions perform comparison operations on two arguments of non-numerical types.

- 4269 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than
  - 4270 This function SHALL take two arguments of data-type
  - 4271 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
  - 4272 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
  - 4273 argument is lexicographically strictly greater than the second argument. Otherwise, it SHALL
  - 4274 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
  - 4275 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].
- 4276 • urn:oasis:names:tc:xacml:1.0:function:string-greater-than-or-equal
  - 4277 This function SHALL take two arguments of data-type
  - 4278 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
  - 4279 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first
  - 4280 argument is lexicographically greater than or equal to the second argument. Otherwise, it SHALL
  - 4281 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the
  - 4282 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].
- 4283 • urn:oasis:names:tc:xacml:1.0:function:string-less-than
  - 4284 This function SHALL take two arguments of data-type
  - 4285 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an
  - 4286 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only the first
  - 4287 argument is lexicographically strictly less than the second argument. Otherwise, it SHALL return
  - 4288 "False". The comparison SHALL use Unicode codepoint collation, as defined for the identifier
  - 4289 http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].
- 4290 • urn:oasis:names:tc:xacml:1.0:function:string-less-than-or-equal

- 4291 This function SHALL take two arguments of data-type  
 4292 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
 4293 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only the first  
 4294 argument is lexicographically less than or equal to the second argument. Otherwise, it SHALL  
 4295 return "False". The comparison SHALL use Unicode codepoint collation, as defined for the  
 4296 identifier http://www.w3.org/2005/xpath-functions/collation/codepoint by [XF].
- 4297 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than
 

4298 This function SHALL take two arguments of data-type  
 4299 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
 4300 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4301 argument is greater than the second argument according to the order relation specified for  
 4302 "http://www.w3.org/2001/XMLSchema#time" [XS] Section 3.2.8. Otherwise, it SHALL return  
 4303 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does  
 4304 not. In such cases, the time-in-range function should be used.
  - 4305 • urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal
 

4306 This function SHALL take two arguments of data-type  
 4307 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
 4308 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4309 argument is greater than or equal to the second argument according to the order relation  
 4310 specified for "http://www.w3.org/2001/XMLSchema#time" [XS] Section 3.2.8. Otherwise, it  
 4311 SHALL return "False". Note: it is illegal to compare a time that includes a time-zone value with  
 4312 one that does not. In such cases, the time-in-range function should be used.
  - 4313 • urn:oasis:names:tc:xacml:1.0:function:time-less-than
 

4314 This function SHALL take two arguments of data-type  
 4315 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
 4316 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4317 argument is less than the second argument according to the order relation specified for  
 4318 "http://www.w3.org/2001/XMLSchema#time" [XS] Section 3.2.8. Otherwise, it SHALL return  
 4319 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does  
 4320 not. In such cases, the time-in-range function should be used.
  - 4321 • urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal
 

4322 This function SHALL take two arguments of data-type  
 4323 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
 4324 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4325 argument is less than or equal to the second argument according to the order relation specified  
 4326 for "http://www.w3.org/2001/XMLSchema#time" [XS] Section 3.2.8. Otherwise, it SHALL return  
 4327 "False". Note: it is illegal to compare a time that includes a time-zone value with one that does  
 4328 not. In such cases, the time-in-range function should be used.
  - 4329 • urn:oasis:names:tc:xacml:2.0:function:time-in-range
 

4330 This function SHALL take three arguments of data-type  
 4331 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
 4332 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first argument falls  
 4333 in the range defined inclusively by the second and third arguments. Otherwise, it SHALL return  
 4334 "False". Regardless of its value, the third argument SHALL be interpreted as a time that is equal  
 4335 to, or later than by less than twenty-four hours, the second argument. If no time zone is provided  
 4336 for the first argument, it SHALL use the default time zone at the *context handler*. If no time zone  
 4337 is provided for the second or third arguments, then they SHALL use the time zone from the first  
 4338 argument.
  - 4339 • urn:oasis:names:tc:xacml:1.0:function:date-time-greater-than
 

4340 This function SHALL take two arguments of data-type  
 4341 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an

- 4342 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4343 argument is greater than the second argument according to the order relation specified for  
 4344 "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7. Otherwise, it  
 4345 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an  
 4346 implicit time-zone value SHALL be assigned, as described in [XS].
- 4347 • urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal
 

4348 This function SHALL take two arguments of data-type  
 4349 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
 4350 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4351 argument is greater than or equal to the second argument according to the order relation  
 4352 specified for "http://www.w3.org/2001/XMLSchema#dateTime" by [XS] part 2, section 3.2.7.  
 4353 Otherwise, it SHALL return "False". Note: if a dateTime value does not include a time-zone  
 4354 value, then an implicit time-zone value SHALL be assigned, as described in [XS].
  - 4355 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than
 

4356 This function SHALL take two arguments of data-type  
 4357 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
 4358 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4359 argument is less than the second argument according to the order relation specified for  
 4360 "http://www.w3.org/2001/XMLSchema#dateTime" by [XS, part 2, section 3.2.7]. Otherwise, it  
 4361 SHALL return "False". Note: if a dateTime value does not include a time-zone value, then an  
 4362 implicit time-zone value SHALL be assigned, as described in [XS].
  - 4363 • urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal
 

4364 This function SHALL take two arguments of data-type "http://www.w3.org/2001/XMLSchema#  
 4365 dateTime" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". It SHALL  
 4366 return "True" if and only if the first argument is less than or equal to the second argument  
 4367 according to the order relation specified for "http://www.w3.org/2001/XMLSchema#dateTime" by  
 4368 [XS] part 2, section 3.2.7. Otherwise, it SHALL return "False". Note: if a dateTime value does  
 4369 not include a time-zone value, then an implicit time-zone value SHALL be assigned, as described  
 4370 in [XS].
  - 4371 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than
 

4372 This function SHALL take two arguments of data-type  
 4373 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
 4374 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4375 argument is greater than the second argument according to the order relation specified for  
 4376 "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it SHALL  
 4377 return "False". Note: if a date value does not include a time-zone value, then an implicit time-  
 4378 zone value SHALL be assigned, as described in [XS].
  - 4379 • urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal
 

4380 This function SHALL take two arguments of data-type  
 4381 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
 4382 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4383 argument is greater than or equal to the second argument according to the order relation  
 4384 specified for "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9.  
 4385 Otherwise, it SHALL return "False". Note: if a date value does not include a time-zone value,  
 4386 then an implicit time-zone value SHALL be assigned, as described in [XS].
  - 4387 • urn:oasis:names:tc:xacml:1.0:function:date-less-than
 

4388 This function SHALL take two arguments of data-type  
 4389 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
 4390 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
 4391 argument is less than the second argument according to the order relation specified for  
 4392 "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it SHALL

4393 return "False". Note: if a date value does not include a time-zone value, then an implicit time-  
4394 zone value SHALL be assigned, as described in [XS].

- 4395 • urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal  
4396 This function SHALL take two arguments of data-type  
4397 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4398 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4399 argument is less than or equal to the second argument according to the order relation specified  
4400 for "http://www.w3.org/2001/XMLSchema#date" by [XS] part 2, section 3.2.9. Otherwise, it  
4401 SHALL return "False". Note: if a date value does not include a time-zone value, then an implicit  
4402 time-zone value SHALL be assigned, as described in [XS].

### 4403 A.3.9 String functions

4404 The following functions operate on strings and convert to and from other data types.

- 4405 • urn:oasis:names:tc:xacml:2.0:function:string-concatenate  
4406 This function SHALL take two or more arguments of data-type  
4407 "http://www.w3.org/2001/XMLSchema#string" and SHALL return a  
4408 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the concatenation, in order,  
4409 of the arguments.
- 4410 • urn:oasis:names:tc:xacml:3.0:function:boolean-from-string  
4411 This function SHALL take one argument of data-type  
4412 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4413 "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be the string converted to a  
4414 boolean.
- 4415 • urn:oasis:names:tc:xacml:3.0:function:string-from-boolean  
4416 This function SHALL take one argument of data-type  
4417 "http://www.w3.org/2001/XMLSchema#boolean", and SHALL return an  
4418 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the boolean converted to a  
4419 string.
- 4420 • urn:oasis:names:tc:xacml:3.0:function:integer-from-string  
4421 This function SHALL take one argument of data-type  
4422 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4423 "http://www.w3.org/2001/XMLSchema#integer". The result SHALL be the string converted to an  
4424 integer.
- 4425 • urn:oasis:names:tc:xacml:3.0:function:string-from-integer  
4426 This function SHALL take one argument of data-type  
4427 "http://www.w3.org/2001/XMLSchema#integer", and SHALL return an  
4428 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the integer converted to a  
4429 string.
- 4430 • urn:oasis:names:tc:xacml:3.0:function:double-from-string  
4431 This function SHALL take one argument of data-type  
4432 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4433 "http://www.w3.org/2001/XMLSchema#double". The result SHALL be the string converted to a  
4434 double.
- 4435 • urn:oasis:names:tc:xacml:3.0:function:string-from-double  
4436 This function SHALL take one argument of data-type  
4437 "http://www.w3.org/2001/XMLSchema#double", and SHALL return an  
4438 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the double converted to a  
4439 string.

- 4440 • urn:oasis:names:tc:xacml:3.0:function:time-from-string
- 4441     This function SHALL take one argument of data-type
- 4442     "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
- 4443     "http://www.w3.org/2001/XMLSchema#time". The result SHALL be the string converted to a time.
  
- 4444 • urn:oasis:names:tc:xacml:3.0:function:string-from-time
- 4445     This function SHALL take one argument of data-type
- 4446     "http://www.w3.org/2001/XMLSchema#time", and SHALL return an
- 4447     "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the time converted to a
- 4448     string.
  
- 4449 • urn:oasis:names:tc:xacml:3.0:function:date-from-string
- 4450     This function SHALL take one argument of data-type
- 4451     "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
- 4452     "http://www.w3.org/2001/XMLSchema#date". The result SHALL be the string converted to a
- 4453     date.
  
- 4454 • urn:oasis:names:tc:xacml:3.0:function:string-from-date
- 4455     This function SHALL take one argument of data-type
- 4456     "http://www.w3.org/2001/XMLSchema#date", and SHALL return an
- 4457     "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the date converted to a
- 4458     string.
  
- 4459 • urn:oasis:names:tc:xacml:3.0:function:dateTime-from-string
- 4460     This function SHALL take one argument of data-type
- 4461     "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
- 4462     "http://www.w3.org/2001/XMLSchema#dateTime". The result SHALL be the string converted to a
- 4463     dateTime.
  
- 4464 urn:oasis:names:tc:xacml:3.0:function:string-from-dateTime
- 4465     This function SHALL take one argument of data-type
- 4466     "http://www.w3.org/2001/XMLSchema#dateTime", and SHALL return an
- 4467     "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the dateTime converted to a
- 4468     string.
  
- 4469 • urn:oasis:names:tc:xacml:3.0:function:anyURI-from-string
- 4470     This function SHALL take one argument of data-type
- 4471     "http://www.w3.org/2001/XMLSchema#string", and SHALL return a
- 4472     "http://www.w3.org/2001/XMLSchema#anyURI". The result SHALL be the URI constructed by
- 4473     converting the argument to an URI.
  
- 4474 • urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI
- 4475     This function SHALL take one argument of data-type
- 4476     "http://www.w3.org/2001/XMLSchema#anyURI", and SHALL return an
- 4477     "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the URI converted to a
- 4478     string.
  
- 4479 • urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-from-string
- 4480     This function SHALL take one argument of data-type
- 4481     "http://www.w3.org/2001/XMLSchema#string", and SHALL return an
- 4482     "http://www.w3.org/2001/XMLSchema#dayTimeDuration ". The result SHALL be the string
- 4483     converted to a dayTimeDuration.
  
- 4484 • urn:oasis:names:tc:xacml:3.0:function:string-from-dayTimeDuration
- 4485     This function SHALL take one argument of data-type
- 4486     "http://www.w3.org/2001/XMLSchema#dayTimeDuration ", and SHALL return an

- 4487 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the dayTimeDuration  
4488 converted to a string.
- 4489 • urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-from-string  
4490 This function SHALL take one argument of data-type  
4491 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4492 "http://www.w3.org/2001/XMLSchema#yearMonthDuration". The result SHALL be the string  
4493 converted to a yearMonthDuration.
  - 4494 • urn:oasis:names:tc:xacml:3.0:function:string-from-yearMonthDuration  
4495 This function SHALL take one argument of data-type  
4496 "http://www.w3.org/2001/XMLSchema#yearMonthDuration", and SHALL return an  
4497 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the yearMonthDuration  
4498 converted to a string.
  - 4499 • urn:oasis:names:tc:xacml:3.0:function:x500Name-from-string  
4500 This function SHALL take one argument of data-type  
4501 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4502 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". The result SHALL be the string converted  
4503 to an x500Name.
  - 4504 • urn:oasis:names:tc:xacml:3.0:function:string-from-x500Name  
4505 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:1.0:data-  
4506 type:x500Name", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result  
4507 SHALL be the x500Name converted to a string.
  - 4508 • urn:oasis:names:tc:xacml:3.0:function:rfc822Name-from-string  
4509 This function SHALL take one argument of data-type  
4510 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4511 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". The result SHALL be the string converted  
4512 to an rfc822Name.
  - 4513 • urn:oasis:names:tc:xacml:3.0:function:string-from-rfc822Name  
4514 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:1.0:data-  
4515 type:rfc822Name", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The  
4516 result SHALL be the rfc822Name converted to a string.
  - 4517 • urn:oasis:names:tc:xacml:3.0:function:ipAddress-from-string  
4518 This function SHALL take one argument of data-type  
4519 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4520 "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress". The result SHALL be the string converted to  
4521 an ipAddress.
  - 4522 • urn:oasis:names:tc:xacml:3.0:function:string-from-ipAddress  
4523 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-  
4524 type:ipAddress", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result  
4525 SHALL be the ipAddress converted to a string.
  - 4526 • urn:oasis:names:tc:xacml:3.0:function:dnsName-from-string  
4527 This function SHALL take one argument of data-type  
4528 "http://www.w3.org/2001/XMLSchema#string", and SHALL return an  
4529 "urn:oasis:names:tc:xacml:2.0:data-type:dnsName". The result SHALL be the string converted to  
4530 a dnsName.
  - 4531 • urn:oasis:names:tc:xacml:3.0:function:string-from-dnsName  
4532 This function SHALL take one argument of data-type "urn:oasis:names:tc:xacml:2.0:data-  
4533 type:dnsName", and SHALL return an "http://www.w3.org/2001/XMLSchema#string". The result  
4534 SHALL be the dnsName converted to a string.

- 4535 • urn:oasis:names:tc:xacml:3.0:function:string-starts-with
- 4536     This function SHALL take two arguments of data-type
- 4537     "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
- 4538     "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
- 4539     begins with the first string, and false otherwise. Equality testing SHALL be done as defined for
- 4540     urn:oasis:names:tc:xacml:1.0:function:string-equal.
  
- 4541 • urn:oasis:names:tc:xacml:3.0:function:anyURI-starts-with
- 4542     This function SHALL take a first argument of data-
- 4543     type"http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
- 4544     "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
- 4545     "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
- 4546     to a string begins with the string, and false otherwise. Equality testing SHALL be done as defined
- 4547     for urn:oasis:names:tc:xacml:1.0:function:string-equal.
  
- 4548 • urn:oasis:names:tc:xacml:3.0:function:string-ends-with
- 4549     This function SHALL take two arguments of data-type
- 4550     "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
- 4551     "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
- 4552     ends with the first string, and false otherwise. Equality testing SHALL be done as defined for
- 4553     urn:oasis:names:tc:xacml:1.0:function:string-equal.
  
- 4554 • urn:oasis:names:tc:xacml:3.0:function:anyURI-ends-with
- 4555     This function SHALL take a first argument of data-type
- 4556     "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
- 4557     "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
- 4558     "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
- 4559     to a string ends with the string, and false otherwise. Equality testing SHALL be done as defined
- 4560     for urn:oasis:names:tc:xacml:1.0:function:string-equal.
  
- 4561 • urn:oasis:names:tc:xacml:3.0:function:string-contains
- 4562     This function SHALL take two arguments of data-type
- 4563     "http://www.w3.org/2001/XMLSchema#string" and SHALL return a
- 4564     "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the second string
- 4565     contains the first string, and false otherwise. Equality testing SHALL be done as defined for
- 4566     urn:oasis:names:tc:xacml:1.0:function:string-equal.
  
- 4567 • urn:oasis:names:tc:xacml:3.0:function:anyURI-contains
- 4568     This function SHALL take a first argument of data-type
- 4569     "http://www.w3.org/2001/XMLSchema#string" and an a second argument of data-type
- 4570     "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return a
- 4571     "http://www.w3.org/2001/XMLSchema#boolean". The result SHALL be true if the URI converted
- 4572     to a string contains the string, and false otherwise. Equality testing SHALL be done as defined for
- 4573     urn:oasis:names:tc:xacml:1.0:function:string-equal.
  
- 4574 • urn:oasis:names:tc:xacml:3.0:function:string-substring
- 4575     This function SHALL take a first argument of data-type
- 4576     "http://www.w3.org/2001/XMLSchema#string" and a second and a third argument of type
- 4577     "http://www.w3.org/2001/XMLSchema#integer" and SHALL return a
- 4578     "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first
- 4579     argument beginning at the position given by the second argument and ending at the position
- 4580     before the position given by the third argument. The first character of the string has position zero.
- 4581     The negative integer value -1 given for the third arguments indicates the end of the string. If the
- 4582     second or third arguments are out of bounds, then the function MUST evaluate to Indeterminate
- 4583     with a status code of urn:oasis:names:tc:xacml:1.0:status:processing-error.
  
- 4584 • urn:oasis:names:tc:xacml:3.0:function:anyURI-substring

4585 This function SHALL take a first argument of data-type  
4586 "http://www.w3.org/2001/XMLSchema#anyURI" and a second and a third argument of type  
4587 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return a  
4588 "http://www.w3.org/2001/XMLSchema#string". The result SHALL be the substring of the first  
4589 argument converted to a string beginning at the position given by the second argument and  
4590 ending at the position before the position given by the third argument. The first character of the  
4591 URI converted to a string has position zero. The negative integer value -1 given for the third  
4592 arguments indicates the end of the string. If the second or third arguments are out of bounds,  
4593 then the function MUST evaluate to Indeterminate with a status code of  
4594 urn:oasis:names:tc:xacml:1.0:status:processing-error. If the resulting substring  
4595 is not syntactically a valid URI, then the function MUST evaluate to Indeterminate with a status  
4596 code of urn:oasis:names:tc:xacml:1.0:status:processing-error.  
4597

### 4598 A.3.10 Bag functions

4599 These functions operate on a **bag** of 'type' values, where type is one of the primitive data-types, and x.x  
4600 is a version of XACML where the function has been defined. Some additional conditions defined for  
4601 each function below SHALL cause the expression to evaluate to "Indeterminate".

- 4602 • urn:oasis:names:tc:xacml:x.x:function:type-one-and-only

4603 This function SHALL take a **bag** of 'type' values as an argument and SHALL return a value of '-  
4604 type'. It SHALL return the only value in the **bag**. If the **bag** does not have one and only one  
4605 value, then the expression SHALL evaluate to "Indeterminate".

- 4606 • urn:oasis:names:tc:xacml:x.x:function:type-bag-size

4607 This function SHALL take a **bag** of 'type' values as an argument and SHALL return an  
4608 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

- 4609 • urn:oasis:names:tc:xacml:x.x:function:type-is-in

4610 This function SHALL take an argument of 'type' as the first argument and a **bag** of type values as  
4611 the second argument and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The  
4612 function SHALL evaluate to "True" if and only if the first argument matches by the  
4613 "urn:oasis:names:tc:xacml:x.x:function:type-equal" any value in the **bag**. Otherwise, it SHALL  
4614 return "False".

- 4615 • urn:oasis:names:tc:xacml:x.x:function:type-bag

4616 This function SHALL take any number of arguments of 'type' and return a **bag** of 'type' values  
4617 containing the values of the arguments. An application of this function to zero arguments SHALL  
4618 produce an empty **bag** of the specified data-type.

### 4619 A.3.11 Set functions

4620 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**.

- 4621 • urn:oasis:names:tc:xacml:x.x:function:type-intersection

4622 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a  
4623 **bag** of 'type' values such that it contains only elements that are common between the two **bags**,  
4624 which is determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal". No duplicates, as  
4625 determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal", SHALL exist in the result.

- 4626 • urn:oasis:names:tc:xacml:x.x:function:type-at-least-one-member-of

4627 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a  
4628 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL evaluate to "True" if and  
4629 only if at least one element of the first argument is contained in the second argument as  
4630 determined by "urn:oasis:names:tc:xacml:x.x:function:type-is-in".

- 4631 • urn:oasis:names:tc:xacml:x.x:function:type-union

4632 This function SHALL take two or more arguments that are both a **bag** of 'type' values. The  
4633 expression SHALL return a **bag** of 'type' such that it contains all elements of all the argument  
4634 **bags**. No duplicates, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",  
4635 SHALL exist in the result.

4636 • urn:oasis:names:tc:xacml:x.x:function:type-subset

4637 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a  
4638 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the first  
4639 argument is a subset of the second argument. Each argument SHALL be considered to have had  
4640 its duplicates removed, as determined by "urn:oasis:names:tc:xacml:x.x:function:type-equal",  
4641 before the subset calculation.

4642 • urn:oasis:names:tc:xacml:x.x:function:type-set-equals

4643 This function SHALL take two arguments that are both a **bag** of 'type' values. It SHALL return a  
4644 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return the result of applying  
4645 "urn:oasis:names:tc:xacml:1.0:function:and" to the application of  
4646 "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the first and second arguments and the  
4647 application of "urn:oasis:names:tc:xacml:x.x:function:type-subset" to the second and first  
4648 arguments.

### 4649 A.3.12 Higher-order bag functions

4650 This section describes functions in XACML that perform operations on **bags** such that functions may be  
4651 applied to the **bags** in general.

4652 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally specify  
4653 the semantics of these functions. Although the English description is adequate, a formal specification of  
4654 the semantics is helpful.

4655 For a quick summary, in the following Haskell notation, a function definition takes the form of clauses that  
4656 are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty list, whereas the  
4657 expression "(x:xs)" matches against an argument of a non-empty list of which "x" represents the first  
4658 element of the list, and "xs" is the rest of the list, which may be an empty list. We use the Haskell notion  
4659 of a list, which is an ordered collection of elements, to model the XACML **bags** of values.

4660 A simple Haskell definition of a familiar function "urn:oasis:names:tc:xacml:1.0:function:and" that takes a  
4661 list of values of type Boolean is defined as follows:

```
4662     and:: [Bool]    -> Bool
4663     and []         = True
4664     and (x:xs)    = x && (and xs)
```

4665 The first definition line denoted by a "::" formally describes the data-type of the function, which takes a list  
4666 of Booleans, denoted by "[Bool]", and returns a Boolean, denoted by "Bool". The second definition line is  
4667 a clause that states that the function "and" applied to the empty list is "True". The third definition line is a  
4668 clause that states that for a non-empty list, such that the first element is "x", which is a value of data-type  
4669 Bool, the function "and" applied to x SHALL be combined with, using the logical conjunction function,  
4670 which is denoted by the infix symbol "&&", the result of recursively applying the function "and" to the rest  
4671 of the list. Of course, an application of the "and" function is "True" if and only if the list to which it is  
4672 applied is empty or every element of the list is "True". For example, the evaluation of the following  
4673 Haskell expressions,

4674 (and []), (and [True]), (and [True,True]), (and [True,True,False])

4675 evaluate to "True", "True", "True", and "False", respectively.

4676 • urn:oasis:names:tc:xacml:1.0:function:any-of

4677 This function applies a Boolean function between specific primitive values and a **bag** of values,  
4678 and SHALL return "True" if and only if the **predicate** is "True" for at least one element of the **bag**.

4679 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4680 be an <Function> element that names a Boolean function that takes n arguments of primitive  
4681 types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-  
4682 types and one SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated as  
4683 if the function named in the <Function> argument were applied to the n-1 non-bag arguments  
4684 and each element of the bag argument and the results are combined with  
4685 "urn:oasis:names:tc:xacml:1.0:function:or".

4686 For example, the following expression SHALL return "True":

```
4687 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
4688   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4689   <AttributeValue  
4690     DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4691     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4692       <AttributeValue  
4693         DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4694         <AttributeValue  
4695           DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4696           <AttributeValue  
4697             DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4698             <AttributeValue  
4699               DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4700             </Apply>  
4701           </Apply>
```

4702 This expression is "True" because the first argument is equal to at least one of the elements of  
4703 the **bag**, according to the function.

4704 • urn:oasis:names:tc:xacml:1.0:function:all-of

4705 This function applies a Boolean function between a specific primitive values and a **bag** of values,  
4706 and returns "True" if and only if the **predicate** is "True" for every element of the **bag**.

4707 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4708 be a <Function> element that names a Boolean function that takes n arguments of primitive  
4709 types. Under the remaining n arguments, n-1 parameters SHALL be values of primitive data-  
4710 types and one SHALL be a **bag** of a primitive data-type. The expression SHALL be evaluated as  
4711 if the function named in the <Function> argument were applied to the n-1 non-bag arguments  
4712 and each element of the bag argument and the results are combined with  
4713 "urn:oasis:names:tc:xacml:1.0:function:and".

4714 For example, the following expression SHALL evaluate to "True":

```
4715 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">  
4716   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-  
4717     greater-than"/>  
4718   <AttributeValue  
4719     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>  
4720     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">  
4721       <AttributeValue  
4722         DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>  
4723         <AttributeValue  
4724           DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>  
4725           <AttributeValue  
4726             DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>  
4727             <AttributeValue  
4728               DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>  
4729             </Apply>  
4730           </Apply>
```

4731 This expression is "True" because the first argument (10) is greater than all of the elements of the  
4732 **bag** (9,3,4 and 2).

4733 • urn:oasis:names:tc:xacml:1.0:function:any-of-any

4734 This function applies a Boolean function on each tuple from the cross product on all bags  
4735 arguments, and returns "True" if and only if the **predicate** is "True" for at least one inside-function  
4736 call.

4737 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4738 be an <Function> element that names a Boolean function that takes n arguments. The  
4739 remaining arguments are either primitive data types or bags of primitive types. The expression  
4740 SHALL be evaluated as if the function named in the <Function> argument was applied between  
4741 every tuple of the cross product on all bags and the primitive values, and the results were  
4742 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the result of  
4743 the expression SHALL be "True" if and only if the applied **predicate** is "True" for at least one  
4744 function call on the tuples from the **bags** and primitive values.

4745 For example, the following expression SHALL evaluate to "True":

```
4746 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">  
4747   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>  
4748   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4749     <AttributeValue  
4750       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4751     <AttributeValue  
4752       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>  
4753   </Apply>  
4754   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4755     <AttributeValue  
4756       DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>  
4757     <AttributeValue  
4758       DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>  
4759     <AttributeValue  
4760       DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>  
4761     <AttributeValue  
4762       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>  
4763   </Apply>  
4764 </Apply>
```

4765 This expression is "True" because at least one of the elements of the first **bag**, namely "Ringo", is  
4766 equal to at least one of the elements of the second **bag**.

4767 • urn:oasis:names:tc:xacml:1.0:function:all-of-any

4768 This function applies a Boolean function between the elements of two **bags**. The expression  
4769 SHALL be "True" if and only if the supplied **predicate** is 'True' between each element of the first  
4770 **bag** and any element of the second **bag**.

4771 This function SHALL take three arguments. The first argument SHALL be an <Function>  
4772 element that names a Boolean function that takes two arguments of primitive types. The second  
4773 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a  
4774 primitive data-type. The expression SHALL be evaluated as if the  
4775 "urn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each value of the first  
4776 **bag** and the whole of the second **bag** using the supplied xacml:Function, and the results were  
4777 then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4778 In Haskell, taking advantage of the "any\_of" function defined in Haskell above, the semantics of  
4779 the "all\_of\_any" function are as follows:

```
4780 all_of_any :: ( a -> b -> Bool )      -> [a]-> [b] -> Bool  
4781 all_of_any f []      ys              = True  
4782 all_of_any f (x:xs)  ys              = (any_of f x ys) && (all_of_any f xs ys)
```

4783 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first element of  
4784 the list as "x" and the rest of the list as "xs".

4785 For example, the following expression SHALL evaluate to "True":

```

4786 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4787   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-
4788   greater-than"/>
4789   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4790     <AttributeValue
4791     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4792     <AttributeValue
4793     DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4794   </Apply>
4795   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4796     <AttributeValue
4797     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4798     <AttributeValue
4799     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4800     <AttributeValue
4801     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4802     <AttributeValue
4803     DataType="http://www.w3.org/2001/XMLSchema#integer">19</AttributeValue>
4804   </Apply>
4805 </Apply>

```

4806 This expression is "True" because each of the elements of the first **bag** is greater than at least  
4807 one of the elements of the second **bag**.

4808 • urn:oasis:names:tc:xacml:1.0:function:any-of-all

4809 This function applies a Boolean function between the elements of two **bags**. The expression  
4810 SHALL be "True" if and only if the supplied **predicate** is "True" between each element of the  
4811 second **bag** and any element of the first **bag**.

4812 This function SHALL take three arguments. The first argument SHALL be an <Function>  
4813 element that names a Boolean function that takes two arguments of primitive types. The second  
4814 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a  
4815 primitive data-type. The expression SHALL be evaluated as if the  
4816 "urn:oasis:names:tc:xacml:1.0:function:any-of" function had been applied to each value of the  
4817 second **bag** and the whole of the first **bag** using the supplied xacml:Function, and the results  
4818 were then combined using "urn:oasis:names:tc:xacml:1.0:function:and".

4819 In Haskell, taking advantage of the "all\_of" function defined in Haskell above, the semantics of the  
4820 "any\_of\_all" function are as follows:

```

4821 any_of_all :: ( a -> b -> Bool )      -> [a]-> [b] -> Bool
4822 any_of_all f []          ys          = False
4823 any_of_all f (x:xs)     ys          = (all_of f x ys) || ( any_of_all f xs ys)

```

4824 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4825 element of the list as "x" and the rest of the list as "xs".

4826 For example, the following expression SHALL evaluate to "True":

```

4827 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4828   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-
4829   greater-than"/>
4830   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4831     <AttributeValue
4832     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4833     <AttributeValue
4834     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4835   </Apply>
4836   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4837     <AttributeValue
4838     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4839     <AttributeValue
4840     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>

```

```

4841     <AttributeValue
4842     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4843     <AttributeValue
4844     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4845     </Apply>
4846 </Apply>

```

4847 This expression is "True" because, for all of the values in the second **bag**, there is a value in the  
4848 first **bag** that is greater.

4849 • urn:oasis:names:tc:xacml:1.0:function:all-of-all

4850 This function applies a Boolean function between the elements of two **bags**. The expression  
4851 SHALL be "True" if and only if the supplied **predicate** is "True" between each and every element  
4852 of the first **bag** collectively against all the elements of the second **bag**.

4853 This function SHALL take three arguments. The first argument SHALL be an <Function>  
4854 element that names a Boolean function that takes two arguments of primitive types. The second  
4855 argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be a **bag** of a  
4856 primitive data-type. The expression is evaluated as if the function named in the <Function>  
4857 element were applied between every element of the second argument and every element of the  
4858 third argument and the results were combined using "urn:oasis:names:tc:xacml:1.0:function:and".  
4859 The semantics are that the result of the expression is "True" if and only if the applied **predicate** is  
4860 "True" for all elements of the first **bag** compared to all the elements of the second **bag**.

4861 In Haskell, taking advantage of the "all\_of" function defined in Haskell above, the semantics of the  
4862 "all\_of\_all" function is as follows:

```

4863     all_of_all :: ( a -> b -> Bool )    -> [a] -> [b] -> Bool
4864     all_of_all f []          ys        = True
4865     all_of_all f (x:xs)     ys        = (all_of f x ys) && (all_of_all f xs ys)

```

4866 In the above notation, "f" is the function to be applied and "(x:xs)" represents the first element of  
4867 the list as "x" and the rest of the list as "xs".

4868 For example, the following expression SHALL evaluate to "True":

```

4869 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4870   <Function FunctionId="urn:oasis:names:tc:xacml:2.0:function:integer-
4871   greater-than"/>
4872   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4873     <AttributeValue
4874     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4875     <AttributeValue
4876     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4877   </Apply>
4878   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4879     <AttributeValue
4880     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4881     <AttributeValue
4882     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4883     <AttributeValue
4884     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4885     <AttributeValue
4886     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4887   </Apply>
4888 </Apply>

```

4889 This expression is "True" because all elements of the first **bag**, "5" and "6", are each greater than  
4890 all of the integer values "1", "2", "3", "4" of the second **bag**.

4891 • urn:oasis:names:tc:xacml:1.0:function:map

4892 This function converts a **bag** of values to another **bag** of values.

4893 This function SHALL take n+1 arguments, where n is one or greater. The first argument SHALL  
4894 be a <Function> element naming a function that takes a n arguments of a primitive data-type  
4895 and returns a value of a primitive data-type Under the remaining n arguments, n-1 parameters  
4896 SHALL be values of primitive data-types and one SHALL be a **bag** of a primitive data-type. The  
4897 expression SHALL be evaluated as if the function named in the <Function> argument were  
4898 applied to the n-1 non-bag arguments and each element of the bag argument and resulting in a  
4899 **bag** of the converted value. The result SHALL be a **bag** of the primitive data-type that is returned  
4900 by the function named in the <xacml:Function> element.

4901 For example, the following expression,

```
4902 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">  
4903   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-  
4904   normalize-to-lower-case">  
4905     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">  
4906       <AttributeValue  
4907       DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>  
4908       <AttributeValue  
4909       DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>  
4910     </Apply>  
4911   </Apply>
```

4912 evaluates to a **bag** containing “hello” and “world!”.

### 4913 A.3.13 Regular-expression-based functions

4914 These functions operate on various types using regular expressions and evaluate to  
4915 “http://www.w3.org/2001/XMLSchema#boolean”.

- 4916 • urn:oasis:names:tc:xacml:1.0:function:string-regexp-match

4917 This function decides a regular expression match. It SHALL take two arguments of  
4918 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an  
4919 “http://www.w3.org/2001/XMLSchema#boolean”. The first argument SHALL be a regular  
4920 expression and the second argument SHALL be a general string. The function specification  
4921 SHALL be that of the “xf:matches” function with the arguments reversed [XF] Section 7.6.2.

- 4922 • urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match

4923 This function decides a regular expression match. It SHALL take two arguments; the first is of  
4924 type “http://www.w3.org/2001/XMLSchema#string” and the second is of type  
4925 “http://www.w3.org/2001/XMLSchema#anyURI”. It SHALL return an  
4926 “http://www.w3.org/2001/XMLSchema#boolean”. The first argument SHALL be a regular  
4927 expression and the second argument SHALL be a URI. The function SHALL convert the second  
4928 argument to type “http://www.w3.org/2001/XMLSchema#string”, then apply  
4929 “urn:oasis:names:tc:xacml:1.0:function:string-regexp-match”.

- 4930 • urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match

4931 This function decides a regular expression match. It SHALL take two arguments; the first is of  
4932 type “http://www.w3.org/2001/XMLSchema#string” and the second is of type  
4933 “urn:oasis:names:tc:xacml:2.0:data-type:ipAddress”. It SHALL return an  
4934 “http://www.w3.org/2001/XMLSchema#boolean”. The first argument SHALL be a regular  
4935 expression and the second argument SHALL be an IPv4 or IPv6 address. The function SHALL  
4936 convert the second argument to type “http://www.w3.org/2001/XMLSchema#string”, then apply  
4937 “urn:oasis:names:tc:xacml:1.0:function:string-regexp-match”.

- 4938 • urn:oasis:names:tc:xacml:2.0:function:dnsName-regexp-match

4939 This function decides a regular expression match. It SHALL take two arguments; the first is of  
4940 type “http://www.w3.org/2001/XMLSchema#string” and the second is of type  
4941 “urn:oasis:names:tc:xacml:2.0:data-type:dnsName”. It SHALL return an  
4942 “http://www.w3.org/2001/XMLSchema#boolean”. The first argument SHALL be a regular  
4943 expression and the second argument SHALL be a DNS name. The function SHALL convert the

- 4944 second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply  
4945 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4946 • urn:oasis:names:tc:xacml:2.0:function:rfc822Name-regexp-match
- 4947 This function decides a regular expression match. It SHALL take two arguments; the first is of  
4948 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4949 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name". It SHALL return an  
4950 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4951 expression and the second argument SHALL be an RFC 822 name. The function SHALL convert  
4952 the second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply  
4953 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".
- 4954 • urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match
- 4955 This function decides a regular expression match. It SHALL take two arguments; the first is of  
4956 type "http://www.w3.org/2001/XMLSchema#string" and the second is of type  
4957 "urn:oasis:names:tc:xacml:1.0:data-type:x500Name". It SHALL return an  
4958 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4959 expression and the second argument SHALL be an X.500 directory name. The function SHALL  
4960 convert the second argument to type "http://www.w3.org/2001/XMLSchema#string", then apply  
4961 "urn:oasis:names:tc:xacml:1.0:function:string-regexp-match".

### 4962 **A.3.14 Special match functions**

4963 These functions operate on various types and evaluate to  
4964 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching algorithm.

- 4965 • urn:oasis:names:tc:xacml:1.0:function:x500Name-match
- 4966 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-type:x500Name"  
4967 and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It shall return "True" if and  
4968 only if the first argument matches some terminal sequence of RDNs from the second argument  
4969 when compared using x500Name-equal.
- 4970 • urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match
- 4971 This function SHALL take two arguments, the first is of data-type  
4972 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type  
4973 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an  
4974 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if the  
4975 first argument matches the second argument according to the following specification.
- 4976 An RFC822 name consists of a local-part followed by "@" followed by a domain-part. The local-  
4977 part is case-sensitive, while the domain-part (which is usually a DNS name) is not case-sensitive.
- 4978 The second argument contains a complete rfc822Name. The first argument is a complete or  
4979 partial rfc822Name used to select appropriate values in the second argument as follows.
- 4980 In order to match a particular address in the second argument, the first argument must specify the  
4981 complete mail address to be matched. For example, if the first argument is  
4982 "Anderson@sun.com", this matches a value in the second argument of "Anderson@sun.com"  
4983 and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com", "anderson@sun.com" or  
4984 "Anderson@east.sun.com".
- 4985 In order to match any address at a particular domain in the second argument, the first argument  
4986 must specify only a domain name (usually a DNS name). For example, if the first argument is  
4987 "sun.com", this matches a value in the first argument of "Anderson@sun.com" or  
4988 "Baxter@SUN.COM", but not "Anderson@east.sun.com".
- 4989 In order to match any address in a particular domain in the second argument, the first argument  
4990 must specify the desired domain-part with a leading ".". For example, if the first argument is  
4991 ".east.sun.com", this matches a value in the second argument of "Anderson@east.sun.com" and  
4992 "anne.anderson@ISRG.EAST.SUN.COM" but not "Anderson@sun.com".

### 4993 A.3.15 XPath-based functions

4994 This section specifies functions that take XPath expressions for arguments. An XPath expression  
4995 evaluates to a node-set, which is a set of XML nodes that match the expression. A node or node-set is  
4996 not in the formal data-type system of XACML. All comparison or other operations on node-sets are  
4997 performed in isolation of the particular function specified. The context nodes and namespace mappings  
4998 of the XPath expressions are defined by the XPath data-type, see section B.3. The following functions  
4999 are defined:

- 5000 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-count

5001 This function SHALL take an “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression” as an  
5002 argument and evaluates to an “http://www.w3.org/2001/XMLSchema#integer”. The value  
5003 returned from the function SHALL be the count of the nodes within the node-set that match the  
5004 given XPath expression. If the <Content> element of the category to which the XPath  
5005 expression applies to is not present in the request, this function SHALL return a value of zero.

- 5006 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal

5007 This function SHALL take two “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression”  
5008 arguments and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. The function  
5009 SHALL return "True" if any of the XML nodes in the node-set matched by the first argument  
5010 equals any of the XML nodes in the node-set matched by the second argument. Two nodes are  
5011 considered equal if they have the same identity. If the <Content> element of the category to  
5012 which either XPath expression applies to is not present in the request, this function SHALL return  
5013 a value of “False”.

- 5014 • urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

5015 This function SHALL take two “urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression”  
5016 arguments and SHALL return an “http://www.w3.org/2001/XMLSchema#boolean”. This function  
5017 SHALL evaluate to "True" if one of the following two conditions is satisfied: (1) Any of the XML  
5018 nodes in the node-set matched by the first argument is equal to any of the XML nodes in the  
5019 node-set matched by the second argument; (2) any node below any of the XML nodes in the  
5020 node-set matched by the first argument is equal to any of the XML nodes in the node-set  
5021 matched by the second argument. Two nodes are considered equal if they have the same  
5022 identity. If the <Content> element of the category to which either XPath expression applies to is  
5023 not present in the request, this function SHALL return a value of “False”.

5024 NOTE: The first **condition** is equivalent to "xpath-node-equal", and guarantees that "xpath-node-equal" is  
5025 a special case of "xpath-node-match".

### 5026 A.3.16 Other functions

- 5027 • urn:oasis:names:tc:xacml:3.0:function:access-permitted

5028 This function SHALL take an “http://www.w3.org/2001/XMLSchema#anyURI” and an  
5029 “http://www.w3.org/2001/XMLSchema#string” as arguments. The first argument SHALL be  
5030 interpreted as an **attribute** category. The second argument SHALL be interpreted as the XML  
5031 content of an <Attributes> element with *Category* equal to the first argument. The function  
5032 evaluates to an “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return  
5033 "True" if and only if the **policy** evaluation described below returns the value of "Permit".

5034 The following evaluation is described as if the **context** is actually instantiated, but it is only  
5035 required that an equivalent result be obtained.

5036 The function SHALL construct a new **context**, by copying all the information from the current  
5037 **context**, omitting any <Attributes> element with *Category* equal to the first argument. The  
5038 second function argument SHALL be added to the **context** as the content of an <Attributes>  
5039 element with *Category* equal to the first argument.

5040 The function SHALL invoke a complete **policy** evaluation using the newly constructed **context**.  
5041 This evaluation SHALL be completely isolated from the evaluation which invoked the function, but  
5042 shall use all current **policies** and combining algorithms, including any per request **policies**.

5043 The **PDP** SHALL detect any loop which may occur if successive evaluations invoke this function  
5044 by counting the number of total invocations of any instance of this function during any single initial  
5045 invocation of the **PDP**. If the total number of invocations exceeds the bound for such invocations,  
5046 the initial invocation of this function evaluates to Indeterminate with a  
5047 “urn:oasis:names:tc:xacml:1.0:status:processing-error” status code. Also, see the security  
5048 considerations in section 9.1.8.

### 5049 **A.3.17 Extension functions and primitive types**

5050 Functions and primitive types are specified by string identifiers allowing for the introduction of functions in  
5051 addition to those specified by XACML. This approach allows one to extend the XACML module with  
5052 special functions and special primitive data-types.

5053 In order to preserve the integrity of the XACML evaluation strategy, the result of an extension function  
5054 SHALL depend only on the values of its arguments. Global and hidden parameters SHALL NOT affect  
5055 the evaluation of an expression. Functions SHALL NOT have side effects, as evaluation order cannot be  
5056 guaranteed in a standard way.

### 5057 **A.4 Functions, data types and algorithms planned for deprecation**

5058 The following functions, data types and algorithms have been defined by previous versions of XACML  
5059 and newer and better alternatives are defined in XACML 3.0. Their use is discouraged for new use and  
5060 they are candidates for deprecation in future versions of XACML.

5061 The following xpath based functions have been replaced with equivalent functions which use the new  
5062 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression datatype instead of strings.

- 5063 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-count
  - 5064 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-count
- 5065 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-equal
  - 5066 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-equal
- 5067 • urn:oasis:names:tc:xacml:1.0:function:xpath-node-match
  - 5068 • Replaced with urn:oasis:names:tc:xacml:3.0:function:xpath-node-match

5069 The following URI and string concatenation function has been replaced with a string to URI conversion  
5070 function, which allows the use of the general string functions with URI through string conversion.

- 5071 • urn:oasis:names:tc:xacml:2.0:function:uri-string-concatenate
  - 5072 • Replaced by urn:oasis:names:tc:xacml:3.0:function:string-from-anyURI

5073 The following identifiers have been replaced with official identifiers defined by W3C.

- 5074 • http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration
  - 5075 • Replaced with http://www.w3.org/2001/XMLSchema#dayTimeDuration
- 5076 • http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration
  - 5077 • Replaced with http://www.w3.org/2001/XMLSchema#yearMonthDuration

5078 The following functions have been replaced with functions which use the updated dayTimeDuration and  
5079 yearMonthDuration data types.

- 5080 • urn:oasis:names:tc:xacml:1.0:function:dayTimeDuration-equal
  - 5081 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal
- 5082 • urn:oasis:names:tc:xacml:1.0:function:yearMonthDuration-equal
  - 5083 • Replaced with urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal

- 5084 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration
- 5085 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-dayTimeDuration
- 5086 • urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration
- 5087 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-add-yearMonthDuration
- 5088 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-dayTimeDuration
- 5089 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-dayTimeDuration
- 5090 • urn:oasis:names:tc:xacml:1.0:function:dateTime-subtract-yearMonthDuration
- 5091 • Replaced with urn:oasis:names:tc:xacml:3.0:function:dateTime-subtract-yearMonthDuration
- 5092 • urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration
- 5093 • Replaced with urn:oasis:names:tc:xacml:3.0:function:date-add-yearMonthDuration
- 5094 • urn:oasis:names:tc:xacml:1.0:function:date-subtract-yearMonthDuration
- 5095 • Replaced with urn:oasis:names:tc:xacml:3.0:function:date-subtract-yearMonthDuration
- 5096 The following combining algorithms have been replaced with new variants which allow for better handling
- 5097 of “Indeterminate” results.
- 5098 • urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides
- 5099 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides
- 5100 • urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides
- 5101 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides
- 5102 • urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides
- 5103 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides
- 5104 • urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides
- 5105 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides
- 5106 • urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides
- 5107 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides
- 5108 • urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides
- 5109 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-overrides
- 5110 • urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides
- 5111 • Replaced with urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-overrides
- 5112 • urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides
- 5113 • Replaced with urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-overrides

---

## 5114 B. XACML identifiers (normative)

5115 This section defines standard identifiers for commonly used entities.

### 5116 B.1 XACML namespaces

5117 XACML is defined using this identifier.

5118 `urn:oasis:names:tc:xacml:3.0:core:schema`

### 5119 B.2 Attribute categories

5120 The following **attribute** category identifiers MUST be used when an XACML 2.0 or earlier **policy** or  
5121 request is translated into XACML 3.0.

5122 **Attributes** previously placed in the **Resource**, **Action**, and **Environment** sections of a request are  
5123 placed in an **attribute** category with the following identifiers respectively. It is RECOMMENDED that they  
5124 are used to list **attributes** of **resources**, **actions**, and the **environment** respectively when authoring  
5125 XACML 3.0 **policies** or requests.

5126 `urn:oasis:names:tc:xacml:3.0:attribute-category:resource`

5127 `urn:oasis:names:tc:xacml:3.0:attribute-category:action`

5128 `urn:oasis:names:tc:xacml:3.0:attribute-category:environment`

5129 **Attributes** previously placed in the **Subject** section of a request are placed in an **attribute** category  
5130 which is identical of the **subject** category in XACML 2.0, as defined below. It is RECOMMENDED that  
5131 they are used to list **attributes** of **subjects** when authoring XACML 3.0 **policies** or requests.

5132 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity in a  
5133 request chain. If **subject** category is not specified in XACML 2.0, this is the default translation value.

5134 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

5135 This identifier indicates the system entity that will receive the results of the request (used when it is  
5136 distinct from the access-**subject**).

5137 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

5138 This identifier indicates a system entity through which the **access** request was passed. There may be  
5139 more than one. No means is provided to specify the order in which they passed the message.

5140 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

5141 This identifier indicates a system entity associated with a local or remote codebase that generated the  
5142 request. Corresponding **subject attributes** might include the URL from which it was loaded and/or the  
5143 identity of the code-signer. There may be more than one. No means is provided to specify the order in  
5144 which they processed the request.

5145 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

5146 This identifier indicates a system entity associated with the computer that initiated the **access** request.  
5147 An example would be an IPsec identity.

5148 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

### 5149 B.3 Data-types

5150 The following identifiers indicate data-types that are defined in Section A.2.

5151 `urn:oasis:names:tc:xacml:1.0:data-type:x500Name`.

5152 `urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name`

5153 `urn:oasis:names:tc:xacml:2.0:data-type:ipAddress`

5154 `urn:oasis:names:tc:xacml:2.0:data-type:dnsName`

5155 urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression  
5156 The following data-type identifiers are defined by XML Schema [XS].  
5157 <http://www.w3.org/2001/XMLSchema#string>  
5158 <http://www.w3.org/2001/XMLSchema#boolean>  
5159 <http://www.w3.org/2001/XMLSchema#integer>  
5160 <http://www.w3.org/2001/XMLSchema#double>  
5161 <http://www.w3.org/2001/XMLSchema#time>  
5162 <http://www.w3.org/2001/XMLSchema#date>  
5163 <http://www.w3.org/2001/XMLSchema#dateTime>  
5164 <http://www.w3.org/2001/XMLSchema#anyURI>  
5165 <http://www.w3.org/2001/XMLSchema#hexBinary>  
5166 <http://www.w3.org/2001/XMLSchema#base64Binary>  
5167 The following data-type identifiers correspond to the dayTimeDuration and yearMonthDuration data-types  
5168 defined in [XF] Sections 10.3.2 and 10.3.1, respectively.  
5169 <http://www.w3.org/2001/XMLSchema#dayTimeDuration>  
5170 <http://www.w3.org/2001/XMLSchema#yearMonthDuration>

## 5171 **B.4 Subject attributes**

5172 These identifiers indicate **attributes** of a **subject**. When used, it is RECOMMENDED that they appear  
5173 within an <Attributes> element of the request **context** with a **subject** category (see section B.2).  
5174 At most one of each of these **attributes** is associated with each **subject**. Each **attribute** associated with  
5175 authentication included within a single <Attributes> element relates to the same authentication event.  
5176 This identifier indicates the name of the **subject**.  
5177 urn:oasis:names:tc:xacml:1.0:subject:subject-id  
5178 This identifier indicates the security domain of the subject. It identifies the administrator and **policy** that  
5179 manages the name-space in which the **subject** id is administered.  
5180 urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier  
5181 This identifier indicates a public key used to confirm the **subject's** identity.  
5182 urn:oasis:names:tc:xacml:1.0:subject:key-info  
5183 This identifier indicates the time at which the **subject** was authenticated.  
5184 urn:oasis:names:tc:xacml:1.0:subject:authentication-time  
5185 This identifier indicates the method used to authenticate the **subject**.  
5186 urn:oasis:names:tc:xacml:1.0:subject:authentication-method  
5187 This identifier indicates the time at which the **subject** initiated the **access** request, according to the **PEP**.  
5188 urn:oasis:names:tc:xacml:1.0:subject:request-time  
5189 This identifier indicates the time at which the **subject's** current session began, according to the **PEP**.  
5190 urn:oasis:names:tc:xacml:1.0:subject:session-start-time  
5191 The following identifiers indicate the location where authentication credentials were activated.  
5192 This identifier indicates that the location is expressed as an IP address.  
5193 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address  
5194 The corresponding **attribute** SHALL be of data-type "<http://www.w3.org/2001/XMLSchema#string>".  
5195 This identifier indicates that the location is expressed as a DNS name.  
5196 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name  
5197 The corresponding **attribute** SHALL be of data-type "<http://www.w3.org/2001/XMLSchema#string>".

5198 Where a suitable **attribute** is already defined in LDAP [LDAP-1], [LDAP-2], the XACML identifier SHALL  
5199 be formed by adding the **attribute** name to the URI of the LDAP specification. For example, the **attribute**  
5200 name for the userPassword defined in the RFC 2256 SHALL be:  
5201 `http://www.ietf.org/rfc/rfc2256.txt#userPassword`

## 5202 B.5 Resource attributes

5203 These identifiers indicate **attributes** of the **resource**. When used, it is RECOMMENDED they appear  
5204 within the <Attributes> element of the request **context** with Category  
5205 `urn:oasis:names:tc:xacml:3.0:attribute-category:resource`.

5206 This **attribute** identifies the **resource** to which **access** is requested.

5207 `urn:oasis:names:tc:xacml:1.0:resource:resource-id`

5208 This **attribute** identifies the namespace of the top element(s) of the contents of the <Content> element.  
5209 In the case where the **resource** content is supplied in the request **context** and the **resource**  
5210 namespaces are defined in the **resource**, the **PEP** MAY provide this **attribute** in the request to indicate  
5211 the namespaces of the **resource** content. In this case there SHALL be one value of this **attribute** for  
5212 each unique namespace of the top level elements in the <Content> element. The type of the  
5213 corresponding **attribute** SHALL be “`http://www.w3.org/2001/XMLSchema#anyURI`”.

5214 `urn:oasis:names:tc:xacml:2.0:resource:target-namespace`

## 5215 B.6 Action attributes

5216 These identifiers indicate **attributes** of the **action** being requested. When used, it is RECOMMENDED  
5217 they appear within the <Attributes> element of the request **context** with Category  
5218 `urn:oasis:names:tc:xacml:3.0:attribute-category:action`.

5219 This **attribute** identifies the **action** for which **access** is requested.

5220 `urn:oasis:names:tc:xacml:1.0:action:action-id`

5221 Where the **action** is implicit, the value of the action-id **attribute** SHALL be

5222 `urn:oasis:names:tc:xacml:1.0:action:implied-action`

5223 This **attribute** identifies the namespace in which the action-id **attribute** is defined.

5224 `urn:oasis:names:tc:xacml:1.0:action:action-namespace`

## 5225 B.7 Environment attributes

5226 These identifiers indicate **attributes** of the **environment** within which the **decision request** is to be  
5227 evaluated. When used in the **decision request**, it is RECOMMENDED they appear in the  
5228 <Attributes> element of the request **context** with Category `urn:oasis:names:tc:xacml:3.0:attribute-`  
5229 `category:environment`.

5230 This identifier indicates the current time at the **context handler**. In practice it is the time at which the  
5231 request **context** was created. For this reason, if these identifiers appear in multiple places within a  
5232 <Policy> or <PolicySet>, then the same value SHALL be assigned to each occurrence in the  
5233 evaluation procedure, regardless of how much time elapses between the processing of the occurrences.

5234 `urn:oasis:names:tc:xacml:1.0:environment:current-time`

5235 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#time`”.

5236 `urn:oasis:names:tc:xacml:1.0:environment:current-date`

5237 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#date`”.

5238 `urn:oasis:names:tc:xacml:1.0:environment:current-dateTime`

5239 The corresponding **attribute** SHALL be of data-type “`http://www.w3.org/2001/XMLSchema#dateTime`”.

## 5240 **B.8 Status codes**

5241 The following status code values are defined.

5242 This identifier indicates success.

5243 urn:oasis:names:tc:xacml:1.0:status:ok

5244 This identifier indicates that all the **attributes** necessary to make a **policy decision** were not available (see Section 5.58).

5246 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

5247 This identifier indicates that some **attribute** value contained a syntax error, such as a letter in a numeric field.

5249 urn:oasis:names:tc:xacml:1.0:status:syntax-error

5250 This identifier indicates that an error occurred during **policy** evaluation. An example would be division by zero.

5252 urn:oasis:names:tc:xacml:1.0:status:processing-error

## 5253 **B.9 Combining algorithms**

5254 The deny-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5256 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides

5257 The deny-overrides **policy-combining algorithm** has the following value for the

5258 policyCombiningAlgId attribute:

5259 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides

5260 The permit-overrides **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5262 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides

5263 The permit-overrides **policy-combining algorithm** has the following value for the

5264 policyCombiningAlgId attribute:

5265 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides

5266 The first-applicable **rule-combining algorithm** has the following value for the ruleCombiningAlgId attribute:

5268 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable

5269 The first-applicable **policy-combining algorithm** has the following value for the

5270 policyCombiningAlgId attribute:

5271 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable

5272 The only-one-applicable-policy **policy-combining algorithm** has the following value for the

5273 policyCombiningAlgId attribute:

5274 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

5275 The ordered-deny-overrides **rule-combining algorithm** has the following value for the

5276 ruleCombiningAlgId attribute:

5277 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides

5278 The ordered-deny-overrides **policy-combining algorithm** has the following value for the

5279 policyCombiningAlgId attribute:

5280 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-

5281 overrides

5282 The ordered-permit-overrides **rule-combining algorithm** has the following value for the

5283 ruleCombiningAlgId attribute:

5284 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-  
5285 overrides

5286 The ordered-permit-overrides **policy-combining algorithm** has the following value for the  
5287 policyCombiningAlgId attribute:

5288 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-  
5289 overrides

5290 The deny-unless-permit **rule-combining algorithm** has the following value for the  
5291 policyCombiningAlgId attribute:

5292 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit

5293 The permit-unless-deny **rule-combining algorithm** has the following value for the  
5294 policyCombiningAlgId attribute:

5295 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny

5296 The deny-unless-permit **policy-combining algorithm** has the following value for the  
5297 policyCombiningAlgId attribute:

5298 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit

5299 The permit-unless-deny **policy-combining algorithm** has the following value for the  
5300 policyCombiningAlgId attribute:

5301 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny

5302 The legacy deny-overrides **rule-combining algorithm** has the following value for the  
5303 ruleCombiningAlgId attribute:

5304 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5305 The legacy deny-overrides **policy-combining algorithm** has the following value for the  
5306 policyCombiningAlgId attribute:

5307 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

5308 The legacy permit-overrides **rule-combining algorithm** has the following value for the  
5309 ruleCombiningAlgId attribute:

5310 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

5311 The legacy permit-overrides **policy-combining algorithm** has the following value for the  
5312 policyCombiningAlgId attribute:

5313 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

5314 The legacy ordered-deny-overrides **rule-combining algorithm** has the following value for the  
5315 ruleCombiningAlgId attribute:

5316 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides

5317 The legacy ordered-deny-overrides **policy-combining algorithm** has the following value for the  
5318 policyCombiningAlgId attribute:

5319 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-  
5320 overrides

5321 The legacy ordered-permit-overrides **rule-combining algorithm** has the following value for the  
5322 ruleCombiningAlgId attribute:

5323 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-  
5324 overrides

5325 The legacy ordered-permit-overrides **policy-combining algorithm** has the following value for the  
5326 policyCombiningAlgId attribute:

5327 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-  
5328 overrides

5329

5330

## C. Combining algorithms (normative)

5331 This section contains a description of the *rule-* and *policy-combining algorithms* specified by XACML.  
5332 Pseudo code is normative, descriptions in English are non-normative.

5333 The legacy *combining algorithms* are defined in previous versions of XACML, and are retained for  
5334 compatibility reasons. It is RECOMMENDED that the new *combining algorithms* are used instead of the  
5335 legacy *combining algorithms* for new use.

### C.1 Extended Indeterminate value

5337 Some combining algorithms are defined in terms of an extended set of "Indeterminate" values. For these  
5338 algorithms, the *PDP* MUST keep track of the extended set of "Indeterminate" values during *rule* and  
5339 *policy* combining. The extended set associated with the "Indeterminate" contains the potential effect  
5340 values which could have occurred if there would not have been an error causing the "Indeterminate". The  
5341 possible extended set "Indeterminate" values are

- 5342 • "Indeterminate{D}": an "Indeterminate" from a *policy* or *rule* which could have evaluated to "Deny",  
5343 but not "Permit"
- 5344 • "Indeterminate{P}": an "Indeterminate" from a *policy* or *rule* which could have evaluated to "Permit",  
5345 but not "Deny"
- 5346 • "Indeterminate{DP}": an "Indeterminate" from a *policy* or *rule* which could have evaluated to "Deny"  
5347 or "Permit".

5348 The combining algorithms which are defined in terms of the extended "Indeterminate" make use of the  
5349 additional information to allow for better treatment of errors in the algorithms.

5350 The following define the base cases for rule evaluation:

- 5351 • A *rule* which evaluates to "Indeterminate" and has Effect="Permit" results in an  
5352 "Indeterminate{P}".
- 5353 • A *rule* which evaluates to "Indeterminate" and has Effect="Deny" results in an "Indeterminate{D}".

### C.2 Deny-overrides

5355 This section defines the "Deny-overrides" *rule-combining algorithm* of a *policy* and *policy-combining*  
5356 *algorithm* of a *policy set*.

5357 This *combining algorithm* makes use of the extended "Indeterminate".

5358 The *rule combining algorithm* defined here has the following identifier:

5359 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides

5360 The *policy combining algorithm* defined here has the following identifier:

5361 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides

5362 The following is a non-normative informative description of this *combining algorithm*.

5363 The deny overrides *combining algorithm* is intended for those cases where a deny  
5364 decision should have priority over a permit decision. This algorithm has the following  
5365 behavior.

- 5366 1. If any decision is "Deny", the result is "Deny".
- 5367 2. Otherwise, if any decision is "Indeterminate{DP}", the result is "Indeterminate{DP}".
- 5368 3. Otherwise, if any decision is "Indeterminate{D}" and another decision is "Indeterminate{P} or  
5369 Permit, the result is "Indeterminate{DP}".
- 5370 4. Otherwise, if any decision is "Indeterminate{D}", the result is "Indeterminate{D}".
- 5371 5. Otherwise, if any decision is "Permit", the result is "Permit".

- 5372           6. Otherwise, if any decision is "Indeterminate{P}", the result is "Indeterminate{P}".  
 5373           7. Otherwise, the result is "NotApplicable".

5374 The following pseudo-code represents the normative specification of this **combining algorithm**.

```

5375 Decision denyOverridesCombiningAlgorithm(Decision[] decisions)
5376 {
5377     Boolean atLeastOneErrorD = false;
5378     Boolean atLeastOneErrorP = false;
5379     Boolean atLeastOneErrorDP = false;
5380     Boolean atLeastOnePermit = false;
5381     for( i=0 ; i < lengthOf(decisions) ; i++ )
5382     {
5383         Decision decision = decisions[i];
5384         if (decision == Deny)
5385         {
5386             return Deny;
5387         }
5388         if (decision == Permit)
5389         {
5390             atLeastOnePermit = true;
5391             continue;
5392         }
5393         if (decision == NotApplicable)
5394         {
5395             continue;
5396         }
5397         if (decision == Indeterminate{D})
5398         {
5399             atLeastOneErrorD = true;
5400             continue;
5401         }
5402         if (decision == Indeterminate{P})
5403         {
5404             atLeastOneErrorP = true;
5405             continue;
5406         }
5407         if (decision == Indeterminate{DP})
5408         {
5409             atLeastOneErrorDP = true;
5410             continue;
5411         }
5412     }
5413     if (atLeastOneErrorDP)
5414     {
5415         return Indeterminate{DP};
5416     }
5417     if (atLeastOneErrorD && (atLeastOneErrorP || atLeastOnePermit))
5418     {
5419         return Indeterminate{DP};
5420     }
5421     if (atLeastOneErrorD)
5422     {
5423         return Indeterminate{D};
5424     }
5425     if (atLeastOnePermit)
5426     {
5427         return Permit;
5428     }
5429     if (atLeastOneErrorP)
5430     {
5431         return Indeterminate{P};
5432     }
5433     return NotApplicable;
  
```

5434

```
}

```

5435 **Obligations** and **advice** shall be combined as described in Section 7.16.

### 5436 C.3 Ordered-deny-overrides

5437 The following specification defines the "Ordered-deny-overrides" **rule-combining algorithm** of a **policy**.

5438 The behavior of this algorithm is identical to that of the "Deny-overrides" **rule-combining**  
5439 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5440 match the order as listed in the **policy**.

5441 The **rule combining algorithm** defined here has the following identifier:

5442 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides

5443 The following specification defines the "Ordered-deny-overrides" **policy-combining algorithm** of a  
5444 **policy set**.

5445 The behavior of this algorithm is identical to that of the "Deny-overrides" **policy-combining**  
5446 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
5447 match the order as listed in the **policy set**.

5448 The **policy combining algorithm** defined here has the following identifier:

5449 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-  
5450 overrides

### 5451 C.4 Permit-overrides

5452 This section defines the "Permit-overrides" **rule-combining algorithm** of a **policy** and **policy-combining**  
5453 **algorithm** of a **policy set**.

5454 This **combining algorithm** makes use of the extended "Indeterminate".

5455 The **rule combining algorithm** defined here has the following identifier:

5456 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides

5457 The **policy combining algorithm** defined here has the following identifier:

5458 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides

5459 The following is a non-normative informative description of this combining algorithm.

5460 The permit overrides **combining algorithm** is intended for those cases where a permit  
5461 decision should have priority over a deny decision. This algorithm has the following  
5462 behavior.

- 5463 1. If any decision is "Permit", the result is "Permit".
- 5464 2. Otherwise, if any decision is "Indeterminate{DP}", the result is "Indeterminate{DP}".
- 5465 3. Otherwise, if any decision is "Indeterminate{P}" and another decision is  
5466 "Indeterminate{D} or Deny, the result is "Indeterminate{DP}".
- 5467 4. Otherwise, if any decision is "Indeterminate{P}", the result is "Indeterminate{P}".
- 5468 5. Otherwise, if decision is "Deny", the result is "Deny".
- 5469 6. Otherwise, if any decision is "Indeterminate{D}", the result is "Indeterminate{D}".
- 5470 7. Otherwise, the result is "NotApplicable".

5471 The following pseudo-code represents the normative specification of this **combining algorithm**.

```
5472 Decision permitOverridesCombiningAlgorithm(Decision[] decisions)
5473 {
5474     Boolean atLeastOneErrorD = false;
5475     Boolean atLeastOneErrorP = false;
5476     Boolean atLeastOneErrorDP = false;
```

```

5477 Boolean atLeastOneDeny = false;
5478 for( i=0 ; i < lengthOf(decisions) ; i++ )
5479 {
5480     Decision decision = decisions[i];
5481     if (decision == Deny)
5482     {
5483         atLeastOneDeny = true;
5484         continue;
5485     }
5486     if (decision == Permit)
5487     {
5488         return Permit;
5489     }
5490     if (decision == NotApplicable)
5491     {
5492         continue;
5493     }
5494     if (decision == Indeterminate{D})
5495     {
5496         atLeastOneErrorD = true;
5497         continue;
5498     }
5499     if (decision == Indeterminate{P})
5500     {
5501         atLeastOneErrorP = true;
5502         continue;
5503     }
5504     if (decision == Indeterminate{DP})
5505     {
5506         atLeastOneErrorDP = true;
5507         continue;
5508     }
5509 }
5510 if (atLeastOneErrorDP)
5511 {
5512     return Indeterminate{DP};
5513 }
5514 if (atLeastOneErrorP && (atLeastOneErrorD || atLeastOneDeny))
5515 {
5516     return Indeterminate{DP};
5517 }
5518 if (atLeastOneErrorP)
5519 {
5520     return Indeterminate{P};
5521 }
5522 if (atLeastOneDeny)
5523 {
5524     return Deny;
5525 }
5526 if (atLeastOneErrorD)
5527 {
5528     return Indeterminate{D};
5529 }
5530 return NotApplicable;
5531 }

```

5532 **Obligations** and **advice** shall be combined as described in Section 7.16.

## 5533 C.5 Ordered-permit-overrides

5534 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a **policy**.

5535 The behavior of this algorithm is identical to that of the “Permit-overrides” **rule-combining**  
5536 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5537 match the order as listed in the **policy**.

5538 The **rule combining algorithm** defined here has the following identifier:

5539 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-  
5540 overrides

5541 The following specification defines the “Ordered-permit-overrides” **policy-combining algorithm** of a  
5542 **policy set**.

5543 The behavior of this algorithm is identical to that of the “Permit-overrides” **policy-combining**  
5544 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
5545 match the order as listed in the **policy set**.

5546 The **policy combining algorithm** defined here has the following identifier:

5547 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-  
5548 overrides

## 5549 C.6 Deny-unless-permit

5550 This section defines the “Deny-unless-permit” **rule-combining algorithm** of a **policy** or **policy-**  
5551 **combining algorithm** of a **policy set**.

5552 The **rule combining algorithm** defined here has the following identifier:

5553 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit

5554 The **policy combining algorithm** defined here has the following identifier:

5555 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit

5556 The following is a non-normative informative description of this **combining algorithm**.

5557 The “Deny-unless-permit” **combining algorithm** is intended for those cases where a  
5558 permit decision should have priority over a deny decision, and an “Indeterminate” or  
5559 “NotApplicable” must never be the result. It is particularly useful at the top level in a  
5560 **policy** structure to ensure that a **PDP** will always return a definite “Permit” or “Deny”  
5561 result. This algorithm has the following behavior.

- 5562 1. If any decision is "Permit", the result is "Permit".
- 5563 2. Otherwise, the result is "Deny".

5564 The following pseudo-code represents the normative specification of this **combining algorithm**.

```
5565 Decision denyUnlessPermitCombiningAlgorithm(Decision[] decisions)  
5566 {  
5567     for( i=0 ; i < lengthOf(decisions) ; i++ )  
5568     {  
5569         if (decisions[i] == Permit)  
5570         {  
5571             return Permit;  
5572         }  
5573     }  
5574     return Deny;  
5575 }
```

5576 **Obligations** and **advice** shall be combined as described in Section 7.16.

## 5577 C.7 Permit-unless-deny

5578 This section defines the “Permit-unless-deny” **rule-combining algorithm** of a **policy** or **policy-**  
5579 **combining algorithm** of a **policy set**.

5580 The **rule combining algorithm** defined here has the following identifier:

5581 urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny

5582 The **policy combining algorithm** defined here has the following identifier:

5583 urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny

5584 The following is a non-normative informative description of this **combining algorithm**.

5585 The "Permit-unless-deny" **combining algorithm** is intended for those cases where a  
5586 deny decision should have priority over a permit decision, and an "Indeterminate" or  
5587 "NotApplicable" must never be the result. It is particularly useful at the top level in a  
5588 **policy** structure to ensure that a **PDP** will always return a definite "Permit" or "Deny"  
5589 result. This algorithm has the following behavior.

5590 1. If any decision is "Deny", the result is "Deny".

5591 2. Otherwise, the result is "Permit".

5592 The following pseudo-code represents the normative specification of this **combining algorithm**.

```
5593 Decision permitUnlessDenyCombiningAlgorithm(Decision[] decisions)
5594 {
5595     for( i=0 ; i < lengthOf(decisions) ; i++ )
5596     {
5597         if (decisions[i] == Deny)
5598         {
5599             return Deny;
5600         }
5601     }
5602     return Permit;
5603 }
```

5604 **Obligations** and **advice** shall be combined as described in Section 7.16.

## 5605 C.8 First-applicable

5606 This section defines the "First-applicable" **rule-combining algorithm** of a **policy** and **policy-combining**  
5607 **algorithm** of a **policy set**.

5608 The **rule combining algorithm** defined here has the following identifier:

5609 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable

5610 The following is a non-normative informative description of the "First-Applicable" **rule-combining**  
5611 **algorithm** of a **policy**.

5612 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a particular  
5613 **rule**, if the **target** matches and the **condition** evaluates to "True", then the evaluation of the  
5614 **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the result of the evaluation  
5615 of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected in the evaluation, if the  
5616 **target** evaluates to "False" or the **condition** evaluates to "False", then the next **rule** in the order  
5617 SHALL be evaluated. If no further **rule** in the order exists, then the **policy** SHALL evaluate to  
5618 "NotApplicable".

5619 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation SHALL  
5620 halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error status.

5621 The following pseudo-code represents the normative specification of this **rule-combining algorithm**.

```
5622 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule[] rules)
5623 {
5624     for( i = 0 ; i < lengthOf(rules) ; i++ )
5625     {
5626         Decision decision = evaluate(rules[i]);
5627         if (decision == Deny)
5628         {
5629             return Deny;
5630         }
5631     }
5632 }
```

```

5631     if (decision == Permit)
5632     {
5633         return Permit;
5634     }
5635     if (decision == NotApplicable)
5636     {
5637         continue;
5638     }
5639     if (decision == Indeterminate)
5640     {
5641         return Indeterminate;
5642     }
5643 }
5644 return NotApplicable;
5645 }

```

5646 The **policy combining algorithm** defined here has the following identifier:

5647 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable

5648 The following is a non-normative informative description of the "First-applicable" **policy-combining algorithm** of a **policy set**.

5650 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular **policy**, if  
5651 the **target** evaluates to "True" and the **policy** evaluates to a determinate value of "Permit" or  
5652 "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to the **effect** value of  
5653 that **policy**. For a particular **policy**, if the **target** evaluate to "False", or the **policy** evaluates to  
5654 "NotApplicable", then the next **policy** in the order SHALL be evaluated. If no further **policy** exists  
5655 in the order, then the **policy set** SHALL evaluate to "NotApplicable".

5656 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**, the  
5657 reference to the **policy** is considered invalid, or the **policy** itself evaluates to "Indeterminate",  
5658 then the evaluation of the **policy-combining algorithm** shall halt, and the **policy set** shall  
5659 evaluate to "Indeterminate" with an appropriate error status.

5660 The following pseudo-code represents the normative specification of this policy-combination algorithm.

```

5661 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy[] policies)
5662 {
5663     for( i = 0 ; i < lengthOf(policies) ; i++ )
5664     {
5665         Decision decision = evaluate(policies[i]);
5666         if(decision == Deny)
5667         {
5668             return Deny;
5669         }
5670         if(decision == Permit)
5671         {
5672             return Permit;
5673         }
5674         if (decision == NotApplicable)
5675         {
5676             continue;
5677         }
5678         if (decision == Indeterminate)
5679         {
5680             return Indeterminate;
5681         }
5682     }
5683     return NotApplicable;
5684 }

```

5685 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.16.

## 5686 C.9 Only-one-applicable

5687 This section defines the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

5688 The **policy combining algorithm** defined here has the following identifier:

5689 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable

5690 The following is a non-normative informative description of the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

5692 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".

5696 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**.

5698 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.

5701 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```
5702 Decision onlyOneApplicablePolicyPolicyCombiningAlogrithm(Policy[] policies)
5703 {
5704     Boolean          atLeastOne      = false;
5705     Policy           selectedPolicy = null;
5706     ApplicableResult appResult;
5707
5708     for ( i = 0; i < lengthOf(policies) ; i++ )
5709     {
5710         appResult = isApplicable(policies[i]);
5711
5712         if ( appResult == Indeterminate )
5713         {
5714             return Indeterminate;
5715         }
5716         if( appResult == Applicable )
5717         {
5718             if ( atLeastOne )
5719             {
5720                 return Indeterminate;
5721             }
5722             else
5723             {
5724                 atLeastOne      = true;
5725                 selectedPolicy = policies[i];
5726             }
5727         }
5728         if ( appResult == NotApplicable )
5729         {
5730             continue;
5731         }
5732     }
5733     if ( atLeastOne )
5734     {
5735         return evaluate(selectedPolicy);
5736     }
5737     else
5738     {
5739         return NotApplicable;
5740     }
5741 }
```

5742 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.16.

## 5743 C.10 Legacy Deny-overrides

5744 This section defines the legacy “Deny-overrides” *rule-combining algorithm* of a *policy* and *policy-*  
5745 *combining algorithm* of a *policy set*.

5746

5747 The *rule combining algorithm* defined here has the following identifier:

5748 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

5749 The following is a non-normative informative description of this combining algorithm.

5750 The “Deny-overrides” rule combining algorithm is intended for those cases where a deny  
5751 decision should have priority over a permit decision. This algorithm has the following  
5752 behavior.

- 5753 1. If any rule evaluates to "Deny", the result is "Deny".
- 5754 2. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is  
5755 "Indeterminate".
- 5756 3. Otherwise, if any rule evaluates to "Permit", the result is "Permit".
- 5757 4. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate", the result is  
5758 "Indeterminate".
- 5759 5. Otherwise, the result is "NotApplicable".

5760 The following pseudo-code represents the normative specification of this *rule-combining algorithm*.

```
5761 Decision denyOverridesRuleCombiningAlgorithm(Rule[] rules)
5762 {
5763     Boolean atLeastOneError = false;
5764     Boolean potentialDeny = false;
5765     Boolean atLeastOnePermit = false;
5766     for( i=0 ; i < lengthOf(rules) ; i++ )
5767     {
5768         Decision decision = evaluate(rules[i]);
5769         if (decision == Deny)
5770         {
5771             return Deny;
5772         }
5773         if (decision == Permit)
5774         {
5775             atLeastOnePermit = true;
5776             continue;
5777         }
5778         if (decision == NotApplicable)
5779         {
5780             continue;
5781         }
5782         if (decision == Indeterminate)
5783         {
5784             atLeastOneError = true;
5785
5786             if (effect(rules[i]) == Deny)
5787             {
5788                 potentialDeny = true;
5789             }
5790             continue;
5791         }
5792     }
5793     if (potentialDeny)
5794     {
5795         return Indeterminate;
5796     }
5797     if (atLeastOnePermit)
5798     {
```

```

5799     return Permit;
5800   }
5801   if (atLeastOneError)
5802   {
5803     return Indeterminate;
5804   }
5805   return NotApplicable;
5806 }

```

5807 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.16.

5808 The **policy combining algorithm** defined here has the following identifier:

5809 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

5810 The following is a non-normative informative description of this combining algorithm.

5811 The "Deny-overrides" policy combining algorithm is intended for those cases where a  
5812 deny decision should have priority over a permit decision. This algorithm has the  
5813 following behavior.

- 5814 1. If any policy evaluates to "Deny", the result is "Deny".
- 5815 2. Otherwise, if any policy evaluates to "Indeterminate", the result is "Deny".
- 5816 3. Otherwise, if any policy evaluates to "Permit", the result is "Permit".
- 5817 4. Otherwise, the result is "NotApplicable".

5818 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```

5819 Decision denyOverridesPolicyCombiningAlgorithm(Policy[] policies)
5820 {
5821   Boolean atLeastOnePermit = false;
5822   for( i=0 ; i < lengthOf(policies) ; i++ )
5823   {
5824     Decision decision = evaluate(policies[i]);
5825     if (decision == Deny)
5826     {
5827       return Deny;
5828     }
5829     if (decision == Permit)
5830     {
5831       atLeastOnePermit = true;
5832       continue;
5833     }
5834     if (decision == NotApplicable)
5835     {
5836       continue;
5837     }
5838     if (decision == Indeterminate)
5839     {
5840       return Deny;
5841     }
5842   }
5843   if (atLeastOnePermit)
5844   {
5845     return Permit;
5846   }
5847   return NotApplicable;
5848 }

```

5849 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.16.

## 5850 C.11 Legacy Ordered-deny-overrides

5851 The following specification defines the legacy "Ordered-deny-overrides" **rule-combining algorithm** of a  
5852 **policy**.

5853 The behavior of this algorithm is identical to that of the “Deny-overrides” **rule-combining**  
5854 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5855 match the order as listed in the **policy**.

5856 The **rule combining algorithm** defined here has the following identifier:

5857 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides

5858 The following specification defines the legacy “Ordered-deny-overrides” **policy-combining algorithm** of  
5859 a **policy set**.

5860 The behavior of this algorithm is identical to that of the “Deny-overrides” **policy-combining**  
5861 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
5862 match the order as listed in the **policy set**.

5863 The **rule combining algorithm** defined here has the following identifier:

5864 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-  
5865 overrides

## 5866 C.12 Legacy Permit-overrides

5867 This section defines the legacy “Permit-overrides” **rule-combining algorithm** of a **policy** and **policy-**  
5868 **combining algorithm** of a **policy set**.

5869 The **rule combining algorithm** defined here has the following identifier:

5870 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

5871 The following is a non-normative informative description of this combining algorithm.

5872 The “Permit-overrides” rule combining algorithm is intended for those cases where a  
5873 permit decision should have priority over a deny decision. This algorithm has the  
5874 following behavior.

- 5875 1. If any rule evaluates to "Permit", the result is "Permit".
- 5876 2. Otherwise, if any rule having Effect="Permit" evaluates to "Indeterminate" the result is  
5877 "Indeterminate".
- 5878 3. Otherwise, if any rule evaluates to "Deny", the result is "Deny".
- 5879 4. Otherwise, if any rule having Effect="Deny" evaluates to "Indeterminate", the result is  
5880 "Indeterminate".
- 5881 5. Otherwise, the result is "NotApplicable".

5882 The following pseudo-code represents the normative specification of this **rule-combining algorithm**.

```
5883 Decision permitOverridesRuleCombiningAlgorithm(Rule[] rules)
5884 {
5885     Boolean atLeastOneError = false;
5886     Boolean potentialPermit = false;
5887     Boolean atLeastOneDeny = false;
5888     for( i=0 ; i < lengthOf(rules) ; i++ )
5889     {
5890         Decision decision = evaluate(rules[i]);
5891         if (decision == Deny)
5892         {
5893             atLeastOneDeny = true;
5894             continue;
5895         }
5896         if (decision == Permit)
5897         {
5898             return Permit;
5899         }
5900         if (decision == NotApplicable)
5901         {
5902             continue;
5903         }
5904     }
5905 }
```

```

5904     if (decision == Indeterminate)
5905     {
5906         atLeastOneError = true;
5907
5908         if (effect(rules[i]) == Permit)
5909         {
5910             potentialPermit = true;
5911         }
5912         continue;
5913     }
5914 }
5915 if (potentialPermit)
5916 {
5917     return Indeterminate;
5918 }
5919 if (atLeastOneDeny)
5920 {
5921     return Deny;
5922 }
5923 if (atLeastOneError)
5924 {
5925     return Indeterminate;
5926 }
5927 return NotApplicable;
5928 }

```

5929 **Obligations** and **advice** of the individual **rules** shall be combined as described in Section 7.16.

5930 The **policy combining algorithm** defined here has the following identifier:

5931 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

5932 The following is a non-normative informative description of this combining algorithm.

5933 The "Permit-overrides" policy combining algorithm is intended for those cases where a  
5934 permit decision should have priority over a deny decision. This algorithm has the  
5935 following behavior.

- 5936 1. If any policy evaluates to "Permit", the result is "Permit".
- 5937 2. Otherwise, if any policy evaluates to "Deny", the result is "Deny".
- 5938 3. Otherwise, if any policy evaluates to "Indeterminate", the result is "Indeterminate".
- 5939 4. Otherwise, the result is "NotApplicable".

5940 The following pseudo-code represents the normative specification of this **policy-combining algorithm**.

```

5941 Decision permitOverridesPolicyCombiningAlgorithm(Policy[] policies)
5942 {
5943     Boolean atLeastOneError = false;
5944     Boolean atLeastOneDeny = false;
5945     for( i=0 ; i < lengthOf(policies) ; i++ )
5946     {
5947         Decision decision = evaluate(policies[i]);
5948         if (decision == Deny)
5949         {
5950             atLeastOneDeny = true;
5951             continue;
5952         }
5953         if (decision == Permit)
5954         {
5955             return Permit;
5956         }
5957         if (decision == NotApplicable)
5958         {
5959             continue;
5960         }
5961         if (decision == Indeterminate)

```

```
5962     {
5963         atLeastOneError = true;
5964         continue;
5965     }
5966 }
5967 if (atLeastOneDeny)
5968 {
5969     return Deny;
5970 }
5971 if (atLeastOneError)
5972 {
5973     return Indeterminate;
5974 }
5975 return NotApplicable;
5976 }
```

5977 **Obligations** and **advice** of the individual **policies** shall be combined as described in Section 7.16.

### 5978 **C.13 Legacy Ordered-permit-overrides**

5979 The following specification defines the legacy "Ordered-permit-overrides" **rule-combining algorithm** of a  
5980 **policy**.

5981 The behavior of this algorithm is identical to that of the "Permit-overrides" **rule-combining**  
5982 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
5983 match the order as listed in the **policy**.

5984 The **rule combining algorithm** defined here has the following identifier:

5985 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-  
5986 overrides

5987 The following specification defines the legacy "Ordered-permit-overrides" **policy-combining algorithm** of  
5988 a **policy set**.

5989 The behavior of this algorithm is identical to that of the "Permit-overrides" **policy-combining**  
5990 **algorithm** with one exception. The order in which the collection of **policies** is evaluated SHALL  
5991 match the order as listed in the **policy set**.

5992 The **policy combining algorithm** defined here has the following identifier:

5993 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-  
5994 overrides

5995

---

5996

## D. Acknowledgements

5997 The following individuals have participated in the creation of this specification and are gratefully  
5998 acknowledged:

5999

6000 Anil Saldhana  
6001 Anil Tappetla  
6002 Anne Anderson  
6003 Anthony Nadalin  
6004 Bill Parducci  
6005 Craig Forster  
6006 David Chadwick  
6007 David Staggs  
6008 Dilli Arumugam  
6009 Duane DeCouteau  
6010 Erik Rissanen  
6011 Gareth Richards  
6012 Hal Lockhart  
6013 Jan Herrmann  
6014 John Tolbert  
6015 Ludwig Seitz  
6016 Michiharu Kudo  
6017 Naomaru Itoi  
6018 Paul Tyson  
6019 Prateek Mishra  
6020 Rich Levinson  
6021 Ronald Jacobson  
6022 Seth Proctor  
6023 Sridhar Muppidi  
6024 Tim Moses  
6025 Vernon Murdoch

6026

## E. Revision History

6027 [optional; should not be included in OASIS Standards]

6028

Revision	Date	Editor	Changes Made
WD 05	10 Oct 2007	Erik Rissanen	Convert to new OASIS template. Fixed typos and errors.
WD 06	18 May 2008	Erik Rissanen	<p>Added missing MaxDelegationDepth in schema fragments.</p> <p>Added missing urn:oasis:names:tc:xacml:1.0:resource:xpath identifier.</p> <p>Corrected typos on xpaths in the example policies.</p> <p>Removed use of xpointer in the examples.</p> <p>Made the &lt;Content&gt; element the context node of all xpath expressions and introduced categorization of XPath expressions so they point to a specific &lt;Content&gt; element.</p> <p>Added &lt;Content&gt; element to the policy issuer.</p> <p>Added description of the &lt;PolicyIssuer&gt; element.</p> <p>Updated the schema figure in the introduction to reflect the new AllOf/AnyOf schema.</p> <p>Remove duplicate &lt;CombinerParameters&gt; element in the &lt;Policy&gt; element in the schema.</p> <p>Removed default attributes in the schema. (Version in &lt;Policy(Set)&gt; and MustBePresent in &lt;AttributeDesignator&gt; in &lt;AttributeSelector&gt;)</p> <p>Removed references in section 7.3 to the &lt;Condition&gt; element having a FunctionId attribute.</p> <p>Fixed typos in data type URIs in section A.3.7.</p>
WD 07	3 Nov 2008	Erik Rissanen	<p>Fixed "...:data-types:..." typo in conformace section.</p> <p>Removed XML default attribute for IncludeInResult for element &lt;Attribute&gt;. Also added this attribute in the associated schema file.</p> <p>Removed description of non-existing XML attribute "ResourceId" from the element &lt;Result&gt;.</p> <p>Moved the urn:oasis:names:tc:xacml:3.0:function:access-permitted function into here from the delegation profile.</p>

			<p>Updated the daytime and yearmonth duration data types to the W3C defined identifiers.</p> <p>Added &lt;Description&gt; to &lt;Apply&gt;.</p> <p>Added XPath versioning to the request.</p> <p>Added security considerations about denial service and the access-permitted function.</p> <p>Changed &lt;Target&gt; matching so NoMatch has priority over Indeterminate.</p> <p>Fixed multiple typos in identifiers.</p> <p>Lower case incorrect use of "MAY".</p> <p>Misc minor typos.</p> <p>Removed whitespace in example attributes.</p> <p>Removed an incorrect sentence about higher order functions in the definition of the &lt;Function&gt; element.</p> <p>Clarified evaluation of empty or missing targets.</p> <p>Use Unicode codepoint collation for string comparisons.</p> <p>Support multiple arguments in multiply functions.</p> <p>Define Indeterminate result for overflow in integer to double conversion.</p> <p>Simplified descriptions of deny/permit overrides algorithms.</p> <p>Add ipAddress and dnsName into conformance section.</p> <p>Don't refer to IEEE 754 for integer arithmetic.</p> <p>Rephrase indeterminate result for arithmetic functions.</p> <p>Fix typos in examples.</p> <p>Clarify Match evaluation and drop list of example functions which can be used in a Match.</p> <p>Added behavior for circular policy/variable references.</p> <p>Fix obligation enforcement so it refers to PEP bias.</p> <p>Added Version xml attribute to the example policies.</p> <p>Remove requirement for PDP to check the target-namespace resource attribute.</p> <p>Added policy identifier list to the response/request.</p> <p>Added statements about Unicode normalization.</p> <p>Clarified definitions of string functions.</p>
--	--	--	---

			<p>Added new string functions.</p> <p>Added section on Unicode security issues.</p>
WD 08	5 Feb 2009	Erik Rissanen	<p>Updated Unicode normalization section according to suggestion from W3C working group.</p> <p>Set union functions now may take more than two arguments.</p> <p>Made obligation parameters into runtime expressions.</p> <p>Added new combining algorithms</p> <p>Added security consideration about policy id collisions.</p> <p>Added the &lt;Advice&gt; feature</p> <p>Made obligations mandatory (per the 19<sup>th</sup> Dec 2008 decision of the TC)</p> <p>Made obligations/advice available in rules</p> <p>Changed wording about deprecation</p>
WD 09			<p>Clarified wording about normative/informative in the combining algorithms section.</p> <p>Fixed duplicate variable in comb.algs and cleaned up variable names.</p> <p>Updated the schema to support the new multiple request scheme.</p>
WD 10	19 Mar 2009	Erik Rissanen	<p>Fixed schema for &lt;Request&gt;</p> <p>Fixed typos.</p> <p>Added optional Category to AttributeAssignments in obligations/advice.</p>
WD 11		Erik Rissanen	<p>Cleanups courtesy of John Tolbert.</p> <p>Added Issuer XML attribute to &lt;AttributeAssignment&gt;</p> <p>Fix the XPath expressions in the example policies and requests</p> <p>Fix inconsistencies in the conformance tables.</p> <p>Editorial cleanups.</p>
WD 12	16 Nov 2009	Erik Rissanen	<p>(Now working draft after public review of CD 1)</p> <p>Fix typos</p> <p>Allow element selection in attribute selector.</p> <p>Improve consistency in the use of the terms obligation, advice, and advice/obligation expressions and where they can appear.</p> <p>Fixed inconsistency in PEP bias between sections 5.1 and 7.2.</p> <p>Clarified text in overview of combining algorithms.</p> <p>Relaxed restriction on matching in xpath-node-</p>

			<p>match function.</p> <p>Remove note about XPath expert review.</p> <p>Removed obsolete resource:xpath identifier.</p> <p>Updated reference to XML spec.</p> <p>Defined error behavior for string-substring and uri-substring functions.</p> <p>Reversed the order of the arguments for the following functions: string-starts-with, uri-starts-with, string-ends-with, uri-ends-with, string-contains and uri-contains</p> <p>Renamed functions:</p> <ul style="list-style-type: none"> <li>• uri-starts-with to anyURI-starts-with</li> <li>• uri-ends-with to anyURI-ends-with</li> <li>• uri-contains to anyURI-contains</li> <li>• uri-substring to anyURI-substring</li> </ul> <p>Removed redundant occurrence indicators from RequestType.</p> <p>Don't use "...:os" namespace in examples since this is still just "...wd-12".</p> <p>Added missing MustBePresent and Version XML attributes in example policies.</p> <p>Added missing ReturnPolicyIdList and IncludeInResult XML attributes in example requests.</p> <p>Clarified error behavior in obligation/advice expressions.</p> <p>Allow bags in attribute assignment expressions.</p> <p>Use the new daytimeduration and yearmonthduration identifiers consistently.</p>
WD 13	14 Dec 2009	Erik Rissanen	<p>Fix small inconsistency in number of arguments to the multiply function.</p> <p>Generalize higher order bag functions.</p> <p>Add ContextSelectorId to attribute selector.</p> <p>Use &lt;Policy(Set)IdReference&gt; in &lt;PolicyIdList&gt;.</p> <p>Fix typos and formatting issues.</p> <p>Make the conformance section clearly reference the functional requirements in the spec.</p> <p>Conformance tests are no longer hosted by Sun.</p>
WD 14	17 Dec 2009	Erik Rissanen	Update acknowledgments
WD 15		Erik Rissanen	<p>Replace DecisionCombiningAlgorithm with a simple Boolean for CombinedDecision.</p> <p>Restrict &lt;Content&gt; to a single child element</p>

			and update the <AttributeSelector> and XPathExpression data type accordingly.
WD 16	12 Jan 2010	Erik Rissanen	Updated cross references Fix typos and minor inconsistencies. Simplify schema of <PolicyIdentifierList> Refactor some of the text to make it easier to understand. Update acknowledgments
WD 17	8 Mar 2010	Erik Rissanen	Updated cross references. Fixed OASIS style issues.
WD 18	23 Jun 2010	Erik Rissanen	Fixed typos in examples. Fixed typos in schema fragments.

6029

6030