



1

2

# eXtensible Access Control Markup Language (XACML) Version 1.1

3

4

## Committee Specification, 24 July 2003

5

Document identifier: cs-xacml-specification-1.1.pdf

6

Location: <http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf>

7

Send comments to: [xacml-comment@lists.oasis-open.org](mailto:xacml-comment@lists.oasis-open.org)

8

Editors:

9

Simon Godik, Overxeer

10

Tim Moses, Entrust

11

Committee members:

12

Anne Anderson, Sun Microsystems

13

Bill Parducci, Overxeer

14

Carlisle Adams, Entrust

15

Don Flinn, Quadrasis

16

Gerald Brose, Xtradyne

17

Hal Lockhart, Entegry

18

Konstantin Beznosov, Quadrasis

19

Michiharu Kudo, IBM

20

Polar Humenn, Self

21

Simon Godik, Overxeer

22

Steve Andersen, OpenNetwork

23

Steve Crocker, Pervasive Security Systems

24

Tim Moses, Entrust

25

26

Abstract:

27

This specification defines an XML schema for an extensible access-control policy

28

language.

29

30

Status:

31

This version of the specification is an OASIS standard.

32

If you are on the [xacml@lists.oasis-open.org](mailto:xacml@lists.oasis-open.org) list for committee members, send comments

33

there. If you are not on that list, subscribe to the [xacml-comment@lists.oasis-open.org](mailto:xacml-comment@lists.oasis-open.org) list

34 and send comments there. To subscribe, send an email message to [xacml-comment-](mailto:xacml-comment-request@lists.oasis-open.org)  
35 [request@lists.oasis-open.org](mailto:xacml-comment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

36

37 Copyright (C) OASIS Open 2003. All Rights Reserved.

## 38 Table of contents

39	1.	Introduction (non-normative) .....	10
40	1.1.	Glossary .....	10
41	1.1.1	Preferred terms .....	10
42	1.1.2	Related terms .....	11
43	1.2.	Notation .....	12
44	1.3.	Schema organization and namespaces .....	12
45	2.	Background (non-normative) .....	13
46	2.1.	Requirements .....	13
47	2.2.	Rule and policy combining .....	14
48	2.3.	Combining algorithms .....	14
49	2.4.	Multiple subjects .....	15
50	2.5.	Policies based on subject and resource attributes .....	15
51	2.6.	Multi-valued attributes .....	15
52	2.7.	Policies based on resource contents .....	16
53	2.8.	Operators .....	16
54	2.9.	Policy distribution .....	17
55	2.10.	Policy indexing .....	17
56	2.11.	Abstraction layer .....	17
57	2.12.	Actions performed in conjunction with enforcement .....	18
58	3.	Models (non-normative) .....	18
59	3.1.	Data-flow model .....	18
60	3.2.	XACML context .....	20
61	3.3.	Policy language model .....	20
62	3.3.1	Rule .....	21
63	3.3.2	Policy .....	23
64	3.3.3	Policy set .....	24
65	4.	Examples (non-normative) .....	25
66	4.1.	Example one .....	25
67	4.1.1	Example policy .....	25
68	4.1.2	Example request context .....	27
69	4.1.3	Example response context .....	28
70	4.2.	Example two .....	28
71	4.2.1	Example medical record instance .....	29
72	4.2.2	Example request context .....	30
73	4.2.3	Example plain-language rules .....	32

74	4.2.4	Example XACML rule instances .....	32
75	5.	Policy syntax (normative, with the exception of the schema fragments) .....	46
76	5.1.	Element <PolicySet> .....	46
77	5.2.	Element <Description> .....	47
78	5.3.	Element <PolicySetDefaults> .....	47
79	5.4.	Element <XPathVersion> .....	48
80	5.5.	Element <Target> .....	48
81	5.6.	Element <Subjects>.....	49
82	5.7.	Element <Subject>.....	49
83	5.8.	Element <AnySubject> .....	49
84	5.9.	Element <SubjectMatch>.....	49
85	5.10.	Element <Resources> .....	50
86	5.11.	Element <Resource> .....	50
87	5.12.	Element <AnyResource>.....	51
88	5.13.	Element <ResourceMatch> .....	51
89	5.14.	Element <Actions>.....	52
90	5.15.	Element <Action>.....	52
91	5.16.	Element <AnyAction> .....	52
92	5.17.	Element <ActionMatch>.....	52
93	5.18.	Element <PolicySetIdReference>.....	53
94	5.19.	Element <PolicyIdReference> .....	53
95	5.20.	Element <Policy> .....	53
96	5.21.	Element <PolicyDefaults>.....	55
97	5.22.	Element <Rule> .....	55
98	5.23.	Simple type EffectType.....	56
99	5.24.	Element <Condition> .....	56
100	5.25.	Element <Apply>.....	56
101	5.26.	Element <Function>.....	57
102	5.27.	Complex type AttributeDesignatorType .....	57
103	5.28.	Element <SubjectAttributeDesignator> .....	58
104	5.29.	Element <ResourceAttributeDesignator> .....	59
105	5.30.	Element <ActionAttributeDesignator> .....	60
106	5.31.	Element <EnvironmentAttributeDesignator> .....	60
107	5.32.	Element <AttributeSelector>.....	61
108	5.33.	Element <AttributeValue>.....	62
109	5.34.	Element <Obligations> .....	63
110	5.35.	Element <Obligation> .....	63

111	5.36.	Element <AttributeAssignment> .....	64
112	6.	Context syntax (normative with the exception of the schema fragments) .....	64
113	6.1.	Element <Request> .....	64
114	6.2.	Element <Subject>.....	65
115	6.3.	Element <Resource> .....	66
116	6.4.	Element <ResourceContent> .....	66
117	6.5.	Element <Action>.....	67
118	6.6.	Element <Environment> .....	67
119	6.7.	Element <Attribute> .....	67
120	6.8.	Element <AttributeValue>.....	68
121	6.9.	Element <Response> .....	68
122	6.10.	Element <Result> .....	69
123	6.11.	Element <Decision>.....	70
124	6.12.	Element <Status> .....	70
125	6.13.	Element <StatusCode>.....	71
126	6.14.	Element <StatusMessage>.....	71
127	6.15.	Element <StatusDetail> .....	71
128	7.	Functional requirements (normative) .....	72
129	7.1.	Policy enforcement point .....	72
130	7.2.	Base policy.....	72
131	7.3.	Target evaluation .....	73
132	7.4.	Condition evaluation .....	73
133	7.5.	Rule evaluation .....	73
134	7.6.	Policy evaluation.....	73
135	7.7.	Policy Set evaluation .....	74
136	7.8.	Hierarchical resources .....	75
137	7.9.	Attributes.....	76
138	7.9.1.	Attribute Matching.....	76
139	7.9.2.	Attribute Retrieval .....	76
140	7.9.3.	Environment Attributes.....	77
141	7.10.	Authorization decision.....	77
142	7.11.	Obligations .....	77
143	7.12.	Unsupported functionality .....	78
144	7.13.	Syntax and type errors.....	78
145	8.	XACML extensibility points (non-normative) .....	78
146	8.1.	Extensible XML attribute types .....	78
147	8.2.	Structured attributes.....	79

148	9.	Security and privacy considerations (non-normative)	79
149	9.1.	Threat model	79
150	9.1.1.	Unauthorized disclosure	80
151	9.1.2.	Message replay	80
152	9.1.3.	Message insertion	80
153	9.1.4.	Message deletion	80
154	9.1.5.	Message modification	80
155	9.1.6.	NotApplicable results	81
156	9.1.7.	Negative rules	81
157	9.2.	Safeguards	82
158	9.2.1.	Authentication	82
159	9.2.2.	Policy administration	82
160	9.2.3.	Confidentiality	82
161	9.2.4.	Policy integrity	83
162	9.2.5.	Policy identifiers	83
163	9.2.6.	Trust model	84
164	9.2.7.	Privacy	84
165	10.	Conformance (normative)	84
166	10.1.	Introduction	84
167	10.2.	Conformance tables	84
168	10.2.1.	Schema elements	85
169	10.2.2.	Identifier Prefixes	86
170	10.2.3.	Algorithms	86
171	10.2.4.	Status Codes	86
172	10.2.5.	Attributes	87
173	10.2.6.	Identifiers	87
174	10.2.7.	Data-types	87
175	10.2.8.	Functions	88
176	11.	References	92
177	Appendix A.	Standard data-types, functions and their semantics (normative)	94
178	A.1.	Introduction	94
179	A.2.	Primitive types	94
180	A.3.	Structured types	95
181	A.4.	Representations	95
182	A.5.	Bags	96
183	A.6.	Expressions	96
184	A.7.	Element <AttributeValue>	97

185	A.8. Elements <AttributeDesignator> and <AttributeSelector> .....	97
186	A.9. Element <Apply> .....	97
187	A.10. Element <Condition> .....	97
188	A.11. Element <Function> .....	98
189	A.12. Matching elements .....	98
190	<u>A.13. Arithmetic evaluation .....</u>	<u>99</u>
191	<u>A.14. XACML standard functions.....</u>	<u>100</u>
192	<u>A14.1 Equality predicates.....</u>	<u>100</u>
193	<u>A14.2 Arithmetic functions.....</u>	<u>102</u>
194	<u>A14.3 String conversion functions.....</u>	<u>103</u>
195	<u>A14.4 Numeric data-type conversion functions.....</u>	<u>103</u>
196	<u>A14.5 Logical functions .....</u>	<u>103</u>
197	<u>A14.6 Arithmetic comparison functions.....</u>	<u>104</u>
198	<u>A14.7 Date and time arithmetic functions .....</u>	<u>105</u>
199	<u>A14.8 Non-numeric comparison functions .....</u>	<u>106</u>
200	<u>A14.9 Bag functions .....</u>	<u>108</u>
201	<u>A14.10 Set functions .....</u>	<u>109</u>
202	<u>A14.11 Higher-order bag functions .....</u>	<u>110</u>
203	<u>A14.12 Special match functions .....</u>	<u>117</u>
204	<u>A14.13 XPath-based functions.....</u>	<u>118</u>
205	<u>A14.14 Extension functions and primitive types.....</u>	<u>118</u>
206	<u>Appendix B. XACML identifiers (normative) .....</u>	<u>119</u>
207	<u>B.1. XACML namespaces .....</u>	<u>119</u>
208	<u>B.2. Access subject categories.....</u>	<u>119</u>
209	<u>B.3. XACML functions .....</u>	<u>119</u>
210	<u>B.4. Data-types .....</u>	<u>119</u>
211	<u>B.5. Subject attributes .....</u>	<u>120</u>
212	<u>B.6. Resource attributes .....</u>	<u>121</u>
213	<u>B.7. Action attributes .....</u>	<u>121</u>
214	<u>B.8. Environment attributes .....</u>	<u>122</u>
215	<u>B.9. Status codes .....</u>	<u>122</u>
216	<u>B.10. Combining algorithms.....</u>	<u>122</u>
217	<u>Appendix C. Combining algorithms (normative) .....</u>	<u>124</u>
218	<u>C.1. Deny-overrides. ....</u>	<u>124</u>
219	<u>C.2. Ordered-deny-overrides (non-normative).....</u>	<u>126</u>
220	<u>C.3. Permit-overrides .....</u>	<u>126</u>
221	<u>C.4. Ordered-permit-overrides (non-normative).....</u>	<u>128</u>

222	C.5. First-applicable .....	128
223	C.6. Only-one-applicable.....	130
224	Appendix D. Acknowledgments .....	132
225	Appendix E. Revision history .....	133
226	Appendix F. Notices.....	134
227		

228 **Errata**

229 Errata can be found at the following location:

230 <http://www.oasis-open.org/committees/xacml/repository/errata-001.pdf>

---

## 232 1. Introduction (non-normative)

### 233 1.1. Glossary

#### 234 1.1.1 Preferred terms

235 **Access** - Performing an *action*

236 **Access control** - Controlling *access* in accordance with a *policy*

237 **Action** - An operation on a *resource*

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

240 **Attribute** - Characteristic of a *subject*, *resource*, *action* or *environment* that may be referenced  
241 in a *predicate* or *target*

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

245 **Bag** – An unordered collection of values, in which there may be duplicate values

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

248 **Conjunctive sequence** - a sequence of boolean elements combined using the logical 'AND'  
249 operation

250 **Context** - The canonical representation of a *decision request* and an *authorization decision*

251 **Context handler** - The system entity that converts *decision requests* in the native request format  
252 to the XACML canonical form and converts *authorization decisions* in the XACML canonical form  
253 to the native response format

254 **Decision** – The result of evaluating a *rule*, *policy* or *policy set*

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

256 **Disjunctive sequence** - a sequence of boolean elements combined using the logical 'OR'  
257 operation

258 **Effect** - The intended consequence of a satisfied *rule* (either "Permit" or "Deny")

259 **Environment** - The set of *attributes* that are relevant to an *authorization decision* and are  
260 independent of a particular *subject*, *resource* or *action*

261 **Obligation** - An operation specified in a **policy** or **policy set** that should be performed in  
262 conjunction with the enforcement of an **authorization decision**

263 **Policy** - A set of **rules**, an identifier for the **rule-combining algorithm** and (optionally) a set of  
264 **obligations**. May be a component of a **policy set**

265 **Policy administration point (PAP)** - The system entity that creates a **policy** or **policy set**

266 **Policy-combining algorithm** - The procedure for combining the **decision** and **obligations** from  
267 multiple **policies**

268 **Policy decision point (PDP)** - The system entity that evaluates **applicable policy** and renders an  
269 **authorization decision**

270 **Policy enforcement point (PEP)** - The system entity that performs **access control**, by making  
271 **decision requests** and enforcing **authorization decisions**

272 **Policy information point (PIP)** - The system entity that acts as a source of **attribute** values

273 **Policy set** - A set of **policies**, other **policy sets**, a **policy-combining algorithm** and (optionally) a  
274 set of **obligations**. May be a component of another **policy set**

275 **Predicate** - A statement about **attributes** whose truth can be evaluated

276 **Resource** - Data, service or system component

277 **Rule** - A **target**, an **effect** and a **condition**. A component of a **policy**

278 **Rule-combining algorithm** - The procedure for combining **decisions** from multiple **rules**

279 **Subject** - An actor whose **attributes** may be referenced by a **predicate**

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

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

### 1.1.2 Related terms

289

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

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

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

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

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

297

## 1.2. Notation

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

300 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",  
301 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be  
302 interpreted as described in IETF RFC 2119 [RFC2119]

303 *"they MUST only be used where it is actually required for interoperability or to limit*  
304 *behavior which has potential for causing harm (e.g., limiting retransmissions)"*

305 These keywords are thus capitalized when used to unambiguously specify requirements over  
306 protocol and application features and behavior that affect the interoperability and security of  
307 implementations. When these words are not capitalized, they are meant in their natural-language  
308 sense.

309 Listings of XACML schemas appear like this.

310

311 Example code listings appear like this.

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

- 315 • The prefix `xacml:` stands for the XACML policy namespace.
- 316 • The prefix `xacml-context:` stands for the XACML context namespace.
- 317 • The prefix `ds:` stands for the W3C XML Signature namespace [DS].
- 318 • The prefix `xs:` stands for the W3C XML Schema namespace [XS].
- 319 • The prefix `xf:` stands for the XQuery 1.0 and XPath 2.0 Function and Operators  
320 specification namespace [XF].

321 This specification uses the following typographical conventions in text: `<XACMLElement>`,  
322 `<ns:ForeignElement>`, `Attribute`, **Datatype**, `OtherCode`. Terms in *italic bold-face* are  
323 intended to have the meaning defined in the Glossary.

## 1.3. Schema organization and namespaces

325 The XACML policy syntax is defined in a schema associated with the following XML namespace:

326 `urn:oasis:names:tc:xacml:1.0:policy`

327 The XACML context syntax is defined in a schema associated with the following XML namespace:

328 `urn:oasis:names:tc:xacml:1.0:context`

329 The XML Signature [DS] is imported into the XACML schema and is associated with the following  
330 XML namespace:

331 `http://www.w3.org/2000/09/xmldsig#`

332

## 2. Background (non-normative)

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

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

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

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

### 2.1. Requirements

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

- 361 • To provide a method for combining individual **rules** and **policies** into a single **policy set** that  
362 applies to a particular **decision request**.
- 363 • To provide a method for flexible definition of the procedure by which **rules** and **policies** are  
364 combined.
- 365 • To provide a method for dealing with multiple **subjects** acting in different capacities.
- 366 • To provide a method for basing an **authorization decision** on **attributes** of the **subject** and  
367 **resource**.
- 368 • To provide a method for dealing with multi-valued **attributes**.
- 369 • To provide a method for basing an **authorization decision** on the contents of an information  
370 **resource**.
- 371 • To provide a set of logical and mathematical operators on **attributes** of the **subject**, **resource**  
372 and **environment**.

- 373 • To provide a method for handling a distributed set of **policy** components, while abstracting the  
374 method for locating, retrieving and authenticating the **policy** components.
- 375 • To provide a method for rapidly identifying the **policy** that applies to a given action, based upon  
376 the values of **attributes** of the **subjects, resource** and **action**.
- 377 • To provide an abstraction-layer that insulates the policy-writer from the details of the application  
378 environment.
- 379 • To provide a method for specifying a set of actions that must be performed in conjunction with  
380 policy enforcement.

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

## 384 2.2. Rule and policy combining

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

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

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

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

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

## 404 2.3. Combining algorithms

405 XACML defines a number of combining algorithms that can be identified by a  
406 RuleCombiningAlgId or PolicyCombiningAlgId attribute of the <Policy> or <PolicySet>  
407 elements, respectively. The **rule-combining algorithm** defines a procedure for arriving at an  
408 **authorization decision** given the individual results of evaluation of a set of **rules**. Similarly, the  
409 **policy-combining algorithm** defines a procedure for arriving at an **authorization decision** given  
410 the individual results of evaluation of a set of **policies**. Standard combining algorithms are defined  
411 for:

- 412 • Deny-overrides (Ordered and Unordered),
- 413 • Permit-overrides (Ordered and Unordered),

- 414 • First applicable and
- 415 • Only-one-applicable.

416 In the first case, if a single <Rule> or <Policy> element is encountered that evaluates to "Deny",  
417 then, regardless of the evaluation result of the other <Rule> or <Policy> elements in the  
418 **applicable policy**, the combined result is "Deny". Likewise, in the second case, if a single "Permit"  
419 result is encountered, then the combined result is "Permit". In the case of the "First-applicable"  
420 combining algorithm, the combined result is the same as the result of evaluating the first <Rule>,  
421 <Policy> or <PolicySet> element in the list of **rules** whose **target** is applicable to the **decision**  
422 **request**. The "Only-one-applicable" **policy-combining algorithm** only applies to **policies**. The  
423 result of this combining algorithm ensures that one and only one **policy** or **policy set** is applicable  
424 by virtue of their **targets**. If no **policy** or **policy set** applies, then the result is "NotApplicable", but if  
425 more than one **policy** or **policy set** is applicable, then the result is "Indeterminate". When exactly  
426 one **policy** or **policy set** is applicable, the result of the combining algorithm is the result of  
427 evaluating the single **applicable policy** or **policy set**.

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

## 429 2.4. Multiple subjects

430 Access-control policies often place requirements on the actions of more than one **subject**. For  
431 instance, the policy governing the execution of a high-value financial transaction may require the  
432 approval of more than one individual, acting in different capacities. Therefore, XACML recognizes  
433 that there may be more than one **subject** relevant to a **decision request**. An **attribute** called  
434 "subject-category" is used to differentiate between **subjects** acting in different capacities. Some  
435 standard values for this **attribute** are specified, and users may define additional ones.

## 436 2.5. Policies based on subject and resource attributes

437 Another common requirement is to base an **authorization decision** on some characteristic of the  
438 **subject** other than its identity. Perhaps, the most common application of this idea is the **subject's**  
439 role [RBAC]. XACML provides facilities to support this approach. **Attributes** of **subjects** may be  
440 identified by the <SubjectAttributeDesignator> element. This element contains a URN that  
441 identifies the **attribute**. Alternatively, the <AttributeSelector> element may contain an XPath  
442 expression over the request **context** to identify a particular **subject attribute** value by its location in  
443 the **context** (see Section 2.11 for an explanation of **context**). XACML provides a standard way to  
444 reference the **attributes** defined in the LDAP series of specifications [LDAP-1, LDAP-2]. This is  
445 intended to encourage implementers to use standard **attribute** identifiers for some common  
446 **subject attributes**.

447 Another common requirement is to base an **authorization decision** on some characteristic of the  
448 **resource** other than its identity. XACML provides facilities to support this approach. **Attributes** of  
449 **resource** may be identified by the <ResourceAttributeDesignator> element. This element  
450 contains a URN that identifies the **attribute**. Alternatively, the <AttributeSelector> element  
451 may contain an XPath expression over the request **context** to identify a particular **resource**  
452 **attribute** value by its location in the **context**.

## 453 2.6. Multi-valued attributes

454 The most common techniques for communicating **attributes** (LDAP, XPath, SAML, etc.) support  
455 multiple values per **attribute**. Therefore, when an XACML **PDP** retrieves the value of a named  
456 **attribute**, the result may contain multiple values. A collection of such values is called a **bag**. A  
457 **bag** differs from a set in that it may contain duplicate values, whereas a set may not. Sometimes

458 this situation represents an error. Sometimes the XACML *rule* is satisfied if any one of the  
459 *attribute* values meets the criteria expressed in the *rule*.

460 XACML provides a set of functions that allow a policy writer to be absolutely clear about how the  
461 *PDP* should handle the case of multiple *attribute* values. These are the “higher-order” functions.

## 462 2.7. Policies based on resource contents

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

468 XACML provides facilities for doing this when the information *resource* can be represented as an  
469 XML document. The `<AttributeSelector>` element may contain an XPath expression over the  
470 request *context* to identify data in the information *resource* to be used in the *policy* evaluation.

471 In cases where the information *resource* is not an XML document, specified *attributes* of the  
472 *resource* can be referenced, as described in Section 2.4.

## 473 2.8. Operators

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

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

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

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

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

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

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

## 503 2.9. Policy distribution

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

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

## 515 2.10. Policy indexing

516 For efficiency of evaluation and ease of management, the overall security policy in force across an  
517 enterprise may be expressed as multiple independent **policy** components. In this case, it is  
518 necessary to identify and retrieve the **applicable policy** statement and verify that it is the correct  
519 one for the requested action before evaluating it. This is the purpose of the <Target> element in  
520 XACML.

521 Two approaches are supported:

- 522 1. **Policy** statements may be stored in a database, whose data-model is congruent with that of the  
523 <Target> element. The **PDP** should use the contents of the **decision request** that it is  
524 processing to form the database read command by which applicable **policy** statements are  
525 retrieved. Nevertheless, the **PDP** should still evaluate the <Target> element of the retrieved  
526 **policy** or **policy set** statements as defined by the XACML specification.
- 527 2. Alternatively, the **PDP** may evaluate the <Target> element from each of the **policies** or  
528 **policy sets** that it has available to it, in the context of a particular **decision request**, in order to  
529 identify the **policies** and **policy sets** that are applicable to that request.

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

## 531 2.11. Abstraction layer

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

541 Naturally, XACML-conformant **PEPs** may issue requests and receive responses in the form of an  
542 XACML **context**. But, where this situation does not exist, an intermediate step is required to

543 convert between the request/response format understood by the *PEP* and the XACML *context*  
544 format understood by the *PDP*.

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

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

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

## 554 2.12. Actions performed in conjunction with enforcement

555 In many applications, policies specify actions that MUST be performed, either instead of, or in  
556 addition to, actions that MAY be performed. This idea was described by Sloman [Sloman94].  
557 XACML provides facilities to specify actions that MUST be performed in conjunction with policy  
558 evaluation in the <Obligations> element. This idea was described as a provisional action by  
559 Kudo [Kudo00]. There are no standard definitions for these actions in version 1.0 of XACML.  
560 Therefore, bilateral agreement between a *PAP* and the *PEP* that will enforce its *policies* is required  
561 for correct interpretation. *PEPs* that conform with v1.0 of XACML are required to deny *access*  
562 unless they understand all the <Obligations> elements associated with the *applicable policy*.  
563 <Obligations> elements are returned to the *PEP* for enforcement.

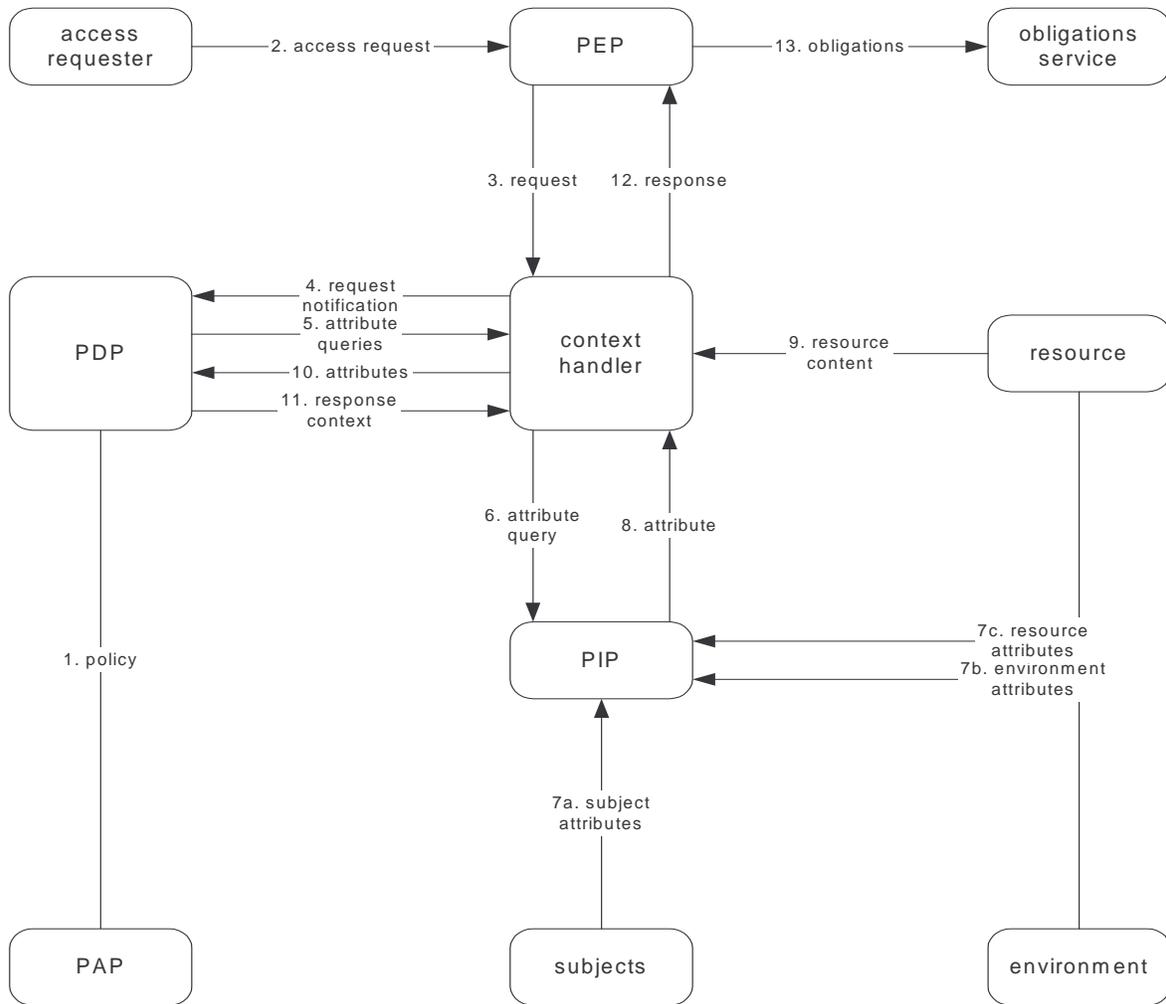
---

## 564 3. Models (non-normative)

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

### 566 3.1. Data-flow model

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



568

569

**Figure 1 - Data-flow diagram**

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

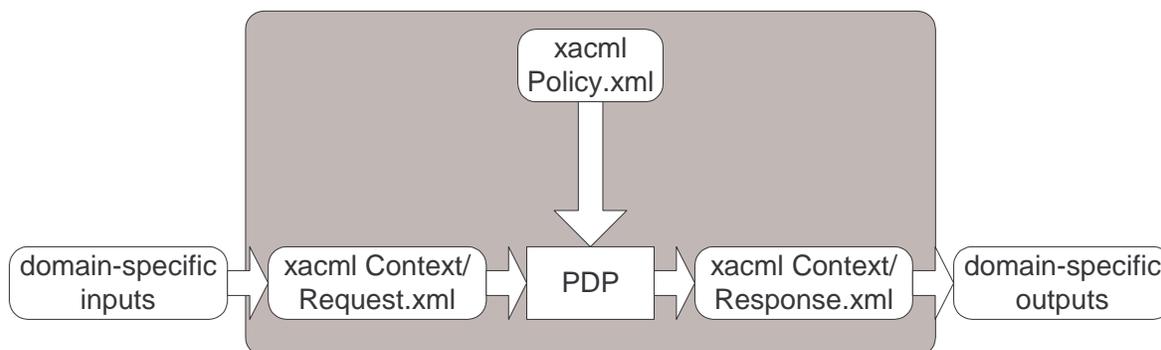
575 The model operates by the following steps.

- 576 1. **PAPs** write **policies** and **policy sets** and make them available to the **PDP**. These **policies** or  
 577 **policy sets** represent the complete policy for a specified **target**.
- 578 2. The access requester sends a request for access to the **PEP**.
- 579 3. The **PEP** sends the request for **access** to the **context handler** in its native request format,  
 580 optionally including **attributes** of the **subjects**, **resource** and **action**. The **context handler**  
 581 constructs an XACML request **context** in accordance with steps 4,5,6 and 7.
- 582 4. **Subject**, **resource** and **environment attributes** may be requested from a **PIP**.
- 583 5. The **PIP** obtains the requested **attributes**.
- 584 6. The **PIP** returns the requested **attributes** to the **context handler**.

- 585 7. Optionally, the **context handler** includes the **resource** in the **context**.
- 586 8. The **context handler** sends a **decision request**, including the **target**, to the **PDP**. The **PDP**  
 587 identifies the **applicable policy** and retrieves the required **attributes** and (optionally) the  
 588 **resource** from the **context handler**. The **PDP** evaluates the **policy**.
- 589 9. The **PDP** returns the response **context** (including the **authorization decision**) to the **context**  
 590 **handler**.
- 591 10. The **context handler** translates the response **context** to the native response format of the  
 592 **PEP**. The **context handler** returns the response to the **PEP**.
- 593 11. The **PEP** fulfills the **obligations**.
- 594 12. (Not shown) If **access** is permitted, then the **PEP** permits **access** to the **resource**; otherwise, it  
 595 denies **access**.

### 3.2. XACML context

597 XACML is intended to be suitable for a variety of application environments. The core language is  
 598 insulated from the application environment by the XACML **context**, as shown in Figure 2, in which  
 599 the scope of the XACML specification is indicated by the shaded area. The XACML **context** is  
 600 defined in XML schema, describing a canonical representation for the inputs and outputs of the  
 601 **PDP**. **Attributes** referenced by an instance of XACML policy may be in the form of XPath  
 602 expressions on the **context**, or attribute designators that identify the **attribute** by **subject**,  
 603 **resource**, **action** or **environment** and its identifier. Implementations must convert between the  
 604 **attribute** representations in the application environment (e.g., SAML, J2SE, CORBA, and so on)  
 605 and the **attribute** representations in the XACML **context**. How this is achieved is outside the  
 606 scope of the XACML specification. In some cases, such as SAML, this conversion may be  
 607 accomplished in an automated way through the use of an XSLT transformation.



608  
 609 **Figure 2 - XACML context**

610 Note: The **PDP** may be implemented such that it uses a processed form of the XML files.

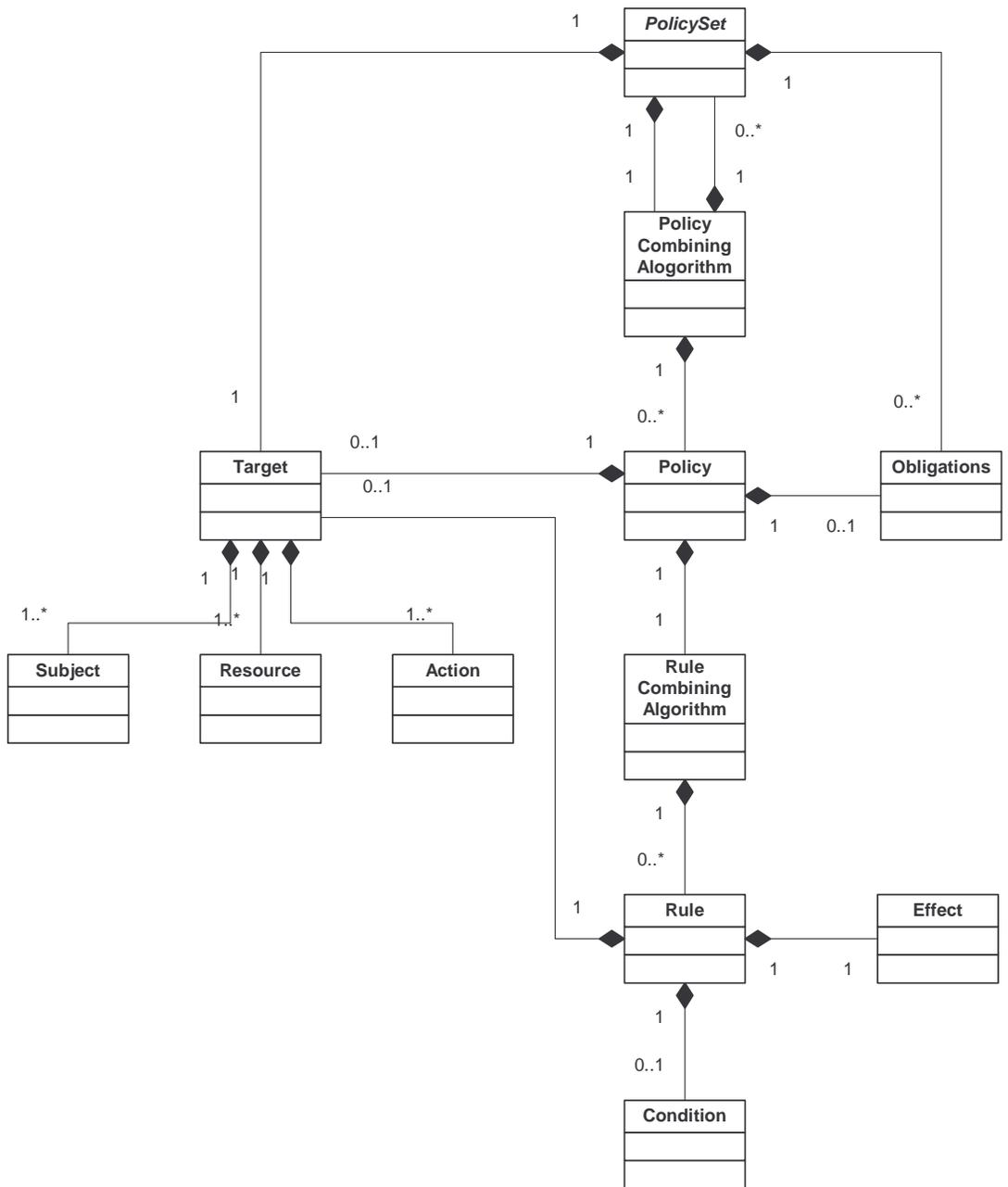
611 See Section 7.9 for a more detailed discussion of the request **context**.

### 3.3. Policy language model

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

- 614 • **Rule**;
- 615 • **Policy**; and

616 • **Policy set.**  
 617 These are described in the following sub-sections.



618

619

**Figure 3 - Policy language model**

620

### 3.3.1 Rule

621 A **rule** is the most elementary unit of **policy**. It may exist in isolation only *within* one of the major  
 622 actors of the XACML domain. In order to exchange **rules** between major actors, they must be  
 623 encapsulated in a **policy**. A **rule** can be evaluated on the basis of its contents. The main  
 624 components of a **rule** are:

- 625 • a **target**,
- 626 • an **effect**, and
- 627 • a **condition**.

628 These are discussed in the following sub-sections.

### 629 3.3.1.1. Rule target

630 The **target** defines the set of:

- 631 • **resources**;
- 632 • **subjects**; and
- 633 • **actions**

634 to which the **rule** is intended to apply. The `<Condition>` element may further refine the  
635 applicability established by the **target**. If the **rule** is intended to apply to all entities of a particular  
636 data-type, then an empty element named `<AnySubject/>`, `<AnyResource/>` or `<AnyAction/>`  
637 is used. An XACML **PDP** verifies that the **subjects**, **resource** and **action** identified in the request  
638 **context** are all present in the **target** of the **rules** that it uses to evaluate the **decision request**.  
639 **Target** definitions are discrete, in order that applicable **rules** may be efficiently identified by the  
640 **PDP**.

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

643 Certain **subject** name-forms, **resource** name-forms and certain types of **resource** are internally  
644 structured. For instance, the X.500 directory name-form and RFC 822 name-form are structured  
645 **subject** name-forms, whereas an account number commonly has no discernible structure. UNIX  
646 file-system path-names and URIs are examples of structured **resource** name-forms. And an XML  
647 document is an example of a structured **resource**.

648 Generally, the name of a node (other than a leaf node) in a structured name-form is also a legal  
649 instance of the name-form. So, for instance, the RFC822 name "medico.com" is a legal RFC822  
650 name identifying the set of mail addresses hosted by the medico.com mail server. And the  
651 XPath/XPointer value `//ctx:ResourceContent/md:record/md:patient/` is a legal  
652 XPath/XPointer value identifying a node-set in an XML document.

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

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

662 On the other hand, in the case of **resource** names and **resources** themselves, three options exist.  
663 The name could refer to:

- 664 1. the contents of the identified node only,
- 665 2. the contents of the identified node and the contents of its immediate child nodes or
- 666 3. the contents of the identified node and all its descendant nodes.

667 All three options are supported in XACML.

### 668 3.3.1.2. Effect

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

### 671 3.3.1.3. Condition

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

## 674 3.3.2 Policy

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

- 677 • a **target**,
- 678 • a **rule-combining algorithm**-identifier;
- 679 • a set of **rules**; and
- 680 • **obligations**.

681 **Rules** are described above. The remaining components are described in the following sub-  
682 sections.

### 683 3.3.2.1. Policy target

684 An XACML <PolicySet>, <Policy> or <Rule> element contains a <Target> element that  
685 specifies the set of **subjects**, **resources** and **actions** to which it applies. The <Target> of a  
686 <PolicySet> or <Policy> may be declared by the writer of the <PolicySet> or <Policy>, or  
687 it may be calculated from the <Target> elements of the <PolicySet>, <Policy> and <Rule>  
688 elements that it contains.

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

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

### 705 3.3.2.2. Rule-combining algorithm

706 The **rule-combining algorithm** specifies the procedure by which the results of evaluating the  
707 component **rules** are combined when evaluating the **policy**, i.e. the `Decision` value placed in the  
708 response **context** by the **PDP** is the value of the **policy**, as defined by the **rule-combining**  
709 **algorithm**.

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

### 711 3.3.2.3. Obligations

712 The XACML `<Rule>` syntax does not contain an element suitable for carrying **obligations**;  
713 therefore, if required in a **policy**, **obligations** must be added by the writer of the **policy**.

714 When a **PDP** evaluates a **policy** containing **obligations**, it returns certain of those **obligations** to  
715 the **PEP** in the response **context**. Section 7.11 explains which **obligations** are to be returned.

## 716 3.3.3 Policy set

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

- 718 • a **target**,
- 719 • a **policy-combining algorithm**-identifier
- 720 • a set of **policies**; and
- 721 • **obligations**.

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

### 724 3.3.3.1. Policy-combining algorithm

725 The **policy-combining algorithm** specifies the procedure by which the results of evaluating the  
726 component **policies** are combined when evaluating the **policy set**, i.e. the `Decision` value placed  
727 in the response **context** by the **PDP** is the result of evaluating the **policy set**, as defined by the  
728 **policy-combining algorithm**.

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

### 730 3.3.3.2. Obligations

731 The writer of a **policy set** may add **obligations** to the **policy set**, in addition to those contained in  
732 the component **policies** and **policy sets**.

733 When a **PDP** evaluates a **policy set** containing **obligations**, it returns certain of those **obligations**  
734 to the **PEP** in its response context. Section 7.11 explains which **obligations** are to be returned.

735

---

## 736 4. Examples (non-normative)

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

### 741 4.1. Example one

#### 742 4.1.1 Example policy

743 Assume that a corporation named Medi Corp (medico.com) has an **access control policy** that  
744 states, in English:

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

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

749 The header for this policy is

```
[p01] <?xml version=1.0" encoding="UTF-8"?>  
[p02] <Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"  
[p03] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
[p04] xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy  
[p05] http://www.oasis-open.org/tc/xacml/1.0/cs-xacml-schema-policy-01.xsd"  
[p06] PolicyId="identifier:example:SimplePolicy1"  
[p07] RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
```

750 [p01] is a standard XML document tag indicating which version of XML is being used and what the  
751 character encoding is.

752 [p02] introduces the XACML Policy itself.

753 [p03-p05] are XML namespace declarations.

754 [p05] gives a URL to the schema for XACML **policies**.

755 [p06] assigns a name to this **policy** instance. The name of a **policy** should be unique for a given  
756 **PDP** so that there is no ambiguity if one **policy** is referenced from another **policy**.

757 [p07] specifies the algorithm that will be used to resolve the results of the various **rules** that may be  
758 in the **policy**. The **deny-overrides rule-combining algorithm** specified here says that, if any **rule**  
759 evaluates to "Deny", then that **policy** must return "Deny". If all **rules** evaluate to "Permit", then the  
760 **policy** must return "Permit". The **rule-combining algorithm**, which is fully described in Appendix  
761 C, also says what to do if an error were to occur when evaluating any **rule**, and what to do with  
762 **rules** that do not apply to a particular **decision request**.

```
[p08] <Description>  
[p09] Medi Corp access control policy  
[p10] </Description>
```

763 [p08-p10] provide a text description of the policy. This description is optional.

```
[p11] <Target>  
[p12] <Subjects>  
[p13] <AnySubject/>  
[p14] </Subjects>  
[p15] <Resources>
```

```

[p16]      <AnyResource/>
[p17]      </Resources>
[p18]      <Actions>
[p19]      <AnyAction/>
[p20]      </Actions>
[p21]      </Target>

```

764 [p11-p21] describe the **decision requests** to which this **policy** applies. If the **subject, resource**  
765 and **action** in a **decision request** do not match the values specified in the **target**, then the  
766 remainder of the **policy** does not need to be evaluated. This **target** section is very useful for  
767 creating an index to a set of **policies**. In this simple example, the **target** section says the **policy** is  
768 applicable to any **decision request**.

```

[p22]      <Rule
[p23]      RuleId= "urn:oasis:names:tc:xacml:1.0:example:SimpleRule1"
[p24]      Effect="Permit">

```

769 [p22] introduces the one and only **rule** in this simple **policy**. Just as for a **policy**, each **rule** must  
770 have a unique identifier (at least unique for any **PDP** that will be using the **policy**).

771 [p23] specifies the identifier for this **rule**.

772 [p24] says what **effect** this **rule** has if the **rule** evaluates to “True”. **Rules** can have an **effect** of  
773 either “Permit” or “Deny”. In this case, the rule will evaluate to “Permit”, meaning that, as far as this  
774 one **rule** is concerned, the requested **access** should be permitted. If a **rule** evaluates to “False”,  
775 then it returns a result of “NotApplicable”. If an error occurs when evaluating the **rule**, the **rule**  
776 returns a result of “Indeterminate”. As mentioned above, the **rule-combining algorithm** for the  
777 **policy** tells how various **rule** values are combined into a single **policy** value.

```

[p25]      <Description>
[p26]      Any subject with an e-mail name in the medico.com domain
[p27]      can perform any action on any resource.
[p28]      </Description>

```

778 [p25-p28] provide a text description of this **rule**. This description is optional.

```

[p29]      <Target>

```

779 [p29] introduces the **target** of the **rule**. As described above for the **target** of a policy, the **target** of  
780 a **rule** describes the **decision requests** to which this **rule** applies. If the **subject, resource** and  
781 **action** in a **decision request** do not match the values specified in the **rule target**, then the  
782 remainder of the **rule** does not need to be evaluated, and a value of “NotApplicable” is returned to  
783 the **policy** evaluation.

```

[p30]      <Subjects>
[p31]      <Subject>
[p32]      <SubjectMatch MatchId="
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
[p33]      <SubjectAttributeDesignator
[p34]      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
[p35]      DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" />
[p36]      <AttributeValue
[p37]      DataType="urn:oasis:names:tc:xacml:1.0:data-
type:rfc822Name">medico.com
[p38]      </AttributeValue>
[p39]      </SubjectMatch>
[p40]      </Subject>
[p41]      </Subjects>
[p42]      <Resources>
[p43]      <AnyResource/>
[p44]      </Resources>
[p45]      <Actions>
[p46]      <AnyAction/>
[p47]      </Actions>
[p48]      </Target>

```

784 The **rule target** is similar to the **target** of the **policy** itself, but with one important difference. [p32-  
785 p41] do not say <AnySubject/>, but instead spell out a specific value that the **subject** in the  
786 **decision request** must match. The <SubjectMatch> element specifies a matching function in  
787 the MatchId attribute, a pointer to a specific **subject attribute** in the request **context** by means of  
788 the <SubjectAttributeDesignator> element, and a literal value of "medico.com". The  
789 matching function will be used to compare the value of the **subject attribute** with the literal value.  
790 Only if the match returns "True" will this **rule** apply to a particular **decision request**. If the match  
791 returns "False", then this **rule** will return a value of "NotApplicable".

```
[p49] </Rule>  
[p50] </ Policy>
```

792 [p49] closes the **rule** we have been examining. In this **rule**, all the **work** is done in the <Target>  
793 element. In more complex **rules**, the <Target> may have been followed by a <Condition>  
794 (which could also be a set of **conditions** to be **ANDed** or **ORed** together).

795 [p50] closes the **policy** we have been examining. As mentioned above, this **policy** has only one  
796 **rule**, but more complex **policies** may have any number of **rules**.

## 797 4.1.2 Example request context

798 Let's examine a hypothetical **decision request** that might be submitted to a **PDP** using the **policy**  
799 above. In English, the **access** request that generates the **decision request** may be stated as  
800 follows:

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

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

```
[c01] <?xml version="1.0" encoding="UTF-8"?>  
[c02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context "  
[c03] Xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "  
[c04] xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context  
[c05] http://www.oasis-open.org/tc/xacml/1.0/cs-xacml-schema-context-01.xsd">
```

805 [c01-c05] are the header for the **request context**, and are used the same way as the header for the  
806 **policy** explained above.

```
[c06] <Subject>  
[c07] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-  
id"  
[c08] DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">  
[c09] <AttributeValue>bs@simpsons.com</AttributeValue>  
[c10] </Attribute>  
[c11] </Subject>
```

807 The <Subject> element contains one or more **attributes** of the entity making the **access** request.  
808 There can be multiple **subjects**, and each **subject** can have multiple **attributes**. In this case, in  
809 [c06-c11], there is only one **subject**, and the **subject** has only one **attribute**: the **subject's** identity,  
810 expressed as an e-mail name, is "bs@simpsons.com".

```
[c12] <Resource>  
[c13] <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:ufs-  
path"  
[c14] DataType="http://www.w3.org/2001/XMLSchema#anyURI">  
[c15] <AttributeValue>/medico/record/patient/BartSimpson</AttributeValue>  
[c16] </Attribute>  
[c17] </Resource>
```

811 The <Resource> element contains one or more **attributes** of the **resource** to which  
812 the **subject** (or **subjects**) has requested **access**. There can be only one <Resource>

813 per **decision request**. Lines [c13-c16] contain the one **attribute** of the **resource**  
814 to which Bart Simpson has requested **access**: the **resource** unix file-system path-  
815 name, which is "/medico/record/patient/BartSimpson".

```
[c18] <Action>  
[c19]   <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"  
[c20]     DataType="http://www.w3.org/2001/XMLSchema#string">  
[c21]     <AttributeValue>read</AttributeValue>  
[c22]   </Attribute>  
[c23] </Action>
```

816 The <Action> element contains one or more **attributes** of the **action** that the **subject** (or  
817 **subjects**) wishes to take on the **resource**. There can be only one **action** per **decision request**.  
818 [c18-c23] describe the identity of the **action** Bart Simpson wishes to take, which is "read".

```
[c24] </Request>
```

819 [c24] closes the **request context**. A more complex **request context** may have contained some  
820 **attributes** not associated with the **subject**, the **resource** or the **action**. These would have been  
821 placed in an optional <Environment> element following the <Action> element.

822 The **PDP** processing this request **context** locates the **policy** in its policy repository. It compares  
823 the **subject**, **resource** and **action** in the request **context** with the **subjects**, **resources** and  
824 **actions** in the **policy target**. Since the **policy target** matches the <AnySubject/>,  
825 <AnyResource/> and <AnyAction/> elements, the **policy** matches this **context**.

826 The **PDP** now compares the **subject**, **resource** and **action** in the request **context** with the **target**  
827 of the one **rule** in this **policy**. The requested **resource** matches the <AnyResource/> element  
828 and the requested **action** matches the <AnyAction/> element, but the requesting subject-id  
829 **attribute** does not match "\*@medico.com".

### 830 4.1.3 Example response context

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

```
[r01] <?xml version="1.0" encoding="UTF-8"?>  
[r02] <Response xmlns="urn:oasis:names:tc:xacml:1.0:context "  
[r03]   xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context  
[r04]   http://www.oasis-open.org/tc/xacml/1.0/cs-xacml-schema-context-  
      01.xsd">
```

834 [r01-r04] contain the same sort of header information for the response as was described above for  
835 a **policy**.

```
[r05] <Result>  
[r06]   <Decision>NotApplicable</Decision>  
[r07] </Result>
```

836 The <Result> element in lines [r05-r07] contains the result of evaluating the **decision request**  
837 against the **policy**. In this case, the result is "NotApplicable". A **policy** can return "Permit", "Deny",  
838 "NotApplicable" or "Indeterminate".

```
[r08] </Response>
```

839 [r08] closes the response **context**.

## 840 4.2. Example two

841 This section contains an example XML document, an example request **context** and example  
842 XACML **rules**. The XML document is a medical record. Four separate **rules** are defined. These  
843 illustrate a **rule-combining algorithm**, **conditions** and **obligations**.

844

## 4.2.1 Example medical record instance

845 The following is an instance of a medical record to which the example XACML *rules* can be  
846 applied. The <record> schema is defined in the registered namespace administered by  
847 "http://medico.com".

```
848 <?xml version="1.0" encoding="UTF-8"?>
849 <record xmlns="http://www.medico.com/schemas/record.xsd "
850 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
851   <patient>
852     <patientName>
853       <first>Bartholomew</first>
854       <last>Simpson</last>
855     </patientName>
856     <patientContact>
857       <street>27 Shelbyville Road</street>
858       <city>Springfield</city>
859       <state>MA</state>
860       <zip>12345</zip>
861       <phone>555.123.4567</phone>
862       <fax/>
863       <email/>
864     </patientContact>
865     <patientDoB>1992-03-21</patientDoB>
866     <patientGender>male</patientGender>
867     <patient-number>555555</patient-number>
868   </patient>
869   <parentGuardian>
870     <parentGuardianId>HS001</parentGuardianId>
871     <parentGuardianName>
872       <first>Homer</first>
873       <last>Simpson</last>
874     </parentGuardianName>
875     <parentGuardianContact>
876       <street>27 Shelbyville Road</street>
877       <city>Springfield</city>
878       <state>MA</state>
879       <zip>12345</zip>
880       <phone>555.123.4567</phone>
881       <fax/>
882       <email>homers@aol.com</email>
883     </parentGuardianContact>
884   </parentGuardian>
885   <primaryCarePhysician>
886     <physicianName>
887       <first>Julius</first>
888       <last>Hibbert</last>
889     </physicianName>
890     <physicianContact>
891       <street>1 First St</street>
892       <city>Springfield</city>
893       <state>MA</state>
894       <zip>12345</zip>
895       <phone>555.123.9012</phone>
896       <fax>555.123.9013</fax>
897       <email/>
898     </physicianContact>
899     <registrationID>ABC123</registrationID>
900   </primaryCarePhysician>
901   <insurer>
902     <name>Blue Cross</name>
903     <street>1234 Main St</street>
904     <city>Springfield</city>
```

```

905     <state>MA</state>
906     <zip>12345</zip>
907     <phone>555.123.5678</phone>
908     <fax>555.123.5679</fax>
909     <email/>
910 </insurer>
911 <medical>
912     <treatment>
913         <drug>
914             <name>methylphenidate hydrochloride</name>
915             <dailyDosage>30mgs</dailyDosage>
916             <startDate>1999-01-12</startDate>
917         </drug>
918         <comment>patient exhibits side-effects of skin coloration and carpal
919 degeneration</comment>
920     </treatment>
921     <result>
922         <test>blood pressure</test>
923         <value>120/80</value>
924         <date>2001-06-09</date>
925         <performedBy>Nurse Betty</performedBy>
926     </result>
927 </medical>
928 </record>

```

## 929 4.2.2 Example request context

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

```

933 [01] <?xml version="1.0" encoding="UTF-8"?>
934 [02] <Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
935 [03] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
936 [04] <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
937 category:access-subject">
938 [05]     <Attribute AttributeId=
939 [06]         "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
940 [07]     DataType=
941 [08]         "urn:oasis:names:tc:xacml:1.0:data-type:x500name"
942 [09]     Issuer="www.medico.com"
943 [10]     IssueInstant="2001-12-17T09:30:47-05:00">
944 [11]         <AttributeValue>CN=Julius Hibbert</AttributeValue>
945 [12]     </Attribute>
946 [13]     <Attribute AttributeId=
947 [14]         "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
948 [15]     DataType="http://www.w3.org/2001/XMLSchema#string"
949 [16]     Issuer="www.medico.com"
950 [17]     IssueInstant="2001-12-17T09:30:47-05:00">
951 [18]         <AttributeValue>physician</AttributeValue>
952 [19]     </Attribute>
953 [20]     <Attribute AttributeId=
954 [21]         "urn:oasis:names:tc:xacml:1.0:example:attribute:physician-id"
955 [22]     DataType="http://www.w3.org/2001/XMLSchema#string"
956 [23]     Issuer="www.medico.com"
957 [24]     IssueInstant="2001-12-17T09:30:47-05:00">
958 [25]         <AttributeValue>jh1234</AttributeValue>
959 [26]     </Attribute>
960 [27] </Subject>
961 [28] <Resource>
962 [29]     <ResourceContent>
963 [30]         <md:record
964 [31]             xmlns:md="//http:www.medico.com/schemas/record.xsd">

```

```

965 [32] <md:patient>
966 [33] <md:patientDoB>1992-03-21</md:patientDoB>
967 [34] </md:patient>
968 [35] <!-- other fields -->
969 [36] </md:record>
970 [37] </ResourceContent>
971 [38] <Attribute AttributeId=
972 [39] "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
973 [40] DataType="http://www.w3.org/2001/XMLSchema#string">
974 [41] <AttributeValue>
975 [42] //medico.com/records/bart-simpson.xml#
976 [43] xmlns(md=/http:www.medico.com/schemas/record.xsd)
977 [44] xpointer(/md:record/md:patient/md:patientDoB)
978 [45] </AttributeValue>
979 [46] </Attribute>
980 [47] <Attribute AttributeId=
981 [48] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
982 [49] DataType="http://www.w3.org/2001/XMLSchema#string">
983 [50] <AttributeValue>
984 [51] xmlns(md=http:www.medico.com/schemas/record.xsd)
985 [52] xpointer(/md:record/md:patient/md:patientDoB)
986 [53] </AttributeValue>
987 [54] </Attribute>
988 [55] <Attribute AttributeId=
989 [56] "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
990 [57] DataType="http://www.w3.org/2001/XMLSchema#string">
991 [58] <AttributeValue>
992 [59] http://www.medico.com/schemas/record.xsd
993 [60] </AttributeValue>
994 [61] </Attribute>
995 [62] </Resource>
996 [63] <Action>
997 [64] <Attribute AttributeId=
998 [65] "urn:oasis:names:tc:xacml:1.0:action:action-id"
999 [66] DataType="http://www.w3.org/2001/XMLSchema#string">
1000 [67] <AttributeValue>read</AttributeValue>
1001 [68] </Attribute>
1002 [69] </Action>
1003 [70] </Request>

```

1004 [02]-[03] Standard namespace declarations.

1005 [04]-[27] **Subject** attributes are placed in the Subject section of the Request. Each **attribute**  
1006 consists of the **attribute** meta-data and the **attribute** value.

1007 [04] Each Subject element has SubjectCategory xml attribute. The value of this attribute  
1008 describes the role that the **subject** plays in making the **decision request**. The value of "access-  
1009 subject" denotes the identity for which the request was issued.

1010 [05]-[12] **Subject** subject-id **attribute**.

1011 [13]-[19] **Subject** role **attribute**.

1012 [20]-[26] **Subject** physician-id **attribute**.

1013 [28]-[62] **Resource** attributes are placed in the Resource section of the Request. Each **attribute**  
1014 consists of **attribute** meta-data and an **attribute** value.

1015 [29]-[36] **Resource** content. The XML document that is being requested is placed here.

1016 [38]-[46] **Resource** identifier.

1017 [47]-[61] The **Resource** is identified with an Xpointer expression that names the URI of the file that  
1018 is accessed, the target namespace of the document, and the XPath location path to the specific  
1019 element.

1020 [47]-[54] The XPath location path in the "resource-id" attribute is extracted and placed in the  
1021 xpath attribute.

1022 [55]-[61] **Resource** target-namespace **attribute**.

1023 [63]-[69] **Action attributes** are placed in the Action section of the Request.

1024 [64]-[68] **Action** identifier.

## 1025 **4.2.3 Example plain-language rules**

1026 The following plain-language rules are to be enforced:

1027 Rule 1: A person, identified by his or her patient number, may read any record for which he  
1028 or she is the designated patient.

1029 Rule 2: A person may read any record for which he or she is the designated parent or  
1030 guardian, and for which the patient is under 16 years of age.

1031 Rule 3: A physician may write to any medical element for which he or she is the designated  
1032 primary care physician, provided an email is sent to the patient.

1033 Rule 4: An administrator shall not be permitted to read or write to medical elements of a  
1034 patient record.

1035 These **rules** may be written by different **PAPs** operating independently, or by a single **PAP**.

## 1036 **4.2.4 Example XACML rule instances**

### 1037 **4.2.4.1. Rule 1**

1038 Rule 1 illustrates a simple **rule** with a single <Condition> element. The following XACML  
1039 <Rule> instance expresses Rule 1:

```
1040 [01] <?xml version="1.0" encoding="UTF-8"?>  
1041 [02] <Rule  
1042 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"  
1043 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1044 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context "  
1045 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"  
1046 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"  
1047 [08]   Effect="Permit">  
1048 [09] <Description>  
1049 [10]   A person may read any medical record in the  
1050 [11]   http://www.medico.com/schemas/record.xsd namespace  
1051 [12]   for which he or she is a designated patient  
1052 [13] </Description>  
1053 [14] <Target>  
1054 [15]   <Subjects>  
1055 [16]     <AnySubject/>  
1056 [17]   </Subjects>  
1057 [18]   <Resources>  
1058 [20]     <Resource>  
1059 [21]       <!-- match document target namespace -->
```

```

1060 [22] <ResourceMatch
1061 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1062 [23] <AttributeValue
1063 DataType="http://www.w3.org/2001/XMLSchema#string">
1064 [24] http://www.medico.com/schemas/record.xsd
1065 [25] </AttributeValue>
1066 [26] <ResourceAttributeDesignator AttributeId=
1067 [27] "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1068 [28] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1069 </ResourceMatch>
1070 [29] <!-- match requested xml element -->
1071 [30] <ResourceMatch
1072 MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1073 [31] <AttributeValue
1074 DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeV
1075 alue>
1076 [32] <ResourceAttributeDesignator AttributeId=
1077 [33] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1078 [34] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1079 </ResourceMatch>
1080 [35] </Resource>
1081 [36] </Resources>
1082 [37] <Actions>
1083 [38] <Action>
1084 [39] <ActionMatch
1085 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1086 [40] <AttributeValue
1087 DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1088 [41] <ActionAttributeDesignator AttributeId=
1089 [42] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1090 [43] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1091 </ActionMatch>
1092 [44] </Action>
1093 [45] </Actions>
1094 [46] </Target>
1095 [47] <!-- compare policy number in the document with
1096 [48] policy-number attribute -->
1097 [49] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1098 equal">
1099 [50] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1100 and-only">
1101 [51] <!-- policy-number attribute -->
1102 [52] <SubjectAttributeDesignator AttributeId=
1103 [53] "urn:oasis:names:tc:xacml:1.0:examples:attribute:policy-number"
1104 [54] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1105 </Apply>
1106 [55] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1107 and-only">
1108 [56] <!-- policy number in the document -->
1109 [57] <AttributeSelector RequestContextPath=
1110 [58] "//md:record/md:patient/md:patient-number/text()"
1111 [59] DataType="http://www.w3.org/2001/XMLSchema#string">
1112 </AttributeSelector>
1113 [60] </Apply>
1114 [61] </Condition>
1115 [62] </Rule>

```

1116 [02]-[06]. XML namespace declarations.

1117 [07] **Rule** identifier.

1118 [08]. When a **rule** evaluates to 'True' it emits the value of the **Effect** attribute. This value is  
1119 combined with the **Effect** values of other rules according to the **rule-combining algorithm**.

1120 [09]-[13] Free form description of the *rule*.

1121 [14]-[46]. A *rule target* defines a set of *decision requests* that are applicable to the *rule*. A  
1122 *decision request*, such that the value of the  
1123 “urn:oasis:names:tc:xacml:1.0:resource:target-namespace” *resource attribute* is  
1124 equal to “http://www.medico.com/schema/records.xsd” and the value of the  
1125 “urn:oasis:names:tc:xacml:1.0:resource:xpath” *resource attribute* matches the XPath  
1126 expression “/md:record” and the value of the  
1127 “urn:oasis:names:tc:xacml:1.0:action:action-id” *action attribute* is equal to “read”,  
1128 matches the *target* of this *rule*.

1129 [15]-[17]. The *Subjects* element may contain either a *disjunctive sequence* of *Subject*  
1130 elements or *AnySubject* element.

1131 [16] The *AnySubject* element is a special element that matches any *subject* in the request  
1132 *context*.

1133 [18]-[36]. The *Resources* element may contain either a *disjunctive sequence* of *Resource*  
1134 elements or *AnyResource* element.

1135 [20]-[35] The *Resource* element encloses the *conjunctive sequence* of *ResourceMatch*  
1136 elements.

1137 [22]-[28] The *ResourceMatch* element compares its first and second child elements according to  
1138 the matching function. A match is positive if the value of the first argument matches any of the  
1139 values selected by the second argument. This match compares the target namespace of the  
1140 requested document with the value of “http://www.medico.com/schema.records.xsd”.

1141 [22] The *MatchId* attribute names the matching function.

1142 [23]-[25] Literal attribute value to match.

1143 [26]-[27] The *ResourceAttributeDesignator* element selects the *resource attribute* values  
1144 from the request *context*. The *attribute* name is specified by the *AttributeId*. The selection  
1145 result is a *bag* of values.

1146 [30]-[34] The *ResourceMatch*. This match compares the results of two XPath expressions. The  
1147 first XPath expression is /md:record and the second XPath expression is the location path to the  
1148 requested xml element. The “xpath-node-match” function evaluates to “True” if the requested XML  
1149 element is below the /md:record element.

1150 [30] *MatchId* attribute names the matching function.

1151 [31] The literal XPath expression to match. The md prefix is resolved using a standard namespace  
1152 declaration.

1153 [32]-[33] The *ResourceAttributeDesignator* selects the *bag* of values for the  
1154 “urn:oasis:names:tc:xacml:1.0:xpath” *resource attribute*. Here, there is just one  
1155 element in the *bag*, which is the location path for the requested XML element.

1156 [37]-[45] The *Actions* element may contain either a *disjunctive sequence* of *Action* elements  
1157 or an *AnyAction* element.

1158 [38]-[44] The *Action* element contains a *conjunctive sequence* of *ActionMatch* elements.

1159 [39]-[43] The *ActionMatch* element compares its first and second child elements according to the  
1160 matching function. Match is positive if the value of the first argument matches any of the values  
1161 selected by the second argument. In this case, the value of the *action-id* action attribute in the  
1162 request *context* is compared with the value “read”.

1163 [39] The `MatchId` attribute names the matching function.

1164 [40] The **Attribute** value to match. This is an **action** name.

1165 [41]-[42] The `ActionAttributeDesignator` selects **action attribute** values from the request  
 1166 **context**. The **attribute** name is specified by the `AttributeId`. The selection result is a **bag** of  
 1167 values. “urn:oasis:names:tc:xacml:1.0:action:action-id” is the predefined name for  
 1168 the action identifier.

1169 [49]-[61] The `<Condition>` element. A **condition** must evaluate to “True” for the **rule** to be  
 1170 applicable. This condition evaluates the truth of the statement: the `patient-number` **subject**  
 1171 **attribute** is equal to the `patient-number` in the XML document.

1172 [49] The `FunctionId` attribute of the `<Condition>` element names the function to be used for  
 1173 comparison. In this case, comparison is done with  
 1174 `urn:oasis:names:tc:xacml:1.0:function:string-equal`; this function takes two  
 1175 arguments of the “`http://www.w3.org/2001/XMLSchema#string`” data-type.

1176 [50] The first argument to the `urn:oasis:names:tc:xacml:1.0:function:string-equal`  
 1177 in the `Condition`. Functions can take other functions as arguments. The `Apply` element  
 1178 encodes the function call with the `FunctionId` attribute naming the function. Since  
 1179 `urn:oasis:names:tc:xacml:1.0:function:string-equal` takes arguments of the  
 1180 “`http://www.w3.org/2001/XMLSchema#string`” data-type and  
 1181 `SubjectAttributeDesignator` selects a **bag** of  
 1182 “`http://www.w3.org/2001/XMLSchema#string`” values,  
 1183 “`urn:oasis:names:tc:xacml:1.0:function:string-one-and-only`” is used. This  
 1184 function guarantees that its argument evaluates to a **bag** containing one and only one  
 1185 “`http://www.w3.org/2001/XMLSchema#string`” element.

1186 [52]-[53] The `SubjectAttributeDesignator` selects a **bag** of values for the `policy-number`  
 1187 **subject attribute** in the request **context**.

1188 [55] The second argument to the “`urn:oasis:names:tc:xacml:1.0:function:string-`  
 1189 `equal`” in the `Condition`. Functions can take other functions as arguments. The `Apply` element  
 1190 encodes function call with the `FunctionId` attribute naming the function. Since  
 1191 “`urn:oasis:names:tc:xacml:1.0:function:string-equal`” takes arguments of the  
 1192 “`http://www.w3.org/2001/XMLSchema#string`” data-type and the `AttributeSelector`  
 1193 selects a **bag** of “`http://www.w3.org/2001/XMLSchema#string`” values,  
 1194 “`urn:oasis:names:tc:xacml:1.0:function:string-one-and-only`” is used. This  
 1195 function guarantees that its argument evaluates to a **bag** containing one and only one  
 1196 “`http://www.w3.org/2001/XMLSchema#string`” element.

1197 [57] The `AttributeSelector` element selects a **bag** of values from the request **context**. The  
 1198 `AttributeSelector` is a free-form XPath pointing device into the request **context**. The  
 1199 `RequestContextPath` attribute specifies an XPath expression over the content of the requested  
 1200 XML document, selecting the policy number. Note that the namespace prefixes in the XPath  
 1201 expression are resolved with the standard XML namespace declarations.

#### 1202 4.2.4.2. Rule 2

1203 Rule 2 illustrates the use of a mathematical function, i.e. the `<Apply>` element with `functionId`  
 1204 “urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration” to calculate date. It also  
 1205 illustrates the use of **predicate** expressions, with the `functionId`  
 1206 “urn:oasis:names:tc:xacml:1.0:function:and”.

1207 [01] `<?xml version="1.0" encoding="UTF-8" ?>`

```

1208 [02] <Rule
1209 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1210 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1211 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1212 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1213 [07]   RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1214 [08]   Effect="Permit">
1215 [09]   <Description>
1216 [10]     A person may read any medical record in the
1217 [11]     http://www.medico.com/records.xsd namespace
1218 [12]     for which he or she is the designated parent or guardian,
1219 [13]     and for which the patient is under 16 years of age
1220 [14]   </Description>
1221 [15]   <Target>
1222 [16]     <Subjects>
1223 [17]       <AnySubject/>
1224 [18]     </Subjects>
1225 [19]     <Resources>
1226 [20]       <Resource>
1227 [21]         <!-- match document target namespace -->
1228 [22]         <ResourceMatch
1229 [23]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1230 [24]             <AttributeValue
1231 [25]               DataType="http://www.w3.org/2001/XMLSchema#string">
1232 [26]                 http://www.medico.com/schemas/record.xsd
1233 [27]               </AttributeValue>
1234 [28]               <ResourceAttributeDesignator AttributeId=
1235 [29]                 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1236 [30]               DataType="http://www.w3.org/2001/XMLSchema#string"/>
1237 [31]             </ResourceMatch>
1238 [32]             <!-- match requested xml element -->
1239 [33]             <ResourceMatch
1240 [34]               MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1241 [35]                 <AttributeValue
1242 [36]                   DataType="http://www.w3.org/2001/XMLSchema#string">/md:record</AttributeV
1243 [37]                   alue>
1244 [38]                 <ResourceAttributeDesignator AttributeId=
1245 [39]                   "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1246 [40]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1247 [41]               </ResourceMatch>
1248 [42]             </Resource>
1249 [43]           </Resources>
1250 [44]         <Actions>
1251 [45]           <Action>
1252 [46]             <!-- match 'read' action -->
1253 [47]             <ActionMatch
1254 [48]               MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1255 [49]                 <AttributeValue
1256 [50]                   DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
1257 [51]                 <ActionAttributeDesignator AttributeId=
1258 [52]                   "urn:oasis:names:tc:xacml:1.0:action:action-id"
1259 [53]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1260 [54]               </ActionMatch>
1261 [55]             </Action>
1262 [56]           </Actions>
1263 [57]         </Target>
1264 [58]       <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
1265 [59]         <!-- compare parent-guardian-id subject attribute with
1266 [60]         the value in the document -->
1267 [61]         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1268 [62]         equal">
1269 [63]           <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1270 [64]           and-only">

```

```

1271 [53] <!-- parent-guardian-id subject attribute -->
1272 [54] <SubjectAttributeDesignator AttributeId=
1273 [55] "urn:oasis:names:tc:xacml:1.0:examples:attribute:
1274 [56] parent-guardian-id"
1275 [56] DataType="http://www.w3.org/2001/XMLSchema#string"/>
1276 [57] </Apply>
1277 [58] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1278 [58] and-only">
1279 [59] <!-- parent-guardian-id element in the document -->
1280 [60] <AttributeSelector RequestContextPath=
1281 [61] "//md:record/md:parentGuardian/md:parentGuardianId/text()"
1282 [62] DataType="http://www.w3.org/2001/XMLSchema#string">
1283 [63] </AttributeSelector>
1284 [64] </Apply>
1285 [65] </Apply>
1286 [66] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-
1287 [66] equal">
1288 [67] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-
1289 [67] and-only">
1290 [68] <EnvironmentAttributeDesignator AttributeId=
1291 [69] "urn:oasis:names:tc:xacml:1.0:environment:current-date"
1292 [69] DataType="http://www.w3.org/2001/XMLSchema#date"/>
1293 [70] </Apply>
1294 [71] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-
1295 [71] yearMonthDuration">
1296 [73] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-
1297 [73] one-and-only">
1298 [74] <!-- patient dob recorded in the document -->
1299 [75] <AttributeSelector RequestContextPath=
1300 [76] "//md:record/md:patient/md:patientDoB/text()"
1301 [76] DataType="http://www.w3.org/2001/XMLSchema#date">
1302 [77] </AttributeSelector>
1303 [78] </Apply>
1304 [79] <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-
1305 [79] operators-20020816#yearMonthDuration">
1306 [80] P16Y
1307 [81] </AttributeValue>
1308 [82] </Apply>
1309 [83] </Apply>
1310 [84] </Condition>
1311 [85] </Rule>

```

1312 [02]-[47] **Rule** declaration and **rule target**. See Rule 1 in Section 4.2.4.1 for the detailed  
1313 explanation of these elements.

1314 [48]-[82] The Condition element. **Condition** must evaluate to "True" for the **rule** to be applicable.  
1315 This **condition** evaluates the truth of the statement: the requestor is the designated parent or  
1316 guardian and the patient is under 16 years of age.

1317 [48] The Condition is using the "urn:oasis:names:tc:xacml:1.0:function:and"  
1318 function. This is a boolean function that takes one or more boolean arguments (2 in this case) and  
1319 performs the logical "AND" operation to compute the truth value of the expression.

1320 [51]-[65] The truth of the first part of the condition is evaluated: The requestor is the designated  
1321 parent or guardian. The Apply element contains a function invocation. The function name is  
1322 contained in the FunctionId attribute. The comparison is done with  
1323 "urn:oasis:names:tc:xacml:1.0:function:string-equal" that takes 2 arguments of  
1324 "http://www.w3.org/2001/XMLSchema#string" data-type.

1325 [52] Since "urn:oasis:names:tc:xacml:1.0:function:string-equal" takes arguments  
1326 of the "http://www.w3.org/2001/XMLSchema#string" data-type,  
1327 "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" is used to ensure

1328 that the **subject attribute** "urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id" in  
 1329 the request **context** contains one and only one value.  
 1330 "urn:oasis:names:tc:xacml:1.0:function:string-equal" takes an argument  
 1331 expression that evaluates to a **bag** of "http://www.w3.org/2001/XMLSchema#string"  
 1332 values.

1333 [54] Value of the **subject attribute**  
 1334 "urn:oasis:names:tc:xacml:1.0:examples:attribute:parent-guardian-id" is  
 1335 selected from the request **context** with the <SubjectAttributeDesignator> element. This  
 1336 expression evaluates to a bag of "http://www.w3.org/2001/XMLSchema#string" values.

1337 [58] "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only" is used to  
 1338 ensure that the **bag** of values selected by it's argument contains one and only one value of data-  
 1339 type "http://www.w3.org/2001/XMLSchema#string".

1340 [60] The value of the md:parentGuardianId element is selected from the **resource** content with  
 1341 the AttributeSelector element. AttributeSelector is a free-form XPath expression,  
 1342 pointing into the request **context**. The RequestContextPath XML attribute contains an XPath  
 1343 expression over the request **context**. Note that all namespace prefixes in the XPath expression  
 1344 are resolved with standard namespace declarations. The AttributeSelector evaluates to the  
 1345 **bag** of values of data-type "http://www.w3.org/2001/XMLSchema#string".

1346 [66]-[83] The expression: "the patient is under 16 years of age" is evaluated. The patient is under  
 1347 16 years of age if the current date is less than the date computed by adding 16 to the patient's date  
 1348 of birth.

1349 [66] "urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal" is used to  
 1350 compute the difference of two dates.

1351 [67] "urn:oasis:names:tc:xacml:1.0:function:date-one-and-only" is used to ensure  
 1352 that the **bag** of values selected by its argument contains one and only one value of data-type  
 1353 "http://www.w3.org/2001/XMLSchema#date".

1354 [68]-[69] Current date is evaluated by selecting the  
 1355 "urn:oasis:names:tc:xacml:1.0:environment:current-date" **environment attribute**.

1356 [71] "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" is  
 1357 used to compute the date by adding 16 to the patient's date of birth. The first argument is a  
 1358 "http://www.w3.org/2001/XMLSchema#date", and the second argument is an  
 1359 "http://www.w3.org/TR/2002/WD-xquery-operators-  
 1360 20020816#yearMonthDuration".

1361 [73] "urn:oasis:names:tc:xacml:1.0:function:date-one-and-only" is used to ensure  
 1362 that the **bag** of values selected by it's argument contains one and only one value of data-type  
 1363 "http://www.w3.org/2001/XMLSchema#date".

1364 [75]-[76] The <AttributeSelector> element selects the patient's date of birth by taking the  
 1365 XPath expression over the document content.

1366 [79]-[81] Year Month Duration of 16 years.

### 4.2.4.3. Rule 3

1367  
 1368 Rule 3 illustrates the use of an **obligation**. The XACML <Rule> element syntax does not include  
 1369 an element suitable for carrying an **obligation**, therefore Rule 3 has to be formatted as a  
 1370 <Policy> element.

1371 [01] <?xml version="1.0" encoding="UTF-8"?>

```

1372 [02] <Policy
1373 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1374 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1375 [05]   xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1376 [06]   xmlns:md="http://www.medico.com/schemas/record.xsd"
1377 [07]   PolicyId="urn:oasis:names:tc:xacml:examples:policyid:3"
1378 [08]   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1379 [09]     rule-combining-algorithm:deny-overrides">
1380 [10] <Description>
1381 [11]   Policy for any medical record in the
1382 [12]   http://www.medico.com/schemas/record.xsd namespace
1383 [13] </Description>
1384 [14] <Target>
1385 [15]   <Subjects>
1386 [16]     <AnySubject/>
1387 [17]   </Subjects>
1388 [18]   <Resources>
1389 [19]     <Resource>
1390 [20]       <!-- match document target namespace -->
1391 [21]       <ResourceMatch
1392 [22]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1393 [23]           <AttributeValue
1394 [24]             DataType="http://www.w3.org/2001/XMLSchema#string">
1395 [25]               http://www.medico.com/schemas/record.xsd
1396 [26]             </AttributeValue>
1397 [27]             <ResourceAttributeDesignator AttributeId=
1398 [28]               "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1399 [29]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1400 [30]           </ResourceMatch>
1401 [31]         </Resource>
1402 [32]       </Resources>
1403 [33]     <Actions>
1404 [34]       <AnyAction/>
1405 [35]     </Actions>
1406 [36]   </Target>
1407 [37] <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:3"
1408 [38]   Effect="Permit">
1409 [39]   <Description>
1410 [40]     A physician may write any medical element in a record
1411 [41]     for which he or she is the designated primary care
1412 [42]     physician, provided an email is sent to the patient
1413 [43]   </Description>
1414 [44]   <Target>
1415 [45]     <Subjects>
1416 [46]       <Subject>
1417 [47]         <!-- match subject group attribute -->
1418 [48]         <SubjectMatch
1419 [49]           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1420 [50]             <AttributeValue
1421 [51]               DataType="http://www.w3.org/2001/XMLSchema#string">physician</AttributeVa
1422 [52]             lue>
1423 [53]             <SubjectAttributeDesignator AttributeId=
1424 [54]               "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1425 [55]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1426 [56]           </SubjectMatch>
1427 [57]         </Subject>
1428 [58]       </Subjects>
1429 [59]     <Resources>
1430 [60]       <Resource>
1431 [61]         <!-- match requested xml element -->
1432 [62]         <ResourceMatch
1433 [63]           MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">

```

```

1434 [ 56]         <AttributeValue
1435         DataType="http://www.w3.org/2001/XMLSchema#string">
1436 [ 57]             /md:record/md:medical
1437 [ 58]         </AttributeValue>
1438 [ 59]         <ResourceAttributeDesignator AttributeId=
1439 [ 60]             "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1440         DataType="http://www.w3.org/2001/XMLSchema#string"/>
1441 [ 61]         </ResourceMatch>
1442 [ 62]     </Resource>
1443 [ 63] </Resources>
1444 [ 64] <Actions>
1445 [ 65]     <Action>
1446 [ 66]         <!-- match action -->
1447 [ 67]         <ActionMatch
1448             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1449 [ 68]             <AttributeValue
1450                 DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
1451 [ 069]             <ActionAttributeDesignator AttributeId=
1452 [ 070]                 "urn:oasis:names:tc:xacml:1.0:action:action-id"
1453                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1454 [ 071]             </ActionMatch>
1455 [ 072]         </Action>
1456 [ 073]     </Actions>
1457 [ 074] </Target>
1458 [ 075] <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
1459     equal">
1460 [ 076]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1461     and-only">
1462 [ 077]         <!-- physician-id subject attribute -->
1463 [ 078]         <SubjectAttributeDesignator AttributeId=
1464 [ 079]             "urn:oasis:names:tc:xacml:1.0:example:
1465 [ 080]             attribute:physician-id"
1466                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1467 [ 081]         </Apply>
1468 [ 082]     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-
1469     and-only">
1470 [ 083]         <AttributeSelector RequestContextPath=
1471 [ 084]             "//md:record/md:primaryCarePhysician/md:registrationID/text()"
1472 [ 085]             DataType="http://www.w3.org/2001/XMLSchema#string"/>
1473 [ 086]         </Apply>
1474 [ 087]     </Condition>
1475 [ 089] </Rule>
1476 [ 090] <Obligations>
1477 [ 091]     <!-- send e-mail message to the document owner -->
1478 [ 092]     <Obligation ObligationId=
1479 [ 093]         "urn:oasis:names:tc:xacml:example:obligation:email"
1480 [ 094]         FulfillOn="Permit">
1481 [ 095]         <AttributeAssignment AttributeId=
1482 [ 096]             "urn:oasis:names:tc:xacml:1.0:example:attribute:mailto"
1483 [ 097]             DataType="http://www.w3.org/2001/XMLSchema#string">
1484 [ 098]             <AttributeSelector RequestContextPath=
1485 [ 099]                 "//md:/record/md:patient/md:patientContact/md:email"
1486 [ 100]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1487 [ 101]             </AttributeAssignment>
1488 [ 102]         <AttributeAssignment AttributeId=
1489 [ 103]             "urn:oasis:names:tc:xacml:1.0:example:attribute:text"
1490 [ 104]             DataType="http://www.w3.org/2001/XMLSchema#string">
1491 [ 105]             <AttributeValue>
1492 [ 106]                 Your medical record has been accessed by:
1493 [ 107]             </AttributeValue>
1494 [ 108]         </AttributeAssignment>
1495 [ 109]         <AttributeAssignment AttributeId=
1496 [ 110]             "urn:oasis:names:tc:xacml:example:attribute:text"

```

```

1497 [111]      DataType="http://www.w3.org/2001/XMLSchema#string">
1498 [112]      <SubjectAttributeDesignator AttributeId=
1499 [113]      "urn:osasis:names:tc:xacml:1.0:subject:subject-id"
1500      DataType="http://www.w3.org/2001/XMLSchema#string"/>
1501 [114]      </AttributeAssignment>
1502 [115]      </Obligation>
1503 [116] </Obligations>
1504 [117] </Policy>

```

1505 [01]-[09] The `Policy` element includes standard namespace declarations as well as policy specific  
1506 parameters, such as `PolicyId` and `RuleCombiningAlgId`.

1507 [07] **Policy** identifier. This parameter is used for the inclusion of the `Policy` in the `PolicySet`  
1508 element.

1509 [08]-[09] **Rule combining algorithm** identifier. This parameter is used to compute the combined  
1510 outcome of **rule effects** for **rules** that are applicable to the **decision request**.

1511 [10-13] Free-form description of the **policy**.

1512 [14]-[33] **Policy target**. The **policy target** defines a set of applicable decision requests. The  
1513 structure of the `Target` element in the `Policy` is identical to the structure of the `Target` element  
1514 in the `Rule`. In this case, the **policy target** is a set of all XML documents conforming to the  
1515 "http://www.medico.com/schemas/record.xsd" target namespace. For the detailed description of  
1516 the `Target` element see Rule 1, Section 4.2.4.1.

1517 [34]-[89] The only `Rule` element included in this `Policy`. Two parameters are specified in the **rule**  
1518 header: `RuleId` and `Effect`. For the detailed description of the `Rule` structure see Rule 1,  
1519 Section 4.2.4.1.

1520 [41]-[74] A **rule target** narrows down a **policy target**. **Decision requests** with the value of  
1521 "urn:osasis:names:tc:xacml:1.0:example:attribute:role" **subject attribute** equal to  
1522 "physician" [42]-[51], and that access elements of the medical record that "xpath-node-match"  
1523 the "/md:record/md:medical" XPath expression [52]-[63], and that have the value of the  
1524 "urn:osasis:names:tc:xacml:1.0:action:action-id" **action attribute** equal to "read".

1525 [65]-[73] match the **target** of this **rule**. For a detailed description of the rule target see example 1,  
1526 Section 4.2.4.1.

1527 [75]-[87] The `Condition` element. For the **rule** to be applicable to the authorization request,  
1528 **condition** must evaluate to True. This **rule condition** compares the value of the  
1529 "urn:osasis:names:tc:xacml:1.0:examples:attribute:physician-id" **subject**  
1530 **attribute** with the value of the `physician id` element in the medical record that is being  
1531 accessed. For a detailed explanation of rule condition see Rule 1, Section 4.2.4.1.

1532 [90]-[116] The `Obligations` element. **Obligations** are a set of operations that must be  
1533 performed by the **PEP** in conjunction with an **authorization decision**. An **obligation** may be  
1534 associated with a positive or negative **authorization decision**.

1535 [92]-[115] The `Obligation` element consists of the `ObligationId`, the authorization decision  
1536 value for which it must fulfill, and a set of attribute assignments.

1537 [92]-[93] `ObligationId` identifies an **obligation**. **Obligation** names are not interpreted by the  
1538 **PDP**.

1539 [94] `FulfillOn` attribute defines an **authorization decision** value for which this **obligation** must  
1540 be fulfilled.

1541 [95]-[101] **Obligation** may have one or more parameters. The **obligation** parameter  
 1542 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" is assigned the value  
 1543 from the content of the xml document.

1544 [95-96] AttributeId declares  
 1545 "urn:oasis:names:tc:xacml:1.0:examples:attribute:mailto" **obligation** parameter.

1546 [97] The **obligation** parameter data-type is defined.

1547 [98]-[100] The **obligation** parameter value is selected from the content of the XML document that is  
 1548 being accessed with the XPath expression over request **context**.

1549 [102]-[108] The **obligation** parameter  
 1550 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of data-type  
 1551 "http://www.w3.org/2001/XMLSchema#string" is assigned the literal value "Your  
 1552 medical record has been accessed by:"

1553 [109]-[114] The **obligation** parameter  
 1554 "urn:oasis:names:tc:xacml:1.0:examples:attribute:text" of the  
 1555 "http://www.w3.org/2001/XMLSchema#string" data-type is assigned the value of the  
 1556 "urn:oasis:names:tc:xacml:1.0:subject:subject-id" **subject attribute**.

#### 1557 4.2.4.4. Rule 4

1558 Rule 4 illustrates the use of the "Deny" Effect value, and a Rule with no Condition element.

```

1559 [01] <?xml version="1.0" encoding="UTF-8"?>
1560 [02] <Rule
1561 [03] xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1562 [04] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1563 [05] xmlns:ctx="urn:oasis:names:tc:xacml:1.0:context"
1564 [06] xmlns:md="http://www.medico.com/schemas/record.xsd"
1565 [07] RuleId="urn:oasis:names:tc:xacml:example:ruleid:4"
1566 [08] Effect="Deny">
1567 [09] <Description>
1568 [10]     An Administrator shall not be permitted to read or write
1569 [11]     medical elements of a patient record in the
1570 [12]     http://www.medico.com/records.xsd namespace.
1571 [13] </Description>
1572 [14] <Target>
1573 [15]     <Subjects>
1574 [16]         <Subject>
1575 [17]             <!-- match role subject attribute -->
1576 [18]             <SubjectMatch
1577 [19]                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1578 [20]                 <AttributeValue
1579 [21]                     DataType="http://www.w3.org/2001/XMLSchema#string">administ
1580 [22]                 r</AttributeValue>
1581 [23]                 <SubjectAttributeDesignator AttributeId=
1582 [24]                     "urn:oasis:names:tc:xacml:1.0:example:attribute:role"
1583 [25]                 DataType="http://www.w3.org/2001/XMLSchema#string"/>
1584 [26]             </SubjectMatch>
1585 [27]         </Subject>
1586 [28]     </Subjects>
1587 [29]     <Resources>
1588 [30]         <Resource>
1589 [31]             <!-- match document target namespace -->
1590 [32]             <ResourceMatch
1591 [33]                 MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1592 [34]                 <AttributeValue
1593 [35]                     DataType="http://www.w3.org/2001/XMLSchema#string">

```

```

1594 [30] http://www.medico.com/schemas/record.xsd
1595 [31] </AttributeValue>
1596 [32] <ResourceAttributeDesignator AttributeId=
1597 [33] "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1598 [34] DataType="http://www.w3.org/2001/XMLSchema#string" />
1599 [35] </ResourceMatch>
1600 [36] <!-- match requested xml element -->
1601 [37] <ResourceMatch
1602 [38] MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-match">
1603 [39] <AttributeValue
1604 [40] DataType="http://www.w3.org/2001/XMLSchema#string">
1605 [41] /md:record/md:medical
1606 [42] </AttributeValue>
1607 [43] <ResourceAttributeDesignator AttributeId=
1608 [44] "urn:oasis:names:tc:xacml:1.0:resource:xpath"
1609 [45] DataType="http://www.w3.org/2001/XMLSchema#string" />
1610 [46] </ResourceMatch>
1611 [47] </Resource>
1612 [48] </Resources>
1613 [49] <Actions>
1614 [50] <Action>
1615 [51] <!-- match 'read' action -->
1616 [52] <ActionMatch
1617 [53] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1618 [54] <AttributeValue
1619 [55] DataType="http://www.w3.org/2001/XMLSchema#string">
1620 [56] read
1621 [57] </AttributeValue>
1622 [58] <ActionAttributeDesignator AttributeId=
1623 [59] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1624 [60] DataType="http://www.w3.org/2001/XMLSchema#string" />
1625 [61] </ActionMatch>
1626 [62] </Action>
1627 [63] <Action>
1628 [64] <!-- match 'write' action -->
1629 [65] <ActionMatch
1630 [66] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1631 [67] <AttributeValue
1632 [68] DataType="http://www.w3.org/2001/XMLSchema#string">
1633 [69] write
1634 [70] </AttributeValue>
1635 [71] <ActionAttributeDesignator AttributeId=
1636 [72] "urn:oasis:names:tc:xacml:1.0:action:action-id"
1637 [73] DataType="http://www.w3.org/2001/XMLSchema#string" />
1638 [74] </ActionMatch>
1639 [75] </Action>
1640 [76] </Actions>
1641 [77] </Target>
1642 [78] </Rule>

```

1643 [01]-[08] The `Rule` element declaration. The most important parameter here is `Effect`. See Rule  
1644 1, Section 4.2.4.1 for a detailed explanation of the `Rule` structure.

1645 [08] **Rule Effect**. Every **rule** that evaluates to “True” emits **rule effect** as its value that will be  
1646 combined later on with other **rule effects** according to the **rule combining algorithm**. This **rule**  
1647 **Effect** is “Deny” meaning that according to this rule, access must be denied.

1648 [09]-[13] Free form description of the **rule**.

1649 [14]-[63] **Rule target**. The **Rule target** defines a set of **decision requests** that are applicable to  
1650 the **rule**. This **rule** is matched by:

- 1651 • a **decision request** with **subject attribute**
- 1652 "urn:oasis:names:tc:xacml:1.0:examples:attribute:role" equal to
- 1653 "administrator";
- 1654 • the value of **resource attribute**
- 1655 "urn:oasis:names:tc:xacml:1.0:resource:target-namespace" is equal to
- 1656 "http://www.medico.com/schemas/record.xsd"
- 1657 • the value of the requested XML element matches the XPath expression
- 1658 "/md:record/md:medical";
- 1659 • the value of **action attribute** "urn:oasis:names:tc:xacml:1.0:action:action-id" is equal to
- 1660 "read"

1661 See Rule 1, Section 4.2.4.1 for the detailed explanation of the Target element.

1662 This **rule** does not have a Condition element.

#### 4.2.4.5. Example PolicySet

1664 This section uses the examples of the previous sections to illustrate the process of combining

1665 **policies**. The policy governing read access to medical elements of a record is formed from each of

1666 the four **rules** described in Section 4.2.3. In plain language, the combined rule is:

- 1667 • Either the requestor is the patient; or
- 1668 • the requestor is the parent or guardian and the patient is under 16; or
- 1669 • the requestor is the primary care physician and a notification is sent to the patient; and
- 1670 • the requestor is not an administrator.

1671 The following XACML <PolicySet> illustrates the combined **policies**. **Policy 3** is included by

1672 reference and **policy 2** is explicitly included.

```

1673 [01] <?xml version="1.0" encoding="UTF-8"?>
1674 [02] <PolicySet
1675 [03]   xmlns="urn:oasis:names:tc:xacml:1.0:policy"
1676 [04]   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1677 [05]   PolicySetId=
1678 [06]   "urn:oasis:names:tc:xacml:1.0:examples:policysetid:1"
1679 [07]   PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
1680 [071] policy-combining-algorithm:deny-overrides"/>
1681 [08] <Description>
1682 [09]   Example policy set.
1683 [10] </Description>
1684 [11] <Target>
1685 [12]   <Subjects>
1686 [13]     <Subject>
1687 [14]       <!-- any subject -->
1688 [15]       <AnySubject/>
1689 [16]     </Subject>
1690 [17]   </Subjects>
1691 [18]   <Resources>
1692 [19]     <Resource>
1693 [20]       <!-- any resource in the target namespace -->
1694 [21]       <ResourceMatch
1695 [22]         MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
1696 [23]           <AttributeValue
1697 [24]             DataType="http://www.w3.org/2001/XMLSchema#string">
1698 [25]               http://www.medico.com/records.xsd

```

```

1699 [24]         </AttributeValue>
1700 [25]         <ResourceAttributeDesignator AttributeId=
1701 [26]         "urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
1702 [26]         DataType="http://www.w3.org/2001/XMLSchema#string" />
1703 [27]         </ResourceMatch>
1704 [28]         </Resource>
1705 [29]     </Resources>
1706 [30]     <Actions>
1707 [31]         <Action>
1708 [32]             <!-- any action -->
1709 [33]             <AnyAction/>
1710 [34]         </Action>
1711 [35]     </Actions>
1712 [36] </Target>
1713 [37] <!-- include policy from the example 3 by reference -->
1714 [38] <PolicyIdReference>
1715 [39]     urn:oasis:names:tc:xacml:1.0:examples:policyid:3
1716 [40] </PolicyIdReference>
1717 [41]     <!-- policy 2 combines rules from the examples 1, 2,
1718 [42]     and 4 is included by value. -->
1719 [43] <Policy
1720 [44]     PolicyId="urn:oasis:names:tc:xacml:examples:policyid:2"
1721 [45]     RuleCombiningAlgId=
1722 [46]     "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
1723 [46]     overrides">
1724 [47]     <Description>
1725 [48]         Policy for any medical record in the
1726 [49]         http://www.medico.com/schemas/record.xsd namespace
1727 [50]     </Description>
1728 [51]     <Target> ... </Target>
1729 [52]     <Rule
1730 [53]         RuleId="urn:oasis:names:tc:xacml:examples:ruleid:1"
1731 [54]         Effect="Permit"> ... </Rule>
1732 [55]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:2"
1733 [56]         Effect="Permit"> ... </Rule>
1734 [57]     <Rule RuleId="urn:oasis:names:tc:xacml:examples:ruleid:4"
1735 [58]         Effect="Deny"> ... </Rule>
1736 [59]     <Obligations> ... </Obligations>
1737 [60] </Policy>
1738 [61] </PolicySet>
1739

```

1740 [02]-[07] PolicySet declaration. Standard XML namespace declarations are included as well as  
1741 PolicySetId, and **policy combining algorithm** identifier.

1742 [05]-[06] PolicySetId is used for identifying this **policy set** and for possible inclusion of this  
1743 **policy set** into another **policy set**.

1744 [07] **Policy combining algorithm** identifier. Policies in the **policy set** are combined according to  
1745 the specified **policy combining algorithm** identifier when the **authorization decision** is  
1746 computed.

1747 [08]-[10] Free form description of the **policy set**.

1748 [11]-[36] PolicySet Target element defines a set of **decision requests** that are applicable to  
1749 this PolicySet.

1750 [38]-[40] PolicyIdReference includes **policy** by id.

1751 [43]-[60] **Policy 2** is explicitly included in this **policy set**.

---

## 1752 5. Policy syntax (normative, with the exception of 1753 the schema fragments)

### 1754 5.1. Element <PolicySet>

1755 The <PolicySet> element is a top-level element in the XACML policy schema. <PolicySet> is  
1756 an aggregation of other *policy sets* and *policies*. *Policy sets* MAY be included in an enclosing  
1757 <PolicySet> element either directly using the <PolicySet> element or indirectly using the  
1758 <PolicySetIdReference> element. *Policies* MAY be included in an enclosing <PolicySet>  
1759 element either directly using the <Policy> element or indirectly using the <PolicyIdReference>  
1760 element.

1761 If a <PolicySet> element contains references to other *policy sets* or *policies* in the form of  
1762 URLs, then these references MAY be resolvable.

1763 *Policies* included in the <PolicySet> element MUST be combined using the algorithm specified  
1764 by the PolicyCombiningAlgId attribute. <PolicySet> is treated exactly like a <Policy> in all  
1765 the *policy combining algorithms*.

1766 The <Target> element defines the applicability of the <PolicySet> to a set of *decision*  
1767 *requests*. If the <Target> element within <PolicySet> matches the *request context*, then the  
1768 <PolicySet> element MAY be used by the *PDP* in making its *authorization decision*.

1769 The <Obligations> element contains a set of *obligations* that MUST be fulfilled by the *PEP* in  
1770 conjunction with the *authorization decision*. If the *PEP* does not understand any of the  
1771 *obligations*, then it MUST act as if the *PDP* had returned a “Deny” *authorization decision* value.

```
1772 <xs:element name="PolicySet" type="xacml:PolicySetType"/>  
1773 <xs:complexType name="PolicySetType">  
1774 <xs:sequence>  
1775 <xs:element ref="xacml:Description" minOccurs="0"/>  
1776 <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>  
1777 <xs:element ref="xacml:Target"/>  
1778 <xs:choice minOccurs="0" maxOccurs="unbounded">  
1779 <xs:element ref="xacml:PolicySet"/>  
1780 <xs:element ref="xacml:Policy"/>  
1781 <xs:element ref="xacml:PolicySetIdReference"/>  
1782 <xs:element ref="xacml:PolicyIdReference"/>  
1783 </xs:choice>  
1784 <xs:element ref="xacml:Obligations" minOccurs="0"/>  
1785 </xs:sequence>  
1786 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>  
1787 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI"  
1788 use="required"/>  
1789 </xs:complexType>
```

1790 The <PolicySet> element is of **PolicySetType** complex type.

1791 The <PolicySet> element contains the following attributes and elements:

1792 PolicySetId [Required]

1793 *Policy set* identifier. It is the responsibility of the *PAP* to ensure that no two *policies*  
1794 visible to the *PDP* have the same identifier. This MAY be achieved by following a  
1795 predefined URN or URI scheme. If the *policy set* identifier is in the form of a URL, then it  
1796 MAY be resolvable.

1797

1798 PolicyCombiningAlgId [Required]

1799 The identifier of the *policy-combining algorithm* by which the <PolicySet>  
1800 components MUST be combined. Standard *policy-combining algorithms* are listed in  
1801 Appendix C. Standard *policy-combining algorithm* identifiers are listed in Section B.10.

1802 <Description> [Optional]

1803 A free-form description of the <PolicySet>.

1804 <PolicySetDefaults> [Optional]

1805 A set of default values applicable to the <PolicySet>. The scope of the  
1806 <PolicySetDefaults> element SHALL be the enclosing *policy set*.

1807 <Target> [Required]

1808 The <Target> element defines the applicability of a <PolicySet> to a set of *decision*  
1809 *requests*.

1810 The <Target> element MAY be declared by the creator of the <PolicySet> or it MAY be  
1811 computed from the <Target> elements of the referenced <Policy> elements, either as  
1812 an intersection or as a union.

1813 <PolicySet> [Any Number]

1814 A *policy set* component that is included in this *policy set*.

1815 <Policy> [Any Number]

1816 A *policy* component that is included in this *policy set*.

1817 <PolicySetIdReference> [Any Number]

1818 A reference to a <PolicySet> component that MUST be included in this *policy set*. If  
1819 <PolicySetIdReference> is a URL, then it MAY be resolvable.

1820 <PolicyIdReference> [Any Number]

1821 A reference to a <Policy> component that MUST be included in this *policy set*. If the  
1822 <PolicyIdReference> is a URL, then it MAY be resolvable.

1823 <Obligations> [Optional]

1824 Contains the set of <Obligation> elements. See Section 7.11 for a description of how  
1825 the set of *obligations* to be returned by the *PDP* shall be determined.

## 1826 5.2. Element <Description>

1827 The <Description> element is used for a free-form description of the <PolicySet> element,  
1828 <Policy> element and <Rule> element. The <Description> element is of **xs:string** simple  
1829 type.

```
1830 <xs:element name="Description" type="xs:string"/>
```

## 1831 5.3. Element <PolicySetDefaults>

1832 The <PolicySetDefaults> element SHALL specify default values that apply to the  
1833 <PolicySet> element.

```

1834 <xs:element name="PolicySetDefaults" type="xacml:DefaultsType"/>
1835 <xs:complexType name="DefaultsType">
1836   <xs:sequence>
1837     <xs:choice>
1838       <xs:element ref="xacml:XPathVersion" minOccurs="0"/>
1839     </xs:choice>
1840   </xs:sequence>
1841 </xs:complexType>

```

1842 <PolicySetDefaults> element is of **DefaultsType** complex type.

1843 The <PolicySetDefaults> element contains the following elements:

1844 <XPathVersion> [Optional]

1845         Default XPath version.

## 1846 **5.4. Element <XPathVersion>**

1847 The <XPathVersion> element SHALL specify the version of the XPath specification to be used by  
1848 <AttributeSelector> elements.

```

1849 <xs:element name="XPathVersion" type="xs:anyURI"/>

```

1850 The URI for the XPath 1.0 specification is "<http://www.w3.org/TR/1999/Rec-xpath-19991116>". The <XPathVersion> element is REQUIRED if the XACML enclosing **policy set**  
1851 or **policy** contains <AttributeSelector> elements or XPath-based functions.  
1852

## 1853 **5.5. Element <Target>**

1854 The <Target> element identifies the set of **decision requests** that the parent element is intended  
1855 to evaluate. The <Target> element SHALL appear as a child of <PolicySet>, <Policy> and  
1856 <Rule> elements. It contains definitions for **subjects**, **resources** and **actions**.

1857 The <Target> element SHALL contain a **conjunctive sequence** of <Subjects>, <Resources>  
1858 and <Actions> elements. For the parent of the <Target> element to be applicable to the  
1859 **decision request**, there MUST be at least one positive match between each section of the  
1860 <Target> element and the corresponding section of the <xacml-context:Request> element.

```

1861 <xs:element name="Target" type="xacml:TargetType"/>
1862 <xs:complexType name="TargetType">
1863   <xs:sequence>
1864     <xs:element ref="xacml:Subjects"/>
1865     <xs:element ref="xacml:Resources"/>
1866     <xs:element ref="xacml:Actions"/>
1867   </xs:sequence>
1868 </xs:complexType>

```

1869 The <Target> element is of **TargetType** complex type.

1870 The <Target> element contains the following elements:

1871 <Subjects> [Required]

1872         Matching specification for the **subject attributes** in the **context**.

1873 <Resources> [Required]

1874         Matching specification for the **resource attributes** in the **context**.

1875

1876 <Actions> [Required]

1877 Matching specification for the *action attributes* in the *context*.

## 1878 5.6. Element <Subjects>

1879 The <Subjects> element SHALL contain a *disjunctive sequence* of <Subject> elements.

```
1880 <xs:element name="Subjects" type="xacml:SubjectsType"/>
1881 <xs:complexType name="SubjectsType">
1882   <xs:choice>
1883     <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
1884     <xs:element ref="xacml:AnySubject"/>
1885   </xs:choice>
1886 </xs:complexType>
```

1887 The <Subjects> element is of **SubjectsType** complex type.

1888 The <Subjects> element contains the following elements:

1889 <Subject> [One To Many, Required Choice]

1890 See Section 5.7.

1891 <AnySubject> [Required Choice]

1892 See Section 5.8.

## 1893 5.7. Element <Subject>

1894 The <Subject> element SHALL contain a *conjunctive sequence* of <SubjectMatch> elements.

```
1896 <xs:element name="Subject" type="xacml:SubjectType"/>
1897 <xs:complexType name="SubjectType">
1898   <xs:sequence>
1899     <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
1900   </xs:sequence>
1901 </xs:complexType>
```

1902 The <Subject> element is of **SubjectType** complex type.

1903 The <Subject> element contains the following elements:

1904 <SubjectMatch> [One to Many]

1905 A *conjunctive sequence* of individual matches of the *subject attributes* in the *context*  
1906 and the embedded *attribute* values.

## 1907 5.8. Element <AnySubject>

1908 The <AnySubject> element SHALL match any *subject attribute* in the *context*.

```
1909 <xs:element name="AnySubject"/>
```

## 1910 5.9. Element <SubjectMatch>

1911 The <SubjectMatch> element SHALL identify a set of *subject*-related entities by matching  
1912 *attribute* values in a <xacml-context:Subject> element of the *context* with the embedded  
1913 *attribute* value.

```

1914 <xs:element name="SubjectMatch" type="xacml:SubjectMatchType" />
1915 <xs:complexType name="SubjectMatchType">
1916   <xs:sequence>
1917     <xs:element ref="xacml:AttributeValue" />
1918     <xs:choice>
1919       <xs:element ref="xacml:SubjectAttributeDesignator" />
1920       <xs:element ref="xacml:AttributeSelector" />
1921     </xs:choice>
1922   </xs:sequence>
1923   <xs:attribute name="MatchId" type="xs:anyURI" use="required" />
1924 </xs:complexType>

```

1925 The <SubjectMatch> element is of **SubjectMatchType** complex type.

1926 The <SubjectMatch> element contains the following attributes and elements:

1927 MatchId [Required]

1928           Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI** with  
1929           legal values documented in Section A.12.

1930 <AttributeValue> [Required]

1931       Embedded **attribute** value.

1932 <SubjectAttributeDesignator> [Required choice]

1933           Identifies one or more **attribute** values in a <Subject> element of the **context**.

1934 <AttributeSelector> [Required choice]

1935           MAY be used to identify one or more **attribute** values in the request **context**. The XPath  
1936           expression SHOULD resolve to an **attribute** in a <Subject> element of the **context**.

## 1937 **5.10. Element <Resources>**

1938 The <Resources> element SHALL contain a **disjunctive sequence** of <Resource> elements.

```

1939 <xs:element name="Resources" type="xacml:ResourcesType" />
1940 <xs:complexType name="ResourcesType">
1941   <xs:choice>
1942     <xs:element ref="xacml:Resource" maxOccurs="unbounded" />
1943     <xs:element ref="xacml:AnyResource" />
1944   </xs:choice>
1945 </xs:complexType>

```

1946 The <Resources> element is of **ResourcesType** complex type.

1947 The <Resources> element contains the following elements:

1948 <Resource> [One To Many, Required Choice]

1949       See Section 5.11.

1950 <AnyResource> [Required Choice]

1951       See Section 5.12.

## 1952 **5.11. Element <Resource>**

1953 The <Resource> element SHALL contain a **conjunctive sequence** of <ResourceMatch>  
1954 elements.

```

1955 <xs:element name="Resource" type="xacml:ResourceType" />
1956 <xs:complexType name="ResourceType">
1957   <xs:sequence>
1958     <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded" />
1959   </xs:sequence>
1960 </xs:complexType>

```

1961 The <Resource> element is of **ResourceType** complex type.

1962 The <Resource> element contains the following elements:

1963 <ResourceMatch> [One to Many]

1964 A **conjunctive sequence** of individual matches of the **resource attributes** in the **context**  
 1965 and the embedded **attribute** values.

## 5.12. Element <AnyResource>

1967 The <AnyResource> element SHALL match any **resource attribute** in the **context**.

```

1968 <xs:element name="AnyResource" />

```

## 5.13. Element <ResourceMatch>

1970 The <ResourceMatch> element SHALL identify a set of **resource**-related entities by matching  
 1971 **attribute** values in the <xacml-context:Resource> element of the **context** with the embedded  
 1972 **attribute** value.

```

1973 <xs:element name="ResourceMatch" type="xacml:ResourceMatchType" />
1974 <xs:complexType name="ResourceMatchType">
1975   <xs:sequence>
1976     <xs:element ref="xacml:AttributeValue" />
1977     <xs:choice>
1978       <xs:element ref="xacml:ResourceAttributeDesignator" />
1979       <xs:element ref="xacml:AttributeSelector" />
1980     </xs:choice>
1981   </xs:sequence>
1982   <xs:attribute name="MatchId" type="xs:anyMatch" use="required" />
1983 </xs:complexType>

```

1984 The <ResourceMatch> element is of **ResourceMatchType** complex type.

1985 The <ResourceMatch> element contains the following attributes and elements:

1986 MatchId [Required]

1987 Specifies a matching function. Values of this attribute MUST be of type **xs:anyURI**, with  
 1988 legal values documented in Section A.12.

1989 <AttributeValue> [Required]

1990 Embedded **attribute** value.

1991 <ResourceAttributeDesignator> [Required Choice]

1992 Identifies one or more **attribute** values in the <Resource> element of the **context**.

1993 <AttributeSelector> [Required Choice]

1994 MAY be used to identify one or more **attribute** values in the request **context**. The XPath  
 1995 expression SHOULD resolve to an **attribute** in the <Resource> element of the **context**.

1996

## 5.14. Element <Actions>

1997

The <Actions> element SHALL contain a **disjunctive sequence** of <Action> elements.

1998

```
<xs:element name="Actions" type="xacml:ActionTypes" />
```

1999

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

2000

```
<xs:choice>
```

2001

```
<xs:element ref="xacml:Action" maxOccurs="unbounded" />
```

2002

```
<xs:element ref="xacml:AnyAction" />
```

2003

```
</xs:choice>
```

2004

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

2005

The <Actions> element is of **ActionTypes** complex type.

2006

The <Actions> element contains the following elements:

2007

<Action> [One To Many, Required Choice]

2008

See Section 5.15.

2009

<AnyAction> [Required Choice]

2010

See Section 5.16.

2011

## 5.15. Element <Action>

2012

The <Action> element SHALL contain a **conjunctive sequence** of <ActionMatch> elements.

2013

```
<xs:element name="Action" type="xacml:ActionType" />
```

2014

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

2015

```
<xs:sequence>
```

2016

```
<xs:element ref="xacml:ActionMatch" maxOccurs="unbounded" />
```

2017

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

2018

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

2019

The <Action> element is of **ActionType** complex type.

2020

The <Action> element contains the following elements:

2021

<ActionMatch> [One to Many]

2022

A **conjunctive sequence** of individual matches of the **action** attributes in the **context** and the embedded **attribute** values.

2023

2024

## 5.16. Element <AnyAction>

2025

The <AnyAction> element SHALL match any **action attribute** in the **context**.

2026

```
<xs:element name="AnyAction" />
```

2027

2028

## 5.17. Element <ActionMatch>

2029

The <ActionMatch> element SHALL identify a set of **action**-related entities by matching **attribute** values in the <xacml-context:Action> element of the **context** with the embedded **attribute** value.

2030

2031

2032

```
<xs:element name="ActionMatch" type="xacml:ActionMatchType" />
```

2033

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

2034

```
<xs:sequence>
```

2035

```
<xs:element ref="xacml:AttributeValue" />
```

```

2036     <xs:choice>
2037         <xs:element ref="xacml:ActionAttributeDesignator" />
2038         <xs:element ref="xacml:AttributeSelector" />
2039     </xs:choice>
2040 </xs:sequence>
2041 <xs:attribute name="MatchId" type="xs:anyURI" use="required" />
2042 </xs:complexType>

```

2043 The <ActionMatch> element is of **ActionMatchType** complex type.

2044 The <ActionMatch> element contains the following attributes and elements:

2045 MatchId [Required]

2046 Specifies a matching function. The value of this attribute MUST be of type **xs:anyURI**, with  
 2047 legal values documented in Section A.12.

2048 <AttributeValue> [Required]

2049 Embedded **attribute** value.

2050 <ActionAttributeDesignator> [Required Choice]

2051 Identifies one or more **attribute** values in the <Action> element of the **context**.

2052 <AttributeSelector> [Required Choice]

2053 MAY be used to identify one or more **attribute** values in the request **context**. The XPath  
 2054 expression SHOULD resolve to an **attribute** in the <Action> element of the **context**.

## 2055 **5.18. Element <PolicySetIdReference>**

2056 The <PolicySetIdReference> element SHALL be used to reference a <PolicySet> element  
 2057 by id. If <PolicySetIdReference> is a URL, then it MAY be resolvable to the <PolicySet>.  
 2058 The mechanism for resolving a **policy set** reference to the corresponding **policy set** is outside the  
 2059 scope of this specification.

```

2060 <xs:element name="PolicySetIdReference" type="xs:anyURI" />

```

2061 Element <PolicySetIdReference> is of **xs:anyURI** simple type.

## 2062 **5.19. Element <PolicyIdReference>**

2063 The <xacml:PolicyIdReference> element SHALL be used to reference a <Policy> element  
 2064 by id. If <PolicyIdReference> is a URL, then it MAY be resolvable to the <Policy>. The  
 2065 mechanism for resolving a **policy** reference to the corresponding **policy** is outside the scope of this  
 2066 specification.

```

2067 <xs:element name="PolicyIdReference" type="xs:anyURI" />

```

2068 Element <PolicyIdReference> is of **xs:anyURI** simple type.

## 2069 **5.20. Element <Policy>**

2070 The <Policy> element is the smallest entity that SHALL be presented to the **PDP** for evaluation.

2071 The main components of this element are the <Target>, <Rule> and <Obligations> elements  
 2072 and the RuleCombiningAlgId attribute.

2073 The <Target> element SHALL define the applicability of the <Policy> to a set of **decision**  
2074 **requests**.

2075 **Rules** included in the <Policy> element MUST be combined by the algorithm specified by the  
2076 RuleCombiningAlgId attribute.

2077 The <Obligations> element SHALL contain a set of **obligations** that MUST be fulfilled by the  
2078 **PDP** in conjunction with the **authorization decision**.

```
2079 <xs:element name="Policy" type="xacml:PolicyType"/>
2080 <xs:complexType name="PolicyType">
2081   <xs:sequence>
2082     <xs:element ref="xacml:Description" minOccurs="0"/>
2083     <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
2084     <xs:element ref="xacml:Target"/>
2085     <xs:element ref="xacml:Rule" minOccurs="0" maxOccurs="unbounded"/>
2086     <xs:element ref="xacml:Obligations" minOccurs="0"/>
2087   </xs:sequence>
2088   <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
2089   <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="required"/>
2090 </xs:complexType>
```

2091 The <Policy> element is of **PolicyType** complex type.

2092 The <Policy> element contains the following attributes and elements:

2093 PolicyId [Required]

2094 **Policy** identifier. It is the responsibility of the **PAP** to ensure that no two **policies** visible to  
2095 the **PDP** have the same identifier. This MAY be achieved by following a predefined URN or  
2096 URI scheme. If the **policy** identifier is in the form of a URL, then it MAY be resolvable.

2097 RuleCombiningAlgId [Required]

2098 The identifier of the rule-combining algorithm by which the <Policy> components MUST  
2099 be combined. Standard rule-combining algorithms are listed in Appendix C. Standard rule-  
2100 combining algorithm identifiers are listed in Section B.10.

2101 <Description> [Optional]

2102 A free-form description of the **policy**. See Section 5.2 Element <Description>.

2103 <PolicyDefaults> [Optional]

2104 Defines a set of default values applicable to the **policy**. The scope of the  
2105 <PolicyDefaults> element SHALL be the enclosing policy.

2106 <Target> [Required]

2107 The <Target> element SHALL define the applicability of a <Policy> to a set of **decision**  
2108 **requests**.

2109 The <Target> element MAY be declared by the creator of the <Policy> element, or it  
2110 MAY be computed from the <Target> elements of the referenced <Rule> elements either  
2111 as an intersection or as a union.

2112 <Rule> [Any Number]

2113 A sequence of authorizations that MUST be combined according to the  
2114 RuleCombiningAlgId attribute. **Rules** whose <Target> elements match the **decision**  
2115 **request** MUST be considered. **Rules** whose <Target> elements do not match the  
2116 **decision request** SHALL be ignored.

2117 <Obligations> [Optional]

2118 A **conjunctive sequence** of **obligations** that MUST be fulfilled by the **PEP** in conjunction  
2119 with the **authorization decision**. See Section 7.11 for a description of how the set of  
2120 **obligations** to be returned by the **PDP** SHALL be determined.

## 2121 5.21. Element <PolicyDefaults>

2122 The <PolicyDefaults> element SHALL specify default values that apply to the <Policy>  
2123 element.

```
2124 <xs:element name="PolicyDefaults" type="xacml:DefaultsType"/>  
2125 <xs:complexType name="DefaultsType">  
2126 <xs:sequence>  
2127 <xs:choice>  
2128 <xs:element ref="xacml:XPathVersion" minOccurs="0"/>  
2129 </xs:choice>  
2130 </xs:sequence>  
2131 </xs:complexType>
```

2132 <PolicyDefaults> element is of **DefaultsType** complex type.

2133 The <PolicyDefaults> element contains the following elements:

2134 <XPathVersion> [Optional]

2135 Default XPath version.

## 2136 5.22. Element <Rule>

2137 The <Rule> element SHALL define the individual **rules** in the **policy**. The main components of  
2138 this element are the <Target> and <Condition> elements and the **Effect** attribute.

```
2139 <xs:element name="Rule" type="xacml:RuleType"/>  
2140 <xs:complexType name="RuleType">  
2141 <xs:sequence>  
2142 <xs:element ref="xacml:Description" minOccurs="0"/>  
2143 <xs:element ref="xacml:Target" minOccurs="0"/>  
2144 <xs:element ref="xacml:Condition" minOccurs="0"/>  
2145 </xs:sequence>  
2146 <xs:attribute name="RuleId" type="xs:anyURI" use="required"/>  
2147 <xs:attribute name="Effect" type="xacml:EffectType" use="required"/>  
2148 </xs:complexType>
```

2149 The <Rule> element is of **RuleType** complex type.

2150 The <Rule> element contains the following attributes and elements:

2151 RuleId [Required]

2152 A URN identifying this **rule**.

2153 Effect [Required]

2154 **Rule effect**. Values of this attribute are either "Permit" or "Deny".

2155 <Description> [Optional]

2156 A free-form description of the **rule**.

2157

2158 <Target> [Optional]  
2159 Identifies the set of **decision requests** that the <Rule> element is intended to evaluate. If  
2160 this element is omitted, then the **target** for the <Rule> SHALL be defined by the  
2161 <Target> element of the enclosing <Policy> element. See Section 5.5 for details.

2162 <Condition> [Optional]  
2163 A **predicate** that MUST be satisfied for the **rule** to be assigned its Effect value. A  
2164 **condition** is a boolean function over a combination of **subject, resource, action** and  
2165 **environment attributes** or other functions.

## 2166 5.23. Simple type EffectType

2167 The **EffectType** simple type defines the values allowed for the Effect attribute of the <Rule>  
2168 element and for the FulfillOn attribute of the <Obligation> element.

```
2169 <xs:simpleType name="EffectType">  
2170 <xs:restriction base="xs:string">  
2171 <xs:enumeration value="Permit"/>  
2172 <xs:enumeration value="Deny"/>  
2173 </xs:restriction>  
2174 </xs:simpleType>
```

## 2175 5.24. Element <Condition>

2176 The <Condition> element is a boolean function over **subject, resource, action** and  
2177 **environment attributes** or functions of **attributes**. If the <Condition> element evaluates to  
2178 "True", then the enclosing <Rule> element is assigned its Effect value.

```
2179 <xs:element name="Condition" type="xacml:ApplyType"/>
```

2180 The <Condition> element is of **ApplyType** complex type.

## 2181 5.25. Element <Apply>

2182 The <Apply> element denotes application of a function to its arguments, thus encoding a function  
2183 call. The <Apply> element can be applied to any combination of <Apply> ,  
2184 <AttributeValue> , <SubjectAttributeDesignator> ,  
2185 <ResourceAttributeDesignator> , <ActionAttributeDesignator> ,  
2186 <EnvironmentAttributeDesignator> and <AttributeSelector> arguments.

```
2187 <xs:element name="Apply" type="xacml:ApplyType"/>  
2188 <xs:complexType name="ApplyType">  
2189 <xs:choice minOccurs="0" maxOccurs="unbounded">  
2190 <xs:element ref="xacml:Function"/>  
2191 <xs:element ref="xacml:Apply"/>  
2192 <xs:element ref="xacml:AttributeValue"/>  
2193 <xs:element ref="xacml:SubjectAttributeDesignator"/>  
2194 <xs:element ref="xacml:ResourceAttributeDesignator"/>  
2195 <xs:element ref="xacml:ActionAttributeDesignator"/>  
2196 <xs:element ref="xacml:EnvironmentAttributeDesignator"/>  
2197 <xs:element ref="xacml:AttributeSelector"/>  
2198 </xs:choice>  
2199 <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>  
2200 </xs:complexType>
```

2201 The <Apply> element is of **ApplyType** complex type.

2202 The <Apply> element contains the following attributes and elements:

- 2203 FunctionId [Required]
- 2204 The URN of a function. XACML-defined functions are described in Appendix A.
- 2205 <Function> [Optional]
- 2206 The name of a function that is applied to the elements of a **bag**. See Section A14.11.
- 2207 <Apply> [Optional]
- 2208 A nested function-call argument.
- 2209 <AttributeValue> [Optional]
- 2210 A literal value argument.
- 2211 <SubjectAttributeDesignator> [Optional]
- 2212 A **subject attribute** argument.
- 2213 <ResourceAttributeDesignator> [Optional]
- 2214 A **resource attribute** argument.
- 2215 <ActionAttributeDesignator> [Optional]
- 2216 An **action attribute** argument.
- 2217 <EnvironmentAttributeDesignator> [Optional]
- 2218 An **environment attribute** argument.
- 2219 <AttributeSelector> [Optional]
- 2220 An **attribute** selector argument.

## 2221 5.26. Element <Function>

2222 The `Function` element SHALL be used to name a function that is applied by the higher-order **bag**  
 2223 functions to every element of a **bag**. The higher-order **bag** functions are described in Section  
 2224 A14.11.

```
2225 <xs:element name="Function" type="xacml:FunctionType"/>
2226 <xs:complexType name="FunctionType">
2227   <xs:attribute name="FunctionId" type="xs:anyURI" use="required"/>
2228 </xs:complexType>
```

2229 The `Function` element is of **FunctionType** complex type.

2230 The `Function` element contains the following attributes:

2231 FunctionId [Required]

2232 The identifier for the function that is applied to the elements of a **bag** by the higher-order **bag**  
 2233 functions.

## 2234 5.27. Complex type AttributeDesignatorType

2235 The **AttributeDesignatorType** complex type is the type for elements and extensions that identify  
 2236 **attributes**. An element of this type contains properties by which it MAY be matched to **attributes**  
 2237 in the request **context**.

2238 In addition, elements of this type MAY control behaviour in the event that no matching **attribute** is  
2239 present in the **context**.

2240 Elements of this type SHALL NOT alter the match semantics of named **attributes**, but MAY narrow  
2241 the search space.

```
2242 <xs:complexType name="AttributeDesignatorType">  
2243   <xs:attribute name="AttributeId" type="xs:anyURI" use="required" />  
2244   <xs:attribute name="DataType" type="xs:anyURI" use="required" />  
2245   <xs:attribute name="Issuer" type="xs:string" use="optional" />  
2246   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2247   default="false" />  
2248 </xs:complexType>
```

2249 A named **attribute** SHALL match an **attribute** if the values of their respective AttributeId,  
2250 DataType and Issuer attributes match. The **attribute** designator's AttributeId MUST match,  
2251 by URI equality, the AttributeId of the **attribute**. The **attribute** designator's DataType MUST  
2252 match, by URI equality, the DataType of the same **attribute**.

2253 If the Issuer attribute is present in the **attribute** designator, then it MUST match, by string  
2254 equality, the Issuer of the same **attribute**. If the Issuer is not present in the **attribute**  
2255 designator, then the matching of the **attribute** to the named **attribute** SHALL be governed by  
2256 AttributeId and DataType attributes alone.

2257 The <AttributeDesignatorType> contains the following attributes:

2258 AttributeId [Required]

2259 This attribute SHALL specify the AttributeId with which to match the **attribute**.

2260 DataType [Required]

2261 This attribute SHALL specify the data-type with which to match the **attribute**.

2262 Issuer [Optional]

2263 This attribute, if supplied, SHALL specify the Issuer with which to match the **attribute**.

2264 MustBePresent [Optional]

2265 This attribute governs whether the element returns "Indeterminate" in the case where the  
2266 named **attribute** is absent. If the *named attribute* is absent and MustBePresent is "True",  
2267 then this element SHALL result in "Indeterminate". The default value SHALL be "False".

## 2268 5.28. Element <SubjectAttributeDesignator>

2269 The <SubjectAttributeDesignator> element is of the **SubjectAttributeDesignatorType**.  
2270 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType**  
2271 complex type. It is the base type for elements and extensions that refer to *named categorized*  
2272 **subject attributes**. A *named categorized subject attribute* is defined as follows:

2273 A **subject** is represented by a <Subject> element in the <xacml-context:Request> element.  
2274 Each <Subject> element SHALL contain the XML attribute SubjectCategory. This attribute is  
2275 called the *subject category attribute*.

2276 A *categorized subject* is a **subject** that is identified by a particular *subject category attribute*.

2277 A **subject attribute** is an **attribute** of a particular **subject**, i.e. contained within a <Subject>  
2278 element.

2279 A named **subject attribute** is a named **attribute** for a **subject**.

2280 A named categorized **subject attribute** is a named **subject attribute** for a particular **categorized**  
2281 **subject**.

2282 The **SubjectAttributeDesignatorType** complex type extends the **AttributeDesignatorType** with a  
2283 **SubjectCategory** attribute. The **SubjectAttributeDesignatorType** extends the match  
2284 semantics of the **AttributeDesignatorType** such that it narrows the **attribute** search space to the  
2285 specific **categorized subject** such that the value of this element's **SubjectCategory** attribute  
2286 matches, by URI-equality, the value of the <Request> element's **subject category attribute**.

2287 If there are multiple **subjects** with the same **SubjectCategory** xml attribute, then they SHALL be  
2288 treated as if they were one **categorized subject**.

2289 Elements and extensions of the **SubjectAttributeDesignatorType** complex type determine the  
2290 presence of select **attribute values** associated with **named categorized subject attributes**.  
2291 Elements and extensions of the **SubjectAttributeDesignatorType** SHALL NOT alter the match  
2292 semantics of **named categorized subject attributes**, but MAY narrow the search space.

```

2293 <xs:complexType name="SubjectAttributeDesignatorType">
2294   <xs:complexContent>
2295     <xs:extension base="xacml:AttributeDesignatorType">
2296       <xs:attribute name="SubjectCategory"
2297         type="xs:anyURI"
2298         use="optional"
2299         default="
2300 urn:oasis:names:tc:xacml:1.0:subject-category:access-
2301 subject"/>
2302     </xs:extension>
2303   </xs:complexContent>
2304 </xs:complexType>

```

2305 The <SubjectAttributeDesignatorType> complex type contains the following attribute in  
2306 addition to the attributes of the **AttributeDesignatorType** complex type:

2307 **SubjectCategory** [Optional]

2308 This attribute SHALL specify the **categorized subject** from which to match **named subject**  
2309 **attributes**. If **SubjectCategory** is not present, then its default value of  
2310 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be  
2311 used.

## 2312 5.29. Element <ResourceAttributeDesignator>

2313 The <ResourceAttributeDesignator> element retrieves a **bag** of values for a **named**  
2314 **resource attribute**. A **resource attribute** is an **attribute** contained within the <Resource>  
2315 element of the <xacml-context:Request> element. A **named resource attribute** is a **named**  
2316 **attribute** that matches a **resource attribute**. A **named resource attribute** SHALL be considered  
2317 **present** if there is at least one **resource attribute** that matches the criteria set out below. A  
2318 **resource attribute** value is an **attribute** value that is contained within a **resource attribute**.

2319 The <ResourceAttributeDesignator> element SHALL return a **bag** containing all the  
2320 **resource attribute** values that are matched by the **named resource attribute**. The  
2321 **MustBePresent** attribute governs whether this element returns an empty **bag** or "Indeterminate"  
2322 in the case that the **named resource attribute** is absent. If the **named resource attribute** is not  
2323 present and the **MustBePresent** attribute is "False" (its default value), then this element SHALL  
2324 evaluate to an empty **bag**. If the **named resource attribute** is not present and the  
2325 **MustBePresent** attribute is "True", then this element SHALL evaluate to "Indeterminate".  
2326 Regardless of the **MustBePresent** attribute, if it cannot be determined whether the **named**

2327 *resource attribute* is present or not in the **request context**, or the value of the *named resource*  
2328 **attribute** is unavailable, then the expression SHALL evaluate to “Indeterminate”.

2329 A *named resource attribute* SHALL match a **resource attribute** as per the match semantics  
2330 specified in the **AttributeDesignatorType** complex type [Section 5.27]

2331 The <ResourceAttributeDesignator> MAY appear in the <ResourceMatch> element and  
2332 MAY be passed to the <Apply> element as an argument.

```
2333 <xs:element name="ResourceAttributeDesignator"  
2334           type="xacml:AttributeDesignatorType" />
```

2335 The <ResourceAttributeDesignator> element is of the **AttributeDesignatorType**  
2336 complex type.

### 2337 **5.30. Element <ActionAttributeDesignator>**

2338 The <ActionAttributeDesignator> element retrieves a **bag** of values for a *named action*  
2339 **attribute**. An **action attribute** is an **attribute** contained within the <Action> element of the  
2340 <xacml-context:Request> element. A *named action attribute* has specific criteria (described  
2341 below) with which to match an **action attribute**. A *named action attribute* SHALL be considered  
2342 *present*, if there is at least one **action attribute** that matches the criteria. An **action attribute value**  
2343 is an **attribute value** that is contained within an **action attribute**.

2344 The <ActionAttributeDesignator> element SHALL return a **bag** of all the **action attribute**  
2345 values that are matched by the *named action attribute*. The `MustBePresent` attribute governs  
2346 whether this element returns an empty **bag** or “Indeterminate” in the case that the *named action*  
2347 **attribute** is absent. If the *named action attribute* is not present and the `MustBePresent` attribute  
2348 is “False” (its default value), then this element SHALL evaluate to an empty **bag**. If the *named*  
2349 **action attribute** is not present and the `MustBePresent` attribute is “True”, then this element  
2350 SHALL evaluate to “Indeterminate”. Regardless of the `MustBePresent` attribute, if it cannot be  
2351 determined whether the *named action attribute* is present or not present in the request **context**, or  
2352 the value of the *named action attribute* is unavailable, then the expression SHALL evaluate to  
2353 “Indeterminate”.

2354 A *named action attribute* SHALL match an **action attribute** as per the match semantics specified  
2355 in the **AttributeDesignatorType** complex type [Section 5.27].

2356 The <ActionAttributeDesignator> MAY appear in the <ActionMatch> element and MAY  
2357 be passed to the <Apply> element as an argument.

```
2358 <xs:element name="ActionAttributeDesignator"  
2359           type="xacml:AttributeDesignatorType" />
```

2360 The <ActionAttributeDesignator> element is of the **AttributeDesignatorType** complex  
2361 type.

### 2362 **5.31. Element <EnvironmentAttributeDesignator>**

2363 The <EnvironmentAttributeDesignator> element retrieves a **bag** of values for a *named*  
2364 **environment attribute**. An **environment attribute** is an **attribute** contained within the  
2365 <Environment> element of the <xacml-context:Request> element. A *named environment*  
2366 **attribute** has specific criteria (described below) with which to match an **environment attribute**. A  
2367 *named environment attribute* SHALL be considered *present*, if there is at least one **environment**  
2368 **attribute** that matches the criteria. An **environment attribute value** is an **attribute value** that is  
2369 contained within an **environment attribute**.

2370 The <EnvironmentAttributeDesignator> element SHALL evaluate to a **bag** of all the  
2371 **environment attribute** values that are matched by the *named environment attribute*. The  
2372 MustBePresent attribute governs whether this element returns an empty **bag** or “Indeterminate”  
2373 in the case that the *named environment attribute* is absent. If the *named environment attribute*  
2374 is not present and the MustBePresent attribute is “False” (its default value), then this element  
2375 SHALL evaluate to an empty **bag**. If the *named environment attribute* is not present and the  
2376 MustBePresent attribute is “True”, then this element SHALL evaluate to “Indeterminate”.  
2377 Regardless of the MustBePresent attribute, if it cannot be determined whether the *named*  
2378 **environment attribute** is present or not present in the request **context**, or the value of the *named*  
2379 **environment attribute** is unavailable, then the expression SHALL evaluate to “Indeterminate”.

2380 A *named environment attribute* SHALL match an **environment attribute** as per the match  
2381 semantics specified in the **AttributeDesignatorType** complex type [Section 5.27].

2382 The <EnvironmentAttributeDesignator> MAY be passed to the <Apply> element as an  
2383 argument.

```
2384 <xs:element name="EnvironmentAttributeDesignator"  
2385           type="xacml:AttributeDesignatorType" />
```

2386 The <EnvironmentAttributeDesignator> element is of the **AttributeDesignatorType**  
2387 complex type.

## 2388 5.32. Element <AttributeSelector>

2389 The AttributeSelector element's RequestContextPath XML attribute SHALL contain a  
2390 legal XPath expression whose context node is the <xacml-context:Request> element. The  
2391 AttributeSelector element SHALL evaluate to a **bag** of values whose data-type is specified by  
2392 the element's DataType attribute. If the DataType specified in the AttributeSelector is a  
2393 primitive data type defined in [XF] or [XS], then the value returned by the XPath expression SHALL  
2394 be converted to the DataType specified in the AttributeSelector using the constructor  
2395 function below [XF Section 4] that corresponds to the DataType. If an error results from using the  
2396 constructor function, then the value of the AttributeSelector SHALL be "Indeterminate".

```
2397  
2398     xs:string()  
2399     xs:boolean()  
2400     xs:integer()  
2401     xs:double()  
2402     xs:dateTime()  
2403     xs:date()  
2404     xs:time()  
2405     xs:hexBinary()  
2406     xs:base64Binary()  
2407     xs:anyURI()  
2408     xf:yearMonthDuration()  
2409     xf:dayTimeDuration()
```

2410  
2411 If the DataType specified in the AttributeSelector is not one of the preceding primitive  
2412 DataTypes, then the AttributeSelector SHALL return a bag of instances of the specified  
2413 DataType. If there are errors encountered in converting the values returned by the XPath  
2414 expression to the specified DataType, then the result of the AttributeSelector SHALL be  
2415 "Indeterminate".

2416  
2417 Each selected node by the specified XPath expression MUST be either a text node, an attribute  
2418 node, a processing instruction node, or a comment node. The string representation of the value of  
2419 each selected node MUST be converted to an **attribute** value of the specified data type, and the

2420 result of the `AttributeSelector` is the **bag** of the **attribute** values generated from all the  
2421 selected nodes.  
2422  
2423 If the selected node is different from the node types listed above (a text node, an attribute node, a  
2424 processing instruction node, or a comment node), then the result of that **policy** SHALL be  
2425 "Indeterminate" with a `StatusCode` value of  
2426 "urn:oasis:names:tc:xacml:1.0:status:syntax-error".  
2427 Support for the `<AttributeSelector>` element is OPTIONAL.

2428

```
2429 <xs:element name="AttributeSelector" type="xacml:AttributeSelectorType"/>  
2430 <xs:complexType name="AttributeSelectorType">  
2431   <xs:attribute name="RequestContextPath" type="xs:string" use="required"/>  
2432   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2433   <xs:attribute name="MustBePresent" type="xs:boolean" use="optional"  
2434   default="false"/>  
2435 </xs:complexType>
```

2436 The `<AttributeSelector>` element is of **AttributeSelectorType** complex type.

2437 The `<AttributeSelector>` element has the following attributes:

2438 `RequestContextPath` [Required]

2439 An XPath expression whose context node is the `<xacml-context:Request>` element.  
2440 There SHALL be no restriction on the XPath syntax.

2441 `DataType` [Required]

2442 The bag of values returned by the `AttributeSelector` SHALL be of this data type.

2443 `MustBePresent` [Optional]

2444 Whether or not the designated **attribute** must be present in the **context**. If the XPath expression  
2445 selects no node, and the `MustBePresent` attribute is TRUE, then the result SHALL be  
2446 "Indeterminate" and the status code SHALL be  
2447 "urn:oasis:names:tc:xacml:1.0:status:missing-attribute". If the XPath  
2448 expression selects no node, and the `MustBePresent` attribute is missing or FALSE, then the  
2449 result SHALL be an empty **bag**. If the XPath expression selects at least one node and the  
2450 selected node(s) could be successfully converted to a **bag** of values of the specified data-type,  
2451 then the result SHALL be the **bag**, regardless of the value of the `MustBePresent` attribute. If  
2452 the XPath expression selects at least one node, but there is an error in converting one or more  
2453 of the nodes to values of the specified data-type, then the result SHALL be "Indeterminate" and  
2454 the status code SHALL be "urn:oasis:names:tc:xacml:1.0:status:processing-  
2455 error", regardless of the value of the `MustBePresent` attribute.

### 2456 **5.33. Element `<AttributeValue>`**

2457 The `<AttributeValue>` element SHALL contain a literal **attribute** value.

```
2458 <xs:element name="AttributeValue" type="xacml:AttributeValueType"/>  
2459 <xs:complexType name="AttributeValueType" mixed="true">  
2460   <xs:sequence>  
2461     <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2462     maxOccurs="unbounded"/>  
2463   </xs:sequence>  
2464   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>  
2465   <xs:anyAttribute namespace="##any" processContents="lax"/>  
2466 </xs:complexType>
```

2467 The <AttributeValue> element is of **AttributeValueType** complex type.

2468 The <AttributeValue> element has the following attributes:

2469 DataType [Required]

2470 The data-type of the *attribute* value.

### 2471 **5.34. Element <Obligations>**

2472 The <Obligations> element SHALL contain a set of <Obligation> elements.

2473 Support for the <Obligations> element is OPTIONAL.

```
2474 <xs:element name="Obligations" type="xacml:ObligationsType"/>
2475 <xs:complexType name="ObligationsType">
2476   <xs:sequence>
2477     <xs:element ref="xacml:Obligation" maxOccurs="unbounded"/>
2478   </xs:sequence>
2479 </xs:complexType>
```

2480 The <Obligations> element is of **ObligationsType** complexType.

2481 The <Obligations> element contains the following element:

2482 <Obligation> [One to Many]

2483 A sequence of *obligations*

### 2484 **5.35. Element <Obligation>**

2485 The <Obligation> element SHALL contain an identifier for the *obligation* and a set of *attributes* that form arguments of the action defined by the *obligation*. The FulfillOn attribute SHALL indicate the *effect* for which this *obligation* applies.

```
2488 <xs:element name="Obligation" type="xacml:ObligationType"/>
2489 <xs:complexType name="ObligationType">
2490   <xs:sequence>
2491     <xs:element ref="xacml:AttributeAssignment" maxOccurs="unbounded"/>
2492   </xs:sequence>
2493   <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
2494   <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
2495 </xs:complexType>
```

2496 The <Obligation> element is of **ObligationType** complexType. See Section 7.11 for a description of how the set of *obligations* to be returned by the PDP is determined.

2498 The <Obligation> element contains the following elements and attributes:

2499 ObligationId [Required]

2500 *Obligation* identifier. The value of the *obligation* identifier SHALL be interpreted by the *PEP*.

2502 FulfillOn [Required]

2503 The *effect* for which this *obligation* applies.

2504 <AttributeAssignment> [One To Many]

2505 *Obligation* arguments assignment. The values of the *obligation* arguments SHALL be interpreted by the *PEP*.

2507

## 5.36. Element <AttributeAssignment>

2508 The <AttributeAssignment> element SHALL contain an `AttributeId` and the corresponding  
2509 **attribute** value. The `AttributeId` is part of **attribute** meta-data, and is used when the **attribute**  
2510 cannot be referenced by its location in the `<xacml-context:Request>`. This situation may arise  
2511 in an <Obligation> element if the **obligation** includes parameters. The  
2512 <AttributeAssignment> element MAY be used in any way consistent with the schema syntax,  
2513 which is a sequence of “any”. The value specified SHALL be understood by the PEP, but it is not  
2514 further specified by XACML. See section 7,11 “Obligations”.

```
2515 <xs:element name="AttributeAssignment"  
2516 type="xacml:AttributeAssignmentType"/>  
2517 <xs:complexType name="AttributeAssignmentType" mixed="true">  
2518 <xs:complexContent>  
2519 <xs:extension base="xacml:AttributeValueType">  
2520 <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>  
2521 </xs:extension>  
2522 </xs:complexContent>  
2523 </xs:complexType>
```

2524 The <AttributeAssignment> element is of **AttributeAssignmentType** complex type.

2525 The <AttributeAssignment> element contains the following attributes:

2526 `AttributeId` [Required]

2527 The **attribute** Identifier

---

## 2528 6. Context syntax (normative with the exception of 2529 the schema fragments)

### 2530 6.1. Element <Request>

2531 The <Request> element is a top-level element in the XACML **context** schema. The <Request>  
2532 element is an abstraction layer used by the **policy** language. Any proprietary system using the  
2533 XACML specification MUST transform its **decision request** into the form of an XACML **context**  
2534 <Request>.

2535 The <Request> element contains <Subject>, <Resource>, <Action> and <Environment>  
2536 elements. There may be multiple <Subject> elements. Each child element contains a sequence  
2537 of <xacml-context:Attribute> elements associated with the **subject**, **resource**, **action** and  
2538 **environment** respectively.

```
2539 <xs:element name="Request" type="xacml-context:RequestType"/>  
2540 <xs:complexType name="RequestType">  
2541 <xs:sequence>  
2542 <xs:element ref="xacml-context:Subject" maxOccurs="unbounded"/>  
2543 <xs:element ref="xacml-context:Resource"/>  
2544 <xs:element ref="xacml-context:Action"/>  
2545 <xs:element ref="xacml-context:Environment" minOccurs="0"/>  
2546 </xs:sequence>  
2547 </xs:complexType>
```

2548 The <Request> element is of **RequestType** complex type.

2549 The <Request> element contains the following elements:

- 2550 <Subject> [One to Many]
- 2551 Specifies information about a **subject** of the request **context** by listing a sequence of  
 2552 <Attribute> elements associated with the **subject**. One or more <Subject> elements  
 2553 are allowed. A **subject** is an entity associated with the **access** request. One **subject**  
 2554 might represent the human user that initiated the application from which the request was  
 2555 issued. Another **subject** might represent the application's executable code that created the  
 2556 request. Another **subject** might represent the machine on which the application was  
 2557 executing. Another **subject** might represent the entity that is to be the recipient of the  
 2558 **resource**. Attributes of each of these entities MUST be enclosed in a separate  
 2559 <Subject> element.
- 2560 <Resource> [Required]
- 2561 Specifies information about the resource for which access is being requested by listing a  
 2562 sequence of <Attribute> elements associated with the resource. It MAY include a  
 2563 <ResourceContent> element.
- 2564 <Action> [Required]
- 2565 Specifies the requested **action** to be performed on the **resource** by listing a set of  
 2566 <Attribute> elements associated with the **action**.
- 2567 <Environment> [Optional]
- 2568 Contains a set of <Attribute> elements of the **environment**. These <Attribute>  
 2569 elements MAY form a part of **policy** evaluation.

## 2570 6.2. Element <Subject>

2571 The <Subject> element specifies a **subject** by listing a sequence of <Attribute> elements  
 2572 associated with the **subject**.

```

2573 <xs:element name="Subject" type="xacml-context:SubjectType" />
2574 <xs:complexType name="SubjectType">
2575   <xs:sequence>
2576     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2577 maxOccurs="unbounded" />
2578   </xs:sequence>
2579   <xs:attribute name="SubjectCategory" type="xs:anyURI" use="optional"
2580 default="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />
2581 </xs:complexType>

```

2582 The <Subject> element is of **SubjectType** complex type.

2583 The <Subject> element contains the following elements:

2584 SubjectCategory [Optional]

2585 This attribute indicates the role that the parent <Subject> played in the formation of the  
 2586 access request. If this attribute is not present in a given <Subject> element, then the  
 2587 default value of "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" SHALL be  
 2588 used, indicating that the parent <Subject> element represents the entity ultimately  
 2589 responsible for initiating the **access** request.

2590 If more than one <Subject> element contains a "urn:oasis:names:tc:xacml:1.0:subject-  
 2591 category" attribute with the same value, then the PDP SHALL treat the contents of those  
 2592 elements as if they were contained in the same <Subject> element.

2593 <Attribute> [Any Number]

- 2594 A sequence of attributes that apply to the subject.
- 2595 Typically, a <Subject> element will contain an <Attribute> with an AttributeId of  
2596 "urn:oasis:names:tc:xacml:1.0:subject:subject-id", containing the identity of the **subject**.
- 2597 A <Subject> element MAY contain additional <Attribute> elements.

### 2598 6.3. Element <Resource>

- 2599 The <Resource> element specifies information about the **resource** to which **access** is requested,  
2600 by listing a sequence of <Attribute> elements associated with the **resource**. It MAY include the  
2601 **resource** content.

```
2602 <xs:element name="Resource" type="xacml-context:ResourceType" />
2603 <xs:complexType name="ResourceType">
2604   <xs:sequence>
2605     <xs:element ref="xacml-context:ResourceContent" minOccurs="0" />
2606     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2607 maxOccurs="unbounded" />
2608   </xs:sequence>
2609 </xs:complexType>
```

- 2610 The <Resource> element is of **ResourceType** complex type.

- 2611 The <Resource> element contains the following elements:

2612 <ResourceContent> [Optional]

- 2613 The **resource** content.

2614 <Attribute> [Any Number]

- 2615 A sequence of **resource attributes**. The <Resource> element MUST contain one and  
2616 only one <Attribute> with an AttributeId of  
2617 "urn:oasis:names:tc:xacml:1.0:resource:resource-id". This **attribute**  
2618 specifies the identity of the **resource** to which **access** is requested.

- 2619 A <Resource> element MAY contain additional <Attribute> elements.

### 2620 6.4. Element <ResourceContent>

- 2621 The <ResourceContent> element is a notional placeholder for the **resource** content. If an  
2622 XACML **policy** references the contents of the **resource**, then the <ResourceContent> element  
2623 SHALL be used as the reference point.

```
2624 <xs:complexType name="ResourceContentType" mixed="true">
2625   <xs:sequence>
2626     <xs:any namespace="##any" processContents="lax" minOccurs="0"
2627 maxOccurs="unbounded" />
2628   </xs:sequence>
2629   <xs:anyAttribute namespace="##any" processContents="lax" />
2630 </xs:complexType>
```

- 2631 The <ResourceContent> element is of **ResourceContentType** complex type.

- 2632 The <ResourceContent> element allows arbitrary elements and attributes.

2633

## 6.5. Element <Action>

2634 The <Action> element specifies the requested **action** on the **resource**, by listing a set of  
2635 <Attribute> elements associated with the **action**.

```
2636 <xs:element name="Action" type="xacml-context:ActionType"/>
2637 <xs:complexType name="ActionType">
2638   <xs:sequence>
2639     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2640     maxOccurs="unbounded"/>
2641   </xs:sequence>
2642 </xs:complexType>
```

2643 The <Action> element is of **ActionType** complex type.

2644 The <Action> element contains the following elements:

2645 <Attribute> [Any Number]

2646 List of **attributes** of the **action** to be performed on the **resource**.

## 2647 6.6. Element <Environment>

2648 The <Environment> element contains a set of **attributes** of the **environment**. These **attributes**  
2649 MAY form part of the **policy** evaluation.

2650

```
2651 <xs:element name="Environment" type="xacml-context:EnvironmentType"/>
2652 <xs:complexType name="EnvironmentType">
2653   <xs:sequence>
2654     <xs:element ref="xacml-context:Attribute" minOccurs="0"
2655     maxOccurs="unbounded"/>
2656   </xs:sequence>
2657 </xs:complexType>
```

2658 The <Environment> element is of **EnvironmentType** complex type.

2659 The <Environment> element contains the following elements:

2660 <Attribute> [Any Number]

2661 A list of **environment attributes**. Environment **attributes** are **attributes** that are not  
2662 associated with either the **resource**, the **action** or any of the **subjects** of the **access**  
2663 request.

## 2664 6.7. Element <Attribute>

2665 The <Attribute> element is the central abstraction of the request **context**. It contains an  
2666 **attribute** value and **attribute** meta-data. The **attribute** meta-data comprises the **attribute**  
2667 identifier, the **attribute** issuer and the **attribute** issue instant. **Attribute** designators and **attribute**  
2668 selectors in the **policy** MAY refer to **attributes** by means of this meta-data.

```
2669 <xs:element name="Attribute" type="xacml-context:AttributeType"/>
2670 <xs:complexType name="AttributeType">
2671   <xs:sequence>
2672     <xs:element ref="xacml-context:AttributeValue"/>
2673   </xs:sequence>
2674   <xs:attribute name="AttributeId" type="xs:anyURI" use="required"/>
2675   <xs:attribute name="DataType" type="xs:anyURI" use="required"/>
2676   <xs:attribute name="Issuer" type="xs:string" use="optional"/>
```

2677 `<xs:attribute name="IssueInstant" type="xs:dateTime" use="optional" />`  
2678 `</xs:complexType>`

2679 The <Attribute> element is of **AttributeType** complex type.

2680 The <Attribute> element contains the following attributes and elements:

2681 **AttributeId** [Required]

2682 **Attribute** identifier. A number of identifiers are reserved by XACML to denote commonly  
2683 used **attributes**.

2684 **DataType** [Required]

2685 The data-type of the contents of the <AttributeValue> element. This SHALL be either  
2686 a primitive type defined by the XACML 1.0 specification or a type defined in a namespace  
2687 declared in the <xacml-context> element.

2688 **Issuer** [Optional]

2689 **Attribute** issuer. This attribute value MAY be an x500Name that binds to a public key, or it  
2690 may be some other identifier exchanged out-of-band by issuing and relying parties.

2691 **IssueInstant** [Optional]

2692 The date and time at which the **attribute** was issued.

2693

2694 <AttributeValue> [Required]

2695 Exactly one **attribute** value. The mandatory **attribute** value MAY have contents that are empty,  
2696 occur once, or occur multiple times.

## 2697 **6.8. Element <AttributeValue>**

2698 The <AttributeValue> element contains the value of an **attribute**.

```
2699 <xs:element name="AttributeValue" type="xacml-context:AttributeValueType" />  
2700 <xs:complexType name="AttributeValueType" mixed="true">  
2701 <xs:sequence>  
2702 <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2703 maxOccurs="unbounded" />  
2704 </xs:sequence>  
2705 <xs:anyAttribute namespace="##any" processContents="lax" />  
2706 </xs:complexType>
```

2707 The <AttributeValue> element is of **AttributeValueType** type.

2708 The data-type of the <AttributeValue> MAY be specified by using the **DataType** attribute of  
2709 the parent <Attribute> element.

## 2710 **6.9. Element <Response>**

2711 The <Response> element is a top-level element in the XACML **context** schema. The  
2712 <Response> element is an abstraction layer used by the **policy** language. Any proprietary  
2713 system using the XACML specification MUST transform an XACML **context** <Response> into the  
2714 form of its **authorization decision**.

2715 The <Response> element encapsulates the **authorization decision** produced by the **PDP**. It  
2716 includes a sequence of one or more results, with one <Result> element per requested **resource**.  
2717 Multiple results MAY be returned when the value of the "urn:oasis:xacml:1.0:resource:scope"  
2718 resource **attribute** in the request **context** is "Descendants" or "Children". Support for multiple  
2719 results is OPTIONAL.

```
2720 <xs:element name="Response" type="xacml-context:ResponseType" />  
2721 <xs:complexType name="ResponseType">  
2722 <xs:sequence>  
2723 <xs:element ref="xacml-context:Result" maxOccurs="unbounded" />  
2724 </xs:sequence>  
2725 </xs:complexType>
```

2726 The <Response> element is of **ResponseType** complex type.

2727 The <Response> element contains the following elements:

2728 <Result> [One to Many]

2729 An authorization decision result.

## 2730 6.10. Element <Result>

2731 The <Result> element represents an **authorization decision** result for the **resource** specified by  
2732 the ResourceId **attribute**. It MAY include a set of **obligations** that MUST be fulfilled by the **PEP**.  
2733 If the **PEP** does not understand an **obligation**, then it MUST act as if the **PDP** had denied **access**  
2734 to the requested **resource**.

```
2735  
2736 <xs:element name="Result" type="xacml-context:ResultType" />  
2737 <xs:complexType name="ResultType">  
2738 <xs:sequence>  
2739 <xs:element ref="xacml-context:Decision" />  
2740 <xs:element ref="xacml-context:Status" />  
2741 <xs:element ref="xacml:Obligations" minOccurs="0" />  
2742 </xs:sequence>  
2743 <xs:attribute name="ResourceId" type="xs:string" use="optional" />  
2744 </xs:complexType>
```

2745 The <Result> element is of **ResultType** complex type.

2746 The <Result> element contains the following attributes and elements:

2747 ResourceId [Optional]

2748 The identifier of the requested **resource**. If this attribute is omitted, then the **resource**  
2749 identity is specified by the "urn:oasis:names:tc:xacml:1.0:resource:resource-  
2750 id" **resource attribute** in the corresponding <Request> element.

2751 <Decision> [Required]

2752 The **authorization decision**: "Permit", "Deny", "Indeterminate" or "NotApplicable".

2753 <Status> [Required]

2754 Indicates whether errors occurred during evaluation of the **decision request**, and  
2755 optionally, information about those errors.

2756 <xacml:Obligations> [Optional]

2757 A list of **obligations** that MUST be fulfilled by the **PEP**. If the **PEP** does not understand an  
2758 **obligation**, then it MUST act as if the **PDP** had denied **access** to the requested **resource**.  
2759 See Section 7.11 for a description of how the set of **obligations** to be returned by the PDP  
2760 is determined.

## 2761 6.11. Element <Decision>

2762 The <Decision> element contains the result of **policy** evaluation.

```
2763 <xs:element name="Decision" type="xacml-context:DecisionType"/>  
2764 <xs:simpleType name="DecisionType">  
2765 <xs:restriction base="xs:string">  
2766 <xs:enumeration value="Permit"/>  
2767 <xs:enumeration value="Deny"/>  
2768 <xs:enumeration value="Indeterminate"/>  
2769 <xs:enumeration value="NotApplicable"/>  
2770 </xs:restriction>  
2771 </xs:simpleType>
```

2772 The <Decision> element is of **DecisionType** simple type.

2773 The values of the <Decision> element have the following meanings:

2774 "Permit": the requested **access** is permitted.

2775 "Deny": the requested **access** is denied.

2776 "Indeterminate": the **PDP** is unable to evaluate the requested **access**. Reasons for such  
2777 inability include: missing **attributes**, network errors while retrieving **policies**, division by  
2778 zero during **policy** evaluation, syntax errors in the **decision request** or in the **policy**, etc..

2779 "NotApplicable": the **PDP** does not have any **policy** that applies to this **decision request**.

## 2780 6.12. Element <Status>

2781 The <Status> element represents the status of the **authorization decision** result.

```
2782 <xs:element name="Status" type="xacml-context:StatusType"/>  
2783 <xs:complexType name="StatusType">  
2784 <xs:sequence>  
2785 <xs:element ref="xacml-context:StatusCode"/>  
2786 <xs:element ref="xacml-context:StatusMessage" minOccurs="0"/>  
2787 <xs:element ref="xacml-context:StatusDetail" minOccurs="0"/>  
2788 </xs:sequence>  
2789 </xs:complexType>
```

2790 The <Status> element is of **StatusType** complex type.

2791 The <Status> element contains the following elements:

2792 <StatusCode> [Required]

2793 Status code.

2794 <StatusMessage> [Optional]

2795 A status message describing the status code.

2796 <StatusDetail> [Optional]

2797 Additional status information.

2798

## 6.13. Element <StatusCode>

2799 The <StatusCode> element contains a major status code value and an optional sequence of  
2800 minor status codes.

```
2801 <xs:element name="StatusCode" type="xacml-context:StatusCodeType" />  
2802 <xs:complexType name="StatusCodeType">  
2803 <xs:sequence>  
2804 <xs:element ref="xacml-context:StatusCode" minOccurs="0" />  
2805 </xs:sequence>  
2806 <xs:attribute name="Value" type="xs:anyURI" use="required" />  
2807 </xs:complexType>
```

2808 The <StatusCode> element is of **StatusCodeType** complex type.

2809 The <StatusCode> element contains the following attributes and elements:

2810 Value [Required]

2811 See Section B.9 for a list of values.

2812 <StatusCode> [Any Number]

2813 Minor status code. This status code qualifies its parent status code.

## 2814 6.14. Element <StatusMessage>

2815 The <StatusMessage> element is a free-form description of the status code.

```
2816 <xs:element name="StatusMessage" type="xs:string" />
```

2817 The <StatusMessage> element is of **xs:string** type.

## 2818 6.15. Element <StatusDetail>

2819 The <StatusDetail> element qualifies the <Status> element with additional information.

```
2820 <xs:element name="StatusDetail" type="xacml-context:StatusDetailType" />  
2821 <xs:complexType name="StatusDetailType">  
2822 <xs:sequence>  
2823 <xs:any namespace="##any" processContents="lax" minOccurs="0"  
2824 maxOccurs="unbounded" />  
2825 </xs:sequence>  
2826 </xs:complexType>
```

2827 The <StatusDetail> element is of **StatusDetailType** complex type.

2828 The <StatusDetail> element allows arbitrary XML content.

2829 Inclusion of a <StatusDetail> element is optional. However, if a **PDP** returns one of the  
2830 following XACML-defined <StatusCode> values and includes a <StatusDetail> element, then  
2831 the following rules apply.

2832 urn:oasis:names:tc:xacml:1.0:status:ok

2833 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the "ok" status value.

2834 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

2835 A **PDP** MAY choose not to return any <StatusDetail> information or MAY choose to return a  
2836 <StatusDetail> element containing one or more <xacml-context:Attribute> elements. If  
2837 the **PDP** includes <AttributeValue> elements in the <Attribute> element, then this indicates

2838 the acceptable values for that **attribute**. If no <AttributeValue> elements are included, then  
2839 this indicates the names of **attributes** that the **PDP** failed to resolve during its evaluation. The list  
2840 of **attributes** may be partial or complete. There is no guarantee by the **PDP** that supplying the  
2841 missing values or **attributes** will be sufficient to satisfy the **policy**.

2842 urn:oasis:names:tc:xacml:1.0:status:syntax-error

2843 A **PDP** MUST NOT return a <StatusDetail> element in conjunction with the "syntax-error" status  
2844 value. A syntax error may represent either a problem with the **policy** being used or with the  
2845 request **context**. The **PDP** MAY return a <StatusMessage> describing the problem.

2846 urn:oasis:names:tc:xacml:1.0:status:processing-error

2847 A **PDP** MUST NOT return <StatusDetail> element in conjunction with the "processing-error"  
2848 status value. This status code indicates an internal problem in the **PDP**. For security reasons, the  
2849 **PDP** MAY choose to return no further information to the **PEP**. In the case of a divide-by-zero error  
2850 or other computational error, the **PDP** MAY return a <StatusMessage> describing the nature of  
2851 the error.

---

## 2852 7. Functional requirements (normative)

2853 This section specifies certain functional requirements that are not directly associated with the  
2854 production or consumption of a particular XACML element.

### 2855 7.1. Policy enforcement point

2856 This section describes the requirements for the **PEP**.

2857 An application functions in the role of the **PEP** if it guards access to a set of **resources** and asks  
2858 the **PDP** for an **authorization decision**. The **PEP** MUST abide by the **authorization decision** in  
2859 the following way:

2860 A **PEP** SHALL allow access to the **resource** only if a valid XACML response of "Permit" is returned  
2861 by the **PDP**. The **PEP** SHALL deny access to the **resource** in all other cases. An XACML  
2862 response of "Permit" SHALL be considered valid only if the **PEP** understands all of the **obligations**  
2863 contained in the response.

### 2864 7.2. Base policy

2865 A **PDP** SHALL represent one **policy** or **policy set**, called its **base policy**. This base **policy** MAY be  
2866 a <Policy> element containing a <Target> element that matches every possible **decision**  
2867 **request**, or (for instance) it MAY be a <Policy> element containing a <Target> element that  
2868 matches only a specific **subject**. In such cases, the base policy SHALL form the root-node of a  
2869 tree of policies connected by <PolicyIdReference> and <PolicySetIdReference>  
2870 elements to all the **rules** that may be applicable to any **decision request** that the **PDP** is capable  
2871 of evaluating.

2872 In the case of a **PDP** that retrieves **policies** according to the **decision request** that it is processing,  
2873 the base policy SHALL contain a <Policy> element containing a <Target> element that matches  
2874 every possible **decision request** and a PolicyCombiningAlgId attribute with the value "Only-  
2875 one-applicable". In other words, the **PDP** SHALL return an error if it retrieves policies that do not  
2876 form a single tree.

2877

### 7.3. Target evaluation

2878 The **target** value SHALL be "Match" if the **subject**, **resource** and **action** specified in the **target** all  
 2879 match values in the request **context**. The **target** value SHALL be "No-match" if one or more of the  
 2880 **subject**, **resource** and **action** specified in the **target** do not match values in the request **context**.  
 2881 The value of a <SubjectMatch>, <ResourceMatch> or <ActionMatch> element, in which a  
 2882 referenced **attribute** value cannot be obtained, depends on the value of the MustBePresent  
 2883 attribute of the <AttributeDesignator> or <AttributeSelector> element. If the  
 2884 MustBePresent attribute is "True", then the result of the <SubjectMatch>, <ResourceMatch>  
 2885 or <ActionMatch> element SHALL be "Indeterminate" in this case. If the MustBePresent  
 2886 attribute is "False" or missing, then the result of the <SubjectMatch>, <ResourceMatch> or  
 2887 <ActionMatch> element SHALL be "No-match".

2888

### 7.4. Condition evaluation

2889 The **condition** value SHALL be "True" if the <Condition> element is absent, or if it evaluates to  
 2890 "True" for the **attribute** values supplied in the request **context**. Its value is "False" if the  
 2891 <Condition> element evaluates to "False" for the **attribute** values supplied in the request  
 2892 **context**. If any **attribute** value referenced in the **condition** cannot be obtained, then the **condition**  
 2893 SHALL evaluate to "Indeterminate".

2894

### 7.5. Rule evaluation

2895 A **rule** has a value that can be calculated by evaluating its contents. **Rule** evaluation involves  
 2896 separate evaluation of the **rule's target** and **condition**. The **rule** truth table is shown in Table 1.

2897

2898

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

2899

Table 1 - Rule truth table

2900 If the **target** value is "No-match" or "Indeterminate" then the **rule** value SHALL be "NotApplicable"  
 2901 or "Indeterminate", respectively, regardless of the value of the **condition**. For these cases,  
 2902 therefore, the **condition** need not be evaluated in order to determine the **rule** value.

2903 If the **target** value is "Match" and the **condition** value is "True", then the **effect** specified in the **rule**  
 2904 SHALL determine the **rule** value.

2905

### 7.6. Policy evaluation

2906 The value of a **policy** SHALL be determined only by its contents, considered in relation to the  
 2907 contents of the **request context**. A **policy's** value SHALL be determined by evaluation of the  
 2908 **policy's target** and **rules**, according to the specified **rule-combining algorithm**.

2909 The **policy's target** SHALL be evaluated to determine the applicability of the **policy**. If the **target**  
 2910 evaluates to "Match", then the value of the **policy** SHALL be determined by evaluation of the  
 2911 **policy's rules**, according to the specified **rule-combining algorithm**. If the **target** evaluates to  
 2912 "No-match", then the value of the **policy** SHALL be "NotApplicable". If the **target** evaluates to  
 2913 "Indeterminate", then the value of the **policy** SHALL be "Indeterminate".

2914 The **policy** truth table is shown in Table 2.

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

2915

**Table 2 - Policy truth table**

2916 A **rules** value of "At least one rule value is its Effect" SHALL be used if the <Rule> element is  
 2917 absent, or if one or more of the **rules** contained in the **policy** is applicable to the **decision request**  
 2918 (i.e., returns a value of "Effect"; see Section 7.5). A **rules** value of "All rule values are  
 2919 'NotApplicable'" SHALL be used if no **rule** contained in the **policy** is applicable to the request and if  
 2920 no **rule** contained in the **policy** returns a value of "Indeterminate". If no **rule** contained in the  
 2921 **policy** is applicable to the request but one or more **rule** returns a value of "Indeterminate", then  
 2922 **rules** value SHALL evaluate to "At least one rule value is 'Indeterminate'".

2923 If the **target** value is "No-match" or "Indeterminate" then the **policy** value SHALL be  
 2924 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **rules**. For these  
 2925 cases, therefore, the **rules** need not be evaluated in order to determine the **policy** value.

2926 If the **target** value is "Match" and the **rules** value is "At least one rule value is its Effect" or "At least  
 2927 one rule value is 'Indeterminate'", then the **rule-combining algorithm** specified in the **policy**  
 2928 SHALL determine the **policy** value.

## 2929 7.7. Policy Set evaluation

2930 The value of a **policy set** SHALL be determined by its contents, considered in relation to the  
 2931 contents of the **request context**. A **policy set's** value SHALL be determined by evaluation of the  
 2932 **policy set's target**, **policies** and **policy sets**, according to the specified **policy-combining**  
 2933 **algorithm**.

2934 The **policy set's target** SHALL be evaluated to determine the applicability of the **policy set**. If the  
 2935 **target** evaluates to "Match" then the value of the **policy set** SHALL be determined by evaluation of  
 2936 the **policy set's policies** and **policy sets**, according to the specified **policy-combining algorithm**.  
 2937 If the **target** evaluates to "No-match", then the value of the **policy set** shall be "NotApplicable". If  
 2938 the **target** evaluates to "Indeterminate", then the value of the **policy set** SHALL be "Indeterminate".

2939 The **policy set** truth table is shown in Table 3.

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

Table 3 – Policy set truth table

2940

2941 A **policies** value of "At least one policy value is its **Decision**" SHALL be used if there are no  
 2942 contained or referenced **policies** or **policy sets**, or if one or more of the **policies** or **policy sets**  
 2943 contained in or referenced by the **policy set** is applicable to the **decision request** (i.e., returns a  
 2944 value determined by its **rule-combining algorithm**; see Section 7.6). A **policies** value of "All  
 2945 policy values are 'NotApplicable'" SHALL be used if no **policy** or **policy set** contained in or  
 2946 referenced by the **policy set** is applicable to the request and if no **policy** or **policy set** contained in  
 2947 or referenced by the **policy set** returns a value of "Indeterminate". If no **policy** or **policy set**  
 2948 contained in or referenced by the **policy set** is applicable to the request but one or more **policy** or  
 2949 **policy set** returns a value of "Indeterminate", then **policies** SHALL evaluate to "At least one policy  
 2950 value is 'Indeterminate'".

2951 If the **target** value is "No-match" or "Indeterminate" then the **policy set** value SHALL be  
 2952 "NotApplicable" or "Indeterminate", respectively, regardless of the value of the **policies**. For these  
 2953 cases, therefore, the **policies** need not be evaluated in order to determine the **policy set** value.

2954 If the **target** value is "Match" and the **policies** value is "At least one policy value is its **Decision**" or  
 2955 "At least one policy value is 'Indeterminate'", then the **policy-combining algorithm** specified in the  
 2956 **policy set** SHALL determine the **policy set** value.

## 7.8. Hierarchical resources

2957

2958 It is often the case that a **resource** is organized as a hierarchy (e.g. file system, XML document).  
 2959 Some access requesters may request **access** to an entire subtree of a **resource** specified by a  
 2960 node. XACML allows the **PEP** (or **context handler**) to specify whether the **decision request** is  
 2961 just for a single **resource** or for a subtree below the specified **resource**. The latter is equivalent to  
 2962 repeating a single request for each node in the entire subtree. When a request **context** contains a  
 2963 resource attribute of type

2964 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2965 with a value of "Immediate", or if it does not contain that **attribute**, then the **decision request**  
 2966 SHALL be interpreted to apply to just the single **resource** specified by the  
 2967 "urn:oasis:names:tc:xacml:1.0:resource:resource-id" **attribute**.

2968 When the

2969 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2970 **attribute** has the value "Children", the **decision request** SHALL be interpreted to apply to the  
2971 specified **resource** and its immediate children **resources**.

2972 When the  
2973 "urn:oasis:names:tc:xacml:1.0:resource:scope"

2974 **attribute** has the value "Descendants", the **decision request** SHALL be interpreted to apply to  
2975 both the specified **resource** and all its descendant **resources**.

2976 In the case of "Children" and "Descendants", the **authorization decision** MAY include multiple  
2977 results for the multiple sub-nodes in the **resource** sub-tree.

2978 An XACML **authorization response** MAY contain multiple <Result> elements.

2979 Note that the method by which the **PDP** discovers whether the **resource** is hierarchically organized  
2980 or not is outside the scope of XACML.

2981 In the case where a child or descendant **resource** cannot be accessed, the <Result> element  
2982 associated with the parent element SHALL contain a <StatusCode> Value of  
2983 "urn:oasis:names:tc:xacml:1.0:status:processing-error".

## 2984 **7.9. Attributes**

2985 **Attributes** are specified in the request **context**, regardless of whether or not they appeared in the  
2986 original **decision request**, and are referred to in the **policy** by **subject**, **resource**, **action** and  
2987 **environment attribute** designators and **attribute** selectors. A *named attribute* is the term used for  
2988 the criteria that the specific **subject**, **resource**, **action** and **environment attribute** designators and  
2989 selectors use to refer to **attributes** in the **subject**, **resource**, **action** and **environment** elements of  
2990 the request **context**, respectively.

### 2991 **7.9.1. Attribute Matching**

2992 A *named attribute* has specific criteria with which to match **attributes** in the **context**. An **attribute**  
2993 specifies `AttributeId`, `DataType` and `Issuer` attributes, and each *named attribute* also  
2994 specifies `AttributeId`, `DataType` and optional `Issuer` attributes. A *named attribute* SHALL  
2995 match an **attribute** if the values of their respective `AttributeId`, `DataType` and optional `Issuer`  
2996 attributes match within their particular element, e.g. **subject**, **resource**, **action** or **environment**, of  
2997 the **context**. The `AttributeId` of the named **attribute** MUST match, by URI equality, the  
2998 `AttributeId` of the context **attribute**. The `DataType` of the named **attribute** MUST match, by  
2999 URI equality, the `DataType` of the same context **attribute**. If `Issuer` is supplied in the named  
3000 **attribute**, then it MUST match, by string equality, the `Issuer` of the same context **attribute**. If  
3001 `Issuer` is not supplied in the *named attribute*, then the matching of the context **attribute** to the  
3002 *named attribute* SHALL be governed by `AttributeId` and `DataType` alone, regardless of the  
3003 presence, absence, or actual value of `Issuer`. In the case of an **attribute** selector, the matching  
3004 of the **attribute** to the *named attribute* SHALL be governed by the XPath expression and  
3005 `DataType`.

### 3006 **7.9.2. Attribute Retrieval**

3007 The **PDP** SHALL request the values of **attributes** in the request **context** from the **context handler**.  
3008 The **PDP** SHALL reference the **attributes** as if they were in a physical request **context** document,  
3009 but the **context handler** is responsible for obtaining and supplying the requested values. The  
3010 **context handler** SHALL return the values of **attributes** that match the **attribute** designator or  
3011 **attribute** selector and form them into a **bag** of values with the specified data-type. If no **attributes**

3012 from the request **context** match, then the **attribute** SHALL be considered missing. If the **attribute**  
3013 is missing, then `MustBePresent` governs whether the **attribute** designator or **attribute** selector  
3014 returns an empty **bag** or an "Indeterminate" result. If `MustBePresent` is "False" (default value),  
3015 then a missing **attribute** SHALL result in an empty **bag**. If `MustBePresent` is "True", then a  
3016 missing **attribute** SHALL result in "Indeterminate". This "Indeterminate" result SHALL be handled  
3017 in accordance with the specification of the encompassing expressions, **rules**, **policies** and **policy**  
3018 **sets**. If the result is "Indeterminate", then the `AttributeId`, `DataTypes` and `Issuer` of the  
3019 **attribute** MAY be listed in the **authorization decision** as described in Section 7.10. However, a  
3020 **PDP** MAY choose not to return such information for security reasons.

### 3021 7.9.3. Environment Attributes

3022 **Environment attributes** are listed in Section B.8. If a value for one of these **attributes** is supplied  
3023 in the **decision request**, then the **context handler** SHALL use that value. Otherwise, the **context**  
3024 **handler** SHALL supply a value. For the date and time **attributes**, the supplied value SHALL have  
3025 the semantics of "date and time that apply to the **decision request**".

## 3026 7.10. Authorization decision

3027 Given a valid XACML **policy** or **policy set**, a compliant XACML **PDP** MUST evaluate the **policy** as  
3028 specified in Sections 5, 0 and 4.2. The **PDP** MUST return a response **context**, with one  
3029 `<Decision>` element of value "Permit", "Deny", "Indeterminate" or "NotApplicable".

3030 If the **PDP** cannot make a decision, then an "Indeterminate" `<Decision>` element contents SHALL  
3031 be returned. The **PDP** MAY return a `<Decision>` element contents of "Indeterminate" with a  
3032 status code of:

3033 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3034 signifying that more information is needed. In this case, the `<Status>` element MAY list the  
3035 names and data-types of any **attributes** of the **subjects**, **resource**, **action**, or **environment** that  
3036 are needed by the **PDP** to refine its decision. A **PEP** MAY resubmit a refined request **context** in  
3037 response to a `<Decision>` element contents of "Indeterminate" with a status code of

3038 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3039 by adding **attribute** values for the **attribute** names that were listed in the previous response. When  
3040 the **PDP** returns a `<Decision>` element contents of "Indeterminate", with a status code of

3041 "urn:oasis:names:tc:xacml:1.0:missing-attribute",

3042 it MUST NOT list the names and data-types of any **attribute** of the **subject**, **resource**, **action**, or  
3043 **environment** for which values were supplied in the original request. Note, this requirement forces  
3044 the **PDP** to eventually return an **authorization decision** of "Permit", "Deny" or "Indeterminate" with  
3045 some other status code, in response to successively-refined requests.

## 3046 7.11. Obligations

3047 A **policy** or **policy set** may contain one or more **obligations**. When such a **policy** or **policy set** is  
3048 evaluated, an **obligation** SHALL be passed up to the next level of evaluation (the enclosing or  
3049 referencing **policy set** or **authorization decision**) only if the **effect** of the **policy** or **policy set**  
3050 being evaluated matches the value of the `xacml:FulfillOn` attribute of the **obligation**.  
3051

3052 As a consequence of this procedure, no **obligations** SHALL be returned to the **PEP** if the **policies**  
3053 or **policy sets** from which they are drawn are not evaluated, or if their evaluated result is

3054 "Indeterminate" or "NotApplicable", or if the **decision** resulting from evaluating the **policy** or **policy set** does not match the **decision** resulting from evaluating an enclosing **policy set**.  
3055  
3056  
3057 If the **PDP's** evaluation is viewed as a tree of **policy sets** and **policies**, each of which returns  
3058 "Permit" or "Deny", then the set of **obligations** returned by the **PDP** to the **PEP** will include only the  
3059 **obligations** associated with those paths where the **effect** at each level of evaluation is the same as  
3060 the **effect** being returned by the **PDP**.  
3061 A **PEP** that receives a valid XACML response of "Permit" with **obligations** SHALL be responsible  
3062 for fulfilling *all* of those **obligations**. A **PEP** that receives an XACML response of "Deny" with  
3063 **obligations** SHALL be responsible for fulfilling all of the **obligations** that it *understands*.

## 3064 7.12. Unsupported functionality

3065 If the **PDP** attempts to evaluate a **policy set** or **policy** that contains an optional element type or  
3066 feature that the **PDP** does not support, then the **PDP** SHALL return a <Decision> value of  
3067 "Indeterminate". If a <StatusCode> element is also returned, then its value SHALL be  
3068 "urn:oasis:names:tc:xacml:1.0:status:syntax-error" in the case of an unsupported element type, and  
3069 "urn:oasis:names:tc:xacml:1.0:status:processing-error" in the case of an unsupported feature.

## 3070 7.13. Syntax and type errors

3071 If a **policy** that contains invalid syntax is evaluated by the XACML **PDP** at the time a **decision request**  
3072 is received, then the result of that **policy** SHALL be "Indeterminate" with a StatusCode  
3073 value of "urn:oasis:names:tc:xacml:1.0:status:syntax-error".

3074 If a **policy** that contains invalid static data-types is evaluated by the XACML **PDP** at the time a  
3075 **decision request** is received, then the result of that **policy** SHALL be "Indeterminate" with a  
3076 StatusCode value of "urn:oasis:names:tc:xacml:1.0:status:processing-error".

---

# 3077 8. XACML extensibility points (non-normative)

3078 This section describes the points within the XACML model and schema where extensions can be  
3079 added

## 3080 8.1. Extensible XML attribute types

3081 The following XML attributes have values that are URIs. These may be extended by the creation of  
3082 new URIs associated with new semantics for these attributes.

3083 AttributeId,

3084 AttributeValue,

3085 DataType,

3086 FunctionId,

3087 MatchId,

3088 ObligationId,

3089 PolicyCombiningAlgId,

3090 RuleCombiningAlgId,

3091    StatusCode,  
3092    SubjectCategory.  
3093    See Section 5 for definitions of these attribute types.

## 3094        **8.2.    Structured attributes**

3095    An XACML <AttributeValue> element MAY contain an instance of a structured XML data-type.  
3096    Section A.3 describes a number of standard techniques to identify data items within such a  
3097    structured attribute. Listed here are some additional techniques that require XACML extensions.

- 3098        1. For a given structured data-type, a community of XACML users MAY define new attribute  
3099        identifiers for each leaf sub-element of the structured data-type that has a type conformant  
3100        with one of the XACML-defined primitive data-types. Using these new attribute identifiers,  
3101        the **PEPs** or **context handlers** used by that community of users can flatten instances of  
3102        the structured data-type into a sequence of individual <Attribute> elements. Each such  
3103        <Attribute> element can be compared using the XACML-defined functions. Using this  
3104        method, the structured data-type itself never appears in an <AttributeValue> element.
- 3105        2. A community of XACML users MAY define a new function that can be used to compare a  
3106        value of the structured data-type against some other value. This method may only be used  
3107        by **PDPs** that support the new function.

---

## 3108        **9. Security and privacy considerations (non-** 3109        **normative)**

3110    This section identifies possible security and privacy compromise scenarios that should be  
3111    considered when implementing an XACML-based system. The section is informative only. It is left  
3112    to the implementer to decide whether these compromise scenarios are practical in their  
3113    environment and to select appropriate safeguards.

### 3114        **9.1.    Threat model**

3115    We assume here that the adversary has access to the communication channel between the  
3116    XACML actors and is able to interpret, insert, delete and modify messages or parts of messages.

3117    Additionally, an actor may use information from a former transaction maliciously in subsequent  
3118    transactions. It is further assumed that **rules** and **policies** are only as reliable as the actors that  
3119    create and use them. Thus it is incumbent on each actor to establish appropriate trust in the other  
3120    actors upon which it relies. Mechanisms for trust establishment are outside the scope of this  
3121    specification.

3122    The messages that are transmitted between the actors in the XACML model are susceptible to  
3123    attack by malicious third parties. Other points of vulnerability include the **PEP**, the **PDP** and the  
3124    **PAP**. While some of these entities are not strictly within the scope of this specification, their  
3125    compromise could lead to the compromise of **access control** enforced by the **PEP**.

3126    It should be noted that there are other components of a distributed system that may be  
3127    compromised, such as an operating system and the domain-name system (DNS) that are outside  
3128    the scope of this discussion of threat models. Compromise in these components may also lead to a  
3129    policy violation.

3130 The following sections detail specific compromise scenarios that may be relevant to an XACML  
3131 system.

### 3132 **9.1.1. Unauthorized disclosure**

3133 XACML does not specify any inherent mechanisms for confidentiality of the messages exchanged  
3134 between actors. Therefore, an adversary could observe the messages in transit. Under certain  
3135 security policies, disclosure of this information is a violation. Disclosure of **attributes** or the types  
3136 of **decision requests** that a **subject** submits may be a breach of privacy policy. In the commercial  
3137 sector, the consequences of unauthorized disclosure of personal data may range from  
3138 embarrassment to the custodian to imprisonment and large fines in the case of medical or financial  
3139 data.

3140 Unauthorized disclosure is addressed by confidentiality mechanisms.

### 3141 **9.1.2. Message replay**

3142 A message replay attack is one in which the adversary records and replays legitimate messages  
3143 between XACML actors. This attack may lead to denial of service, the use of out-of-date  
3144 information or impersonation.

3145 Prevention of replay attacks requires the use of message freshness mechanisms.

3146 Note that encryption of the message does not mitigate a replay attack since the message is just  
3147 replayed and does not have to be understood by the adversary.

### 3148 **9.1.3. Message insertion**

3149 A message insertion attack is one in which the adversary inserts messages in the sequence of  
3150 messages between XACML actors.

3151 The solution to a message insertion attack is to use mutual authentication and a message  
3152 sequence integrity mechanism between the actors. It should be noted that just using SSL mutual  
3153 authentication is not sufficient. This only proves that the other party is the one identified by the  
3154 subject of the X.509 certificate. In order to be effective, it is necessary to confirm that the certificate  
3155 subject is authorized to send the message.

### 3156 **9.1.4. Message deletion**

3157 A message deletion attack is one in which the adversary deletes messages in the sequence of  
3158 messages between XACML actors. Message deletion may lead to denial of service. However, a  
3159 properly designed XACML system should not render an incorrect authorization decision as a result  
3160 of a message deletion attack.

3161 The solution to a message deletion attack is to use a message integrity mechanism between the  
3162 actors.

### 3163 **9.1.5. Message modification**

3164 If an adversary can intercept a message and change its contents, then they may be able to alter an  
3165 **authorization decision**. Message integrity mechanisms can prevent a successful message  
3166 modification attack.

3167

### 9.1.6. NotApplicable results

3168 A result of "NotApplicable" means that the *PDP* did not have a policy whose target matched the  
3169 information in the *decision request*. In general, we highly recommend using a "default-deny"  
3170 policy, so that when a *PDP* would have returned "NotApplicable", a result of "Deny" is returned  
3171 instead.

3172 In some security models, however, such as is common in many Web Servers, a result of  
3173 "NotApplicable" is treated as equivalent to "Permit". There are particular security considerations  
3174 that must be taken into account for this to be safe. These are explained in the following  
3175 paragraphs.

3176 If "NotApplicable" is to be treated as "Permit", it is vital that the matching algorithms used by the  
3177 policy to match elements in the decision request are closely aligned with the data syntax used by  
3178 the applications that will be submitting the decision request. A failure to match will be treated as  
3179 "Permit", so an unintended failure to match may allow unintended access.

3180 A common example of this is a Web Server. Commercial http responders allow a variety of  
3181 syntaxes to be treated equivalently. The "%" can be used to represent characters by hex value.  
3182 The URL path "/./" provides multiple ways of specifying the same value. Multiple character sets  
3183 may be permitted and, in some cases, the same printed character can be represented by different  
3184 binary values. Unless the matching algorithm used by the policy is sophisticated enough to catch  
3185 these variations, unintended access may be permitted.

3186 It is safe to treat "NotApplicable" as "Permit" only in a closed environment where all applications  
3187 that formulate a decision request can be guaranteed to use the exact syntax expected by the  
3188 policies used by the *PDP*. In a more open environment, where decision requests may be received  
3189 from applications that may use any legal syntax, it is strongly recommended that "NotApplicable"  
3190 NOT be treated as "Permit" unless matching rules have been very carefully designed to match all  
3191 possible applicable inputs, regardless of syntax or type variations.

3192

### 9.1.7. Negative rules

3193 A negative *rule* is one that is based on a *predicate* not being "True". If not used with care,  
3194 negative *rules* can lead to policy violation, therefore some authorities recommend that they not be  
3195 used. However, negative *rules* can be extremely efficient in certain cases, so XACML has chosen  
3196 to include them. Nevertheless, it is recommended that they be used with care and avoided if  
3197 possible.

3198 A common use for negative *rules* is to deny *access* to an individual or subgroup when their  
3199 membership in a larger group would otherwise permit them access. For example, we might want to  
3200 write a *rule* that allows all Vice Presidents to see the unpublished financial data, except for Joe,  
3201 who is only a Ceremonial Vice President and can be indiscreet in his communications. If we have  
3202 complete control of the administration of *subject attributes*, a superior approach would be to  
3203 define "Vice President" and "Ceremonial Vice President" as distinct groups and then define *rules*  
3204 accordingly. However, in some environments this approach may not be feasible. (It is worth noting  
3205 in passing that, generally speaking, referring to individuals in *rules* does not scale well. Generally,  
3206 shared *attributes* are preferred.)

3207 If not used with care, negative *rules* can lead to policy violation in two common cases. They are:  
3208 when *attributes* are suppressed and when the base group changes. An example of suppressed  
3209 *attributes* would be if we have a policy that *access* should be permitted, *unless* the *subject* is a  
3210 credit risk. If it is possible that the *attribute* of being a credit risk may be unknown to the *PDP* for  
3211 some reason, then unauthorized *access* may be permitted. In some environments, the *subject*  
3212 may be able to suppress the publication of *attributes* by the application of privacy controls, or the  
3213 server or repository that contains the information may be unavailable for accidental or intentional  
3214 reasons.

3215 An example of a changing base group would be if there is a policy that everyone in the engineering  
3216 department may change software source code, except for secretaries. Suppose now that the  
3217 department was to merge with another engineering department and the intent is to maintain the  
3218 same policy. However, the new department also includes individuals identified as administrative  
3219 assistants, who ought to be treated in the same way as secretaries. Unless the policy is altered,  
3220 they will unintentionally be permitted to change software source code. Problems of this type are  
3221 easy to avoid when one individual administers all **policies**, but when administration is distributed,  
3222 as XACML allows, this type of situation must be explicitly guarded against.

## 3223 **9.2. Safeguards**

### 3224 **9.2.1. Authentication**

3225 Authentication provides the means for one party in a transaction to determine the identity of the  
3226 other party in the transaction. Authentication may be in one direction, or it may be bilateral.

3227 Given the sensitive nature of **access control** systems, it is important for a **PEP** to authenticate the  
3228 identity of the **PDP** to which it sends **decision requests**. Otherwise, there is a risk that an  
3229 adversary could provide false or invalid **authorization decisions**, leading to a policy violation.

3230 It is equally important for a **PDP** to authenticate the identity of the **PEP** and assess the level of trust  
3231 to determine what, if any, sensitive data should be passed. One should keep in mind that even  
3232 simple "Permit" or "Deny" responses could be exploited if an adversary were allowed to make  
3233 unlimited requests to a **PDP**.

3234 Many different techniques may be used to provide authentication, such as co-located code, a  
3235 private network, a VPN or digital signatures. Authentication may also be performed as part of the  
3236 communication protocol used to exchange the **contexts**. In this case, authentication may be  
3237 performed at the message level or at the session level.

### 3238 **9.2.2. Policy administration**

3239 If the contents of **policies** are exposed outside of the **access control** system, potential **subjects**  
3240 may use this information to determine how to gain unauthorized **access**.

3241 To prevent this threat, the repository used for the storage of **policies** may itself require **access**  
3242 **control**. In addition, the <Status> element should be used to return values of missing **attributes**  
3243 only when exposure of the identities of those **attributes** will not compromise security.

### 3244 **9.2.3. Confidentiality**

3245 Confidentiality mechanisms ensure that the contents of a message can be read only by the desired  
3246 recipients and not by anyone else who encounters the message while it is in transit. There are two  
3247 areas in which confidentiality should be considered: one is confidentiality during transmission; the  
3248 other is confidentiality within a <Policy> element.

#### 3249 **9.2.3.1. Communication confidentiality**

3250 In some environments it is deemed good practice to treat all data within an **access control** system  
3251 as confidential. In other environments, **policies** may be made freely available for distribution,  
3252 inspection and audit. The idea behind keeping **policy** information secret is to make it more difficult  
3253 for an adversary to know what steps might be sufficient to obtain unauthorized **access**. Regardless  
3254 of the approach chosen, the security of the **access control** system should not depend on the  
3255 secrecy of the **policy**.

3256 Any security concerns or requirements related to transmitting or exchanging XACML <Policy>  
3257 elements are outside the scope of the XACML standard. While it is often important to ensure that  
3258 the integrity and confidentiality of <Policy> elements is maintained when they are exchanged  
3259 between two parties, it is left to the implementers to determine the appropriate mechanisms for their  
3260 environment.

3261 Communications confidentiality can be provided by a confidentiality mechanism, such as SSL.  
3262 Using a point-to-point scheme like SSL may lead to other vulnerabilities when one of the end-points  
3263 is compromised.

### 3264 **9.2.3.2. Statement level confidentiality**

3265 In some cases, an implementation may want to encrypt only parts of an XACML <Policy>  
3266 element.

3267 The XML Encryption Syntax and Processing Candidate Recommendation from W3C can be used  
3268 to encrypt all or parts of an XML document. This specification is recommended for use with  
3269 XACML.

3270 It should go without saying that if a repository is used to facilitate the communication of cleartext  
3271 (i.e., unencrypted) **policy** between the **PAP** and **PDP**, then a secure repository should be used to  
3272 store this sensitive data.

### 3273 **9.2.4. Policy integrity**

3274 The XACML **policy**, used by the **PDP** to evaluate the request **context**, is the heart of the system.  
3275 Therefore, maintaining its integrity is essential. There are two aspects to maintaining the integrity of  
3276 the **policy**. One is to ensure that <Policy> elements have not been altered since they were  
3277 originally created by the **PAP**. The other is to ensure that <Policy> elements have not been  
3278 inserted or deleted from the set of **policies**.

3279 In many cases, both aspects can be achieved by ensuring the integrity of the actors and  
3280 implementing session-level mechanisms to secure the communication between actors. The  
3281 selection of the appropriate mechanisms is left to the implementers. However, when **policy** is  
3282 distributed between organizations to be acted on at a later time, or when the **policy** travels with the  
3283 protected resource, it would be useful to sign the **policy**. In these cases, the XML Signature  
3284 Syntax and Processing standard from W3C is recommended to be used with XACML.

3285 Digital signatures should only be used to ensure the integrity of the statements. Digital signatures  
3286 should not be used as a method of selecting or evaluating **policy**. That is, the **PDP** should not  
3287 request a **policy** based on who signed it or whether or not it has been signed (as such a basis for  
3288 selection would, itself, be a matter of policy). However, the **PDP** must verify that the key used to  
3289 sign the **policy** is one controlled by the purported issuer of the **policy**. The means to do this are  
3290 dependent on the specific signature technology chosen and are outside the scope of this document.

### 3291 **9.2.5. Policy identifiers**

3292 Since **policies** can be referenced by their identifiers, it is the responsibility of the **PAP** to ensure  
3293 that these are unique. Confusion between identifiers could lead to misidentification of the  
3294 **applicable policy**. This specification is silent on whether a **PAP** must generate a new identifier  
3295 when a **policy** is modified or may use the same identifier in the modified **policy**. This is a matter of  
3296 administrative practice. However, care must be taken in either case. If the identifier is reused,  
3297 there is a danger that other **policies** or **policy sets** that reference it may be adversely affected.  
3298 Conversely, if a new identifier is used, these other **policies** may continue to use the prior **policy**,  
3299 unless it is deleted. In either case the results may not be what the **policy** administrator intends.

3300

## 9.2.6. Trust model

3301 Discussions of authentication, integrity and confidentiality mechanisms necessarily assume an  
3302 underlying trust model: how can one actor come to believe that a given key is uniquely associated  
3303 with a specific, identified actor so that the key can be used to encrypt data for that actor or verify  
3304 signatures (or other integrity structures) from that actor? Many different types of trust model exist,  
3305 including strict hierarchies, distributed authorities, the Web, the bridge and so on.

3306 It is worth considering the relationships between the various actors of the **access control** system in  
3307 terms of the interdependencies that do and do not exist.

- 3308 • None of the entities of the authorization system are dependent on the **PEP**. They may  
3309 collect data from it, for example authentication, but are responsible for verifying it.
- 3310 • The correct operation of the system depends on the ability of the **PEP** to actually enforce  
3311 **policy** decisions.
- 3312 • The **PEP** depends on the **PDP** to correctly evaluate **policies**. This in turn implies that the  
3313 **PDP** is supplied with the correct inputs. Other than that, the **PDP** does not depend on the  
3314 **PEP**.
- 3315 • The **PDP** depends on the **PAP** to supply appropriate policies. The **PAP** is not dependent  
3316 on other components.

3317

## 9.2.7. Privacy

3318 It is important to be aware that any transactions that occur with respect to **access control** may  
3319 reveal private information about the actors. For example, if an XACML **policy** states that certain  
3320 data may only be read by **subjects** with "Gold Card Member" status, then any transaction in which  
3321 a **subject** is permitted **access** to that data leaks information to an adversary about the **subject's**  
3322 status. Privacy considerations may therefore lead to encryption and/or to **access control policies**  
3323 surrounding the enforcement of XACML **policy** instances themselves: confidentiality-protected  
3324 channels for the request/response protocol messages, protection of **subject attributes** in storage  
3325 and in transit, and so on.

3326 Selection and use of privacy mechanisms appropriate to a given environment are outside the scope  
3327 of XACML. The decision regarding whether, how and when to deploy such mechanisms is left to  
3328 the implementers associated with the environment.

---

# 10. Conformance (normative)

3329

## 10.1. Introduction

3330

3331 The XACML specification addresses the following aspect of conformance:

3332 The XACML specification defines a number of functions, etc. that have somewhat specialist  
3333 application, therefore they are not required to be implemented in an implementation that claims to  
3334 conform with the OASIS standard.

## 10.2. Conformance tables

3335

3336 This section lists those portions of the specification that **MUST** be included in an implementation of  
3337 a **PDP** that claims to conform with XACML v1.0. A set of test cases has been created to assist in  
3338 this process. These test cases are hosted by Sun Microsystems and can be located from the

3339 XACML Web page. The site hosting the test cases contains a full description of the test cases and  
 3340 how to execute them.

3341 Note: "M" means mandatory-to-implement. "O" means optional.

3342 **10.2.1. Schema elements**

3343 The implementation MUST support those schema elements that are marked "M".

Element name	M/O
xacml-context:Action	M
xacml-context:Attribute	M
xacml-context:AttributeValue	M
xacml-context:Decision	M
xacml-context:Environment	M
xacml-context:Obligations	O
xacml-context:Request	M
xacml-context:Resource	M
xacml-context:ResourceContent	O
xacml-context:Response	M
xacml-context:Result	M
xacml-context:Status	M
xacml-context:StatusCode	M
xacml-context:StatusDetail	O
xacml-context:StatusMessage	O
xacml-context:Subject	M
xacml:Action	M
xacml:ActionAttributeDesignator	M
xacml:ActionMatch	M
xacml:Actions	M
xacml:AnyAction	M
xacml:AnyResource	M
xacml:AnySubject	M
xacml:Apply	M
xacml:AttributeAssignment	O
xacml:AttributeSelector	O
xacml:AttributeValue	M
xacml:Condition	M
xacml:Description	M
xacml:EnvironmentAttributeDesignator	M
xacml:Function	M
xacml:Obligation	O
xacml:Obligations	O
xacml:Policy	M
xacml:PolicyDefaults	O
xacml:PolicyIdReference	M
xacml:PolicySet	M
xacml:PolicySetDefaults	O
xacml:PolicySetIdReference	M
xacml:Resource	M
xacml:ResourceAttributeDesignator	M
xacml:ResourceMatch	M
xacml:Resources	M
xacml:Rule	M
xacml:Subject	M
xacml:SubjectMatch	M
xacml:Subjects	M

xacml:Target	M
xacml:XPathVersion	O

3344 **10.2.2. Identifier Prefixes**

3345 The following identifier prefixes are reserved by XACML.

Identifier
urn:oasis:names:tc:xacml:1.0
urn:oasis:names:tc:xacml:1.0:conformance-test
urn:oasis:names:tc:xacml:1.0:context
urn:oasis:names:tc:xacml:1.0:example
urn:oasis:names:tc:xacml:1.0:function
urn:oasis:names:tc:xacml:1.0: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

3346 **10.2.3. Algorithms**

3347 The implementation MUST include the rule- and policy-combining algorithms associated with the  
 3348 following identifiers that are marked "M".

Algorithm	M/O
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:1.0: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-applicable	M
urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides	
urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides	
urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides	
urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides	

3349 **10.2.4. Status Codes**

3350 Implementation support for the urn:oasis:names:tc:xacml:1.0:context:status element is optional, but  
 3351 if the element is supported, then the following status codes must be supported and must be used in  
 3352 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
--	---

3353 **10.2.5. Attributes**

3354 The implementation MUST support the attributes associated with the following attribute identifiers  
 3355 as specified by XACML. If values for these **attributes** are not present in the **decision request**,  
 3356 then their values MUST be supplied by the **PDP**. So, unlike most other **attributes**, their semantics  
 3357 are not 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

3358 **10.2.6. Identifiers**

3359 The implementation MUST use the attributes associated with the following identifiers in the way  
 3360 XACML has defined. This requirement pertains primarily to implementations of a **PAP** or **PEP** that  
 3361 use XACML, 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:scope	O
urn:oasis:names:tc:xacml:1.0:resource:simple-file-name	O
urn:oasis:names:tc:xacml:1.0:action:action-id	M
urn:oasis:names:tc:xacml:1.0:action:implied-action	M

3362 **10.2.7. Data-types**

3363 The implementation MUST support the data-types associated with the following identifiers marked  
 3364 "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/TR/2002/WD-xquery-operators-20020816#dayTimeDuration	M

http://www.w3.org/TR/2002/WD-xquery-operators-20020816#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

3365

## 10.2.8. Functions

3366

The implementation MUST properly process those functions associated with the identifiers marked with an "M".

3367

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: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: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:present	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: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: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: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: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:regexp-string-match	M
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: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

3368

---

## 11. References

3369

- 3370 [DS] D. Eastlake et al., *XML-Signature Syntax and Processing*,  
3371 <http://www.w3.org/TR/xmlsig-core/>, World Wide Web Consortium.
- 3372 [Hancock] Hancock, "Polymorphic Type Checking", in Simon L. Peyton Jones,  
3373 "Implementation of Functional Programming Languages", Section 8,  
3374 Prentice-Hall International, 1987
- 3375 [Haskell] Haskell, a purely functional language. Available at  
3376 <http://www.haskell.org/>
- 3377 [Hinton94] Hinton, H, M, Lee,, E, S, The Compatibility of Policies, Proceedings 2nd  
3378 ACM Conference on Computer and Communications Security, Nov 1994,  
3379 Fairfax, Virginia, USA.
- 3380 [IEEE754] IEEE Standard for Binary Floating-Point Arithmetic 1985, ISBN 1-5593-  
3381 7653-8, IEEE Product No. SH10116-TBR
- 3382 [Kudo00] Kudo M and Hada S, XML document security based on provisional  
3383 authorization, Proceedings of the Seventh ACM Conference on Computer  
3384 and Communications Security, Nov 2000, Athens, Greece, pp 87-96.
- 3385 [LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3,  
3386 Section 5, M Wahl, December 1997 <http://www.ietf.org/rfc/rfc2798.txt>
- 3387 [LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000  
3388 <http://www.ietf.org/rfc/rfc2798.txt>
- 3389 [MathML] Mathematical Markup Language (MathML), Version 2.0, W3C  
3390 Recommendation, 21 February 2001. Available at:  
3391 <http://www.w3.org/TR/MathML2/>
- 3392 [Perritt93] Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference  
3393 on Technological Strategies for Protecting Intellectual Property in the  
3394 Networked Multimedia Environment, April 1993. Available at:  
3395 <http://www.ifla.org/documents/infopol/copyright/perh2.txt>
- 3396 [RBAC] Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th  
3397 National Computer Security Conference, 1992. Available at:  
3398 <http://csrc.nist.gov/rbac>
- 3399 [RegEx] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001,  
3400 Appendix D. Available at: <http://www.w3.org/TR/xmlschema-0/>
- 3401 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,  
3402 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997
- 3403 [SAML] Security Assertion Markup Language available from [http://www.oasis-  
3404 open.org/committees/security/#documents](http://www.oasis-open.org/committees/security/#documents)
- 3405 [Sloman94] Sloman, M. Policy Driven Management for Distributed Systems. *Journal*  
3406 *of Network and Systems Management*, Volume 2, part 4. Plenum Press.  
3407 1994.
- 3408 [XF] XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft  
3409 16 August 2002. Available at: [http://www.w3.org/TR/2002/WD-xquery-  
3410 operators-20020816](http://www.w3.org/TR/2002/WD-xquery-operators-20020816)
- 3411 [XS] XML Schema, parts 1 and 2. Available at:  
3412 <http://www.w3.org/TR/xmlschema-1/> and  
3413 <http://www.w3.org/TR/xmlschema-2/>
- 3414 [XPath] XML Path Language (XPath), Version 1.0, W3C Recommendation 16  
3415 November 1999. Available at: <http://www.w3.org/TR/xpath>

3416  
3417  
3418

**[XSLT]**

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

---

3419 **Appendix A. Standard data-types, functions and**  
3420 **their semantics (normative)**

3421 **A.1. Introduction**

3422 This section contains a specification of the data-types and functions used in XACML to create  
3423 **predicates** for a **rule's condition** and **target** matches.

3424 This specification combines the various standards set forth by IEEE and ANSI for string  
3425 representation of numeric values, as well as the evaluation of arithmetic functions.

3426 This section describes the primitive data-types, **bags** and construction of expressions using  
3427 XACML constructs. Finally, each standard function is named and its operational semantics are  
3428 described.

3429 **A.2. Primitive types**

3430 Although XML instances represent all data-types as strings, an XACML **PDP** must reason about  
3431 types of data that, while they have string representations, are not just strings. Types such as  
3432 boolean, integer and double **MUST** be converted from their XML string representations to values  
3433 that can be compared with values in their domain of discourse, such as numbers. The following  
3434 primitive data-types are specified for use with XACML and have explicit data representations:

- 3435 • <http://www.w3.org/2001/XMLSchema#string>
- 3436 • <http://www.w3.org/2001/XMLSchema#boolean>
- 3437 • <http://www.w3.org/2001/XMLSchema#integer>
- 3438 • <http://www.w3.org/2001/XMLSchema#double>
- 3439 • <http://www.w3.org/2001/XMLSchema#time>
- 3440 • <http://www.w3.org/2001/XMLSchema#date>
- 3441 • <http://www.w3.org/2001/XMLSchema#dateTime>
- 3442 • <http://www.w3.org/2001/XMLSchema#anyURI>
- 3443 • <http://www.w3.org/2001/XMLSchema#hexBinary>
- 3444 • <http://www.w3.org/2001/XMLSchema#base64Binary>
- 3445 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration>
- 3446 • <http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration>
- 3447 • <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
- 3448 • <urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name>

## 3449 A.3. Structured types

3450 An XACML `<AttributeValue>` element MAY contain an instance of a structured XML data-type,  
3451 for example `<ds:KeyInfo>`. XACML 1.0 supports several ways for comparing such  
3452 `<AttributeValue>` elements.

3453 1. In some cases, such an `<AttributeValue>` element MAY be compared using one of the  
3454 XACML string functions, such as “regexp-string-match”, described below. This requires  
3455 that the structured data `<AttributeValue>` be given the  
3456 `DataType="http://www.w3.org/2001/XMLSchema#string"`. For example, a structured data-  
3457 type that is actually a `ds:KeyInfo/KeyName` would appear in the Context as:

```
3458 <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string" >  
3459 <ds:KeyName>jhibbert-key</ds:KeyName>  
3460 </AttributeValue>
```

3461 In general, this method will not be adequate unless the structured data-type is quite simple.

3462 2. An `<AttributeSelector>` element MAY be used to select the value of a leaf sub-  
3463 element of the structured data-type by means of an XPath expression. That value MAY  
3464 then be compared using one of the supported XACML functions appropriate for its primitive  
3465 data-type. This method requires support by the **PDP** for the optional XPath expressions  
3466 feature.

3467 3. An `<AttributeSelector>` element MAY be used to select the value of any node in the  
3468 structured data-type by means of an XPath expression. This node MAY then be compared  
3469 using one of the XPath-based functions described in Section A14.13. This method requires  
3470 support by the **PDP** for the optional XPath expressions and XPath functions features.

## 3471 A.4. Representations

3472 An XACML **PDP** SHALL be capable of converting string representations into various primitive data-  
3473 types. For integers and doubles, XACML SHALL use the conversions described in [IEEE754].

3474 This document combines the various standards set forth by IEEE and ANSI for string  
3475 representation of numeric values.

3476 XACML defines two additional data-types; these are “urn:oasis:names:tc:xacml:1.0:data-  
3477 type:x500Name” and “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name”. These types  
3478 represent identifiers for **subjects** and appear in several standard applications, such as TLS/SSL  
3479 and electronic mail.

3480 The “urn:oasis:names:tc:xacml:1.0:data-type:x500Name” primitive type represents an X.500  
3481 Distinguished Name. The string representation of an X.500 distinguished name is specified in IETF  
3482 RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of  
3483 Distinguished Names".<sup>1</sup>

3484 The “urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name” primitive type represents electronic mail  
3485 addresses, and its string representation is specified by RFC 822.

---

1 An earlier RFC, RFC 1779 "A String Representation of Distinguished Names", is less restrictive, so urn:oasis:names:tc:xacml:1.0:data-type:x500Name uses the syntax in RFC 2253 for better interoperability.

3486 An RFC822 name consists of a *local-part* followed by "@" followed by a *domain-part*. The *local-*  
3487 *part* is case-sensitive, while the *domain-part* (which is usually a DNS host name) is not case-  
3488 sensitive.<sup>2</sup>

## 3489 A.5. Bags

3490 XACML defines implicit collections of its primitive types. XACML refers to a collection of values that  
3491 are of a single primitive type as a **bag**. **Bags** of primitive types are needed because selections of  
3492 nodes from an XML **resource** or XACML request **context** may return more than one value.

3493 The <AttributeSelector> element uses an XPath expression to specify the selection of data  
3494 from an XML **resource**. The result of an XPath expression is termed a *node-set*, which contains all  
3495 the leaf nodes from the XML **resource** that match the predicate in the XPath expression. Based on  
3496 the various indexing functions provided in the XPath specification, it SHALL be implied that a  
3497 resultant node-set is the collection of the matching nodes. XACML also defines the  
3498 <AttributeDesignator> **element** to have the same matching methodology for attributes in the  
3499 XACML request **context**.

3500 The values in a **bag** are not ordered, and some of the values may be duplicates. There SHALL be  
3501 no notion of a **bag** containing **bags**, or a **bag** containing values of differing types. I.e. a **bag** in  
3502 XACML SHALL contain only values that are of the same primitive type.

## 3503 A.6. Expressions

3504 XACML specifies expressions in terms of the following elements, of which the <Apply> and  
3505 <Condition> elements recursively compose greater expressions. Valid expressions shall be type  
3506 correct, which means that the types of each of the elements contained within <Apply> and  
3507 <Condition> elements shall agree with the respective argument types of the function that is  
3508 named by the FunctionId attribute. The resultant type of the <Apply> or <Condition>  
3509 element shall be the resultant type of the function, which may be narrowed to a primitive data-type,  
3510 or a bag of a primitive data-type, by type-unification. XACML defines an evaluation result of  
3511 "Indeterminate", which is said to be the result of an invalid expression, or an operational error  
3512 occurring during the evaluation of the expression.

3513 XACML defines the following elements to be legal XACML expressions:

- 3514 • <AttributeValue>
- 3515 • <SubjectAttributeDesignator>
- 3516 • <SubjectAttributeSelector>
- 3517 • <ResourceAttributeDesignator>
- 3518 • <ActionAttributeDesignator>
- 3519 • <EnvironmentAttributeDesignator>

---

2 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part*. However, many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This is considered an error by mail-system designers and is not encouraged.

- 3520 • <AttributeSelector>
- 3521 • <Apply>
- 3522 • <Condition>
- 3523 • <Function>

## 3524 A.7. Element <AttributeValue>

3525 The <AttributeValue> element SHALL represent an explicit value of a primitive type. For  
3526 example:

```
3527 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">  
3528   <AttributeValue  
3529     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3530   <AttributeValue  
3531     DataType="http://www.w3.org/2001/XMLSchema#integer">123</AttributeValue>  
3532 </Apply>
```

## 3533 A.8. Elements <AttributeDesignator> and 3534 <AttributeSelector>

3535 The <AttributeDesignator> and <AttributeSelector> elements SHALL evaluate to a **bag**  
3536 of a specific primitive type. The type SHALL be inferred from the function in which it appears. Each  
3537 element SHALL contain a URI or XPath expression, respectively, to identify the required **attribute**  
3538 values. If an operational error were to occur while finding the values, the value of the element  
3539 SHALL be set to "Indeterminate". If the required **attribute** cannot be located, then the value of the  
3540 element SHALL be set to an empty **bag** of the inferred primitive type.

## 3541 A.9. Element <Apply>

3542 XACML function calls are represented by the <Apply> element. The function to be applied is  
3543 named in the `FunctionId` attribute of this element. The value of the <Apply> element SHALL be  
3544 set to either a primitive data-type or a **bag** of a primitive type, whose data-type SHALL be inferred  
3545 from the `FunctionId`. The arguments of a function SHALL be the values of the XACML  
3546 expressions that are contained as ordered elements in an <Apply> element. The legal number of  
3547 arguments within an <Apply> element SHALL depend upon the `functionId`.

## 3548 A.10. Element <Condition>

3549 The <Condition> element MAY appear in the <Rule> element as the premise for emitting the  
3550 corresponding **effect** of the **rule**. The <Condition> element has the same structure as the  
3551 <Apply> element, with the restriction that its result SHALL be of data-type  
3552 "http://www.w3.org/2001/XMLSchema#boolean". The evaluation of the <Condition> element  
3553 SHALL follow the same evaluation semantics as those of the <Apply> element.

## 3554 A.11.Element <Function>

3555 The <Function> element names a standard XACML function or an extension function in its  
3556 FunctionId attribute. The <Function> element MAY be used as an argument in functions that  
3557 take a function as an argument.

## 3558 A.12.Matching elements

3559 Matching elements appear in the <Target> element of *rules*, *policies* and *policy sets*. They are  
3560 the following:

3561 <SubjectMatch>

3562 <ResourceMatch>

3563 <ActionMatch>

3564 These elements represent boolean expressions over attributes of the subject, resource, and action,  
3565 respectively. A matching element contains a MatchId attribute that specifies the function to be  
3566 used in performing the match evaluation, an **attribute value**, and an <AttributeDesignator>  
3567 or <AttributeSelector> element that specifies the **attribute** in the **context** that is to be  
3568 matched against the specified value.

3569 The MatchId attribute SHALL specify a function that compares two arguments, returning a result  
3570 type of "http://www.w3.org/2001/XMLSchema#boolean". The **attribute** value specified in the  
3571 matching element SHALL be supplied to the MatchId function as its first argument. An element of  
3572 the **bag** returned by the <AttributeDesignator> or <AttributeSelector> element SHALL  
3573 be supplied to the MatchId function as its second argument. The data-type of the **attribute** value  
3574 SHALL match the data-type of the first argument expected by the MatchId function. The data-type  
3575 of the <AttributeDesignator> or <AttributeSelector> element SHALL match the data-  
3576 type of the second argument expected by the MatchId function.

3577 The XACML standard functions that meet the requirements for use as a MatchId attribute value  
3578 are:

3579 urn:oasis:names:tc:xacml:1.0:function:-type-equal

3580 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than

3581 urn:oasis:names:tc:xacml:1.0:function:-type-greater-than-or-equal

3582 urn:oasis:names:tc:xacml:1.0:function:-type-less-than

3583 urn:oasis:names:tc:xacml:1.0:function:-type-less-than-or-equal

3584 urn:oasis:names:tc:xacml:1.0:function:-type-match

3585 In addition, functions that are strictly within an extension to XACML MAY appear as a value for the  
3586 MatchId attribute, and those functions MAY use data-types that are also extensions, so long as  
3587 the extension function returns a boolean result and takes an **attribute** value as its first argument  
3588 and an <AttributeDesignator> or <AttributeSelector> as its second argument. The  
3589 function used as the value for the MatchId attribute SHOULD be easily indexable. Use of non-  
3590 indexable or complex functions may prevent efficient evaluation of **decision requests**.

3591 The evaluation semantics for a matching element is as follows. If an operational error were to  
3592 occur while evaluating the <AttributeDesignator> or <AttributeSelector> element, then

3593 the result of the entire expression SHALL be "Indeterminate". If the <AttributeDesignator> or  
3594 <AttributeSelector> element were to evaluate to an empty **bag**, then the result of the  
3595 expression SHALL be "False". Otherwise, the MatchId function SHALL be applied between the  
3596 explicit **attribute** value and each element of the **bag** returned from the <AttributeDesignator>  
3597 or <AttributeSelector> element. If at least one of those function applications were to evaluate  
3598 to "True", then the result of the entire expression SHALL be "True". Otherwise, if at least one of the  
3599 function applications results in "Indeterminate", then the result SHALL be "Indeterminate". Finally,  
3600 only if all function applications evaluate to "False", the result of the entire expression SHALL be  
3601 "False".

3602 It is possible to express the semantics of a **target** matching element in a **condition**. For instance,  
3603 the **target** match expression that compares a "subject-name" starting with the name "John" can be  
3604 expressed as follows:

```
3605 <SubjectMatch  
3606   MatchId="urn:oasis:names:tc:xacml:1.0:function:regex-string-match">  
3607   <SubjectAttributeDesignator  
3608     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3609     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3610   <AttributeValue  
3611     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3612 </SubjectMatch>
```

3613 Alternatively, the same match semantics can be expressed as an <Apply> element in a **condition**  
3614 by using the "urn:oasis:names:tc:xacml:1.0:function:any-of" function, as follows:

```
3615 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">  
3616   <Function  
3617     FunctionId="urn:oasis:names:tc:xacml:1.0:function:regex-string-match"/>  
3618   <AttributeValue  
3619     DataType="http://www.w3.org/2001/XMLSchema#string">John.*</AttributeValue>  
3620   <SubjectAttributeDesignator  
3621     AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"  
3622     DataType="http://www.w3.org/2001/XMLSchema#string"/>  
3623 </Apply>
```

3624

3625 This expression of the semantics is NOT normative.

## 3626 A.13.Arithmetic evaluation

3627 IEEE 754 [IEEE 754] specifies how to evaluate arithmetic functions in a context, which specifies  
3628 defaults for precision, rounding, etc. XACML SHALL use this specification for the evaluation of all  
3629 integer and double functions relying on the *Extended Default Context*, enhanced with double  
3630 precision:

3631 flags - all set to 0

3632 trap-enablers - all set to 0 (IEEE 854 §7) with the exception of the "division-by-zero" trap  
3633 enabler, which SHALL be set to 1

3634 precision - is set to the designated double precision

3635 rounding - is set to round-half-even (IEEE 854 §4.1)

## 3636 A.14.XACML standard functions

3637 XACML specifies the following functions that are prefixed with the  
3638 "urn:oasis:names:tc:xacml:1.0:function:" relative name space identifier.

### 3639 A14.1 Equality predicates

3640 The following functions are the *equality* functions for the various primitive types. Each function for a  
3641 particular data-type follows a specified standard convention for that data-type. If an argument of  
3642 one of these functions were to evaluate to "Indeterminate", then the function SHALL be set to  
3643 "Indeterminate".

- 3644 • string-equal

3645 This function SHALL take two arguments of "http://www.w3.org/2001/XMLSchema#string"  
3646 and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean". The function  
3647 SHALL return "True" if and only if the value of both of its arguments are of equal length and  
3648 each string is determined to be equal byte-by-byte according to the function "integer-equal".

- 3649 • boolean-equal

3650 This function SHALL take two arguments of  
3651 "http://www.w3.org/2001/XMLSchema#boolean" and SHALL return "True" if and only if both  
3652 values are equal.

- 3653 • integer-equal

3654 This function SHALL take two arguments of data-type  
3655 "http://www.w3.org/2001/XMLSchema#integer" and SHALL return an  
3656 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on  
3657 integers according to IEEE 754 [IEEE 754].

- 3658 • double-equal

3659 This function SHALL take two arguments of data-type  
3660 "http://www.w3.org/2001/XMLSchema#double" and SHALL return an  
3661 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation on  
3662 doubles according to IEEE 754 [IEEE 754].

- 3663 • date-equal

3664 This function SHALL take two arguments of data-type  
3665 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
3666 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
3667 according to the "op:date-equal" function [XF Section 8.3.11].

- 3668 • time-equal

3669 This function SHALL take two arguments of data-type  
3670 "http://www.w3.org/2001/XMLSchema#time" and SHALL return an  
3671 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation according  
3672 to the "op:time-equal" function [XF Section 8.3.14].

- 3673 • dateTime-equal

3674 This function SHALL take two arguments of data-type  
3675 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an

3676 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
3677 according to the "op:dateTime-equal" function [XF Section 8.3.8].

3678 • dayTimeDuration-equal

3679 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-  
3680 xquery-operators-20020816#dayTimeDuration" and SHALL return an  
3681 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
3682 according to the "op:dayTimeDuration-equal" function [XF Section 8.3.5]. Note that the  
3683 lexical representation of each argument MUST be converted to a value expressed in  
3684 fractional seconds [XF Section 8.2.2].

3685 • yearMonthDuration-equal

3686 This function SHALL take two arguments of data-type "http://www.w3.org/TR/2002/WD-  
3687 xquery-operators-20020816#yearMonthDuration" and SHALL return an  
3688 "http://www.w3.org/2001/XMLSchema#boolean". This function shall perform its evaluation  
3689 according to the "op:yearMonthDuration-equal" function [XF Section 8.3.2]. Note that the  
3690 lexical representation of each argument MUST be converted to a value expressed in  
3691 integer months [XF Section 8.2.1].

3692 • anyURI-equal

3693 This function SHALL take two arguments of data-type  
3694 "http://www.w3.org/2001/XMLSchema#anyURI" and SHALL return an  
3695 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL perform its evaluation  
3696 according to the "op:anyURI-equal" function [XF Section 10.2.1].

3697 • x500Name-equal

3698 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-  
3699 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It  
3700 shall return "True" if and only if each Relative Distinguished Name (RDN) in the two  
3701 arguments matches. Two RDNs shall be said to match if and only if the result of the  
3702 following operations is "True"<sup>3</sup>.

- 3703 1. Normalize the two arguments according to IETF RFC 2253 "Lightweight Directory  
3704 Access Protocol (v3): UTF-8 String Representation of Distinguished Names".
- 3705 2. If any RDN contains multiple attributeTypeAndValue pairs, re-order the Attribute  
3706 ValuePairs in that RDN in ascending order when compared as octet strings  
3707 (described in ITU-T Rec. X.690 (1997 E) Section 11.6 "Set-of components").
- 3708 3. Compare RDNs using the rules in IETF RFC 3280 "Internet X.509 Public Key  
3709 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", Section  
3710 4.1.2.4 "Issuer".

3711 • rfc822Name-equal

3712 This function SHALL take two arguments of data-type "urn:oasis:names:tc:xacml:1.0:data-  
3713 type:rfc822Name" and SHALL return an "http://www.w3.org/2001/XMLSchema#boolean".  
3714 This function SHALL determine whether two "urn:oasis:names:tc:xacml:1.0:data-  
3715 type:rfc822Name" arguments are equal. An RFC822 name consists of a *local-part* followed  
3716 by "@" followed by a *domain-part*. The *local-part* is case-sensitive, while the *domain-part*  
3717 (which is usually a DNS host name) is not case-sensitive. Perform the following  
3718 operations:

---

<sup>3</sup> ITU-T Rec. X.520 contains rules for matching X500 names, but these are very complex and require knowledge of the syntax of various AttributeTypes. IETF RFC 3280 contains simplified matching rules that the XACML x500Name-equal function uses.

- 3719                   1. Normalize the *domain*-part of each argument to lower case
- 3720                   2. Compare the expressions by applying the function  
3721                    “urn:oasis:names:tc:xacml:1.0:function:string-equal” to the normalized arguments.
- 3722   • hexBinary-equal
- 3723                   This function SHALL take two arguments of data-type  
3724                   “http://www.w3.org/2001/XMLSchema#hexBinary” and SHALL return an  
3725                   “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the  
3726                   octet sequences represented by the value of both arguments have equal length and are  
3727                   equal in a conjunctive, point-wise, comparison using the  
3728                   “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string  
3729                   representation to an octet sequence SHALL be as specified in [XS Section 8.2.15]
- 3730   • base64Binary-equal
- 3731                   This function SHALL take two arguments of data-type  
3732                   “http://www.w3.org/2001/XMLSchema#base64Binary” and SHALL return an  
3733                   “http://www.w3.org/2001/XMLSchema#boolean”. This function SHALL return "True" if the  
3734                   octet sequences represented by the value of both arguments have equal length and are  
3735                   equal in a conjunctive, point-wise, comparison using the  
3736                   “urn:oasis:names:tc:xacml:1.0:function:integer-equal”. The conversion from the string  
3737                   representation to an octet sequence SHALL be as specified in [XS Section 8.2.16]

## 3738                   **A14.2 Arithmetic functions**

3739                   All of the following functions SHALL take two arguments of the specified *data-type*, integer or  
3740                   double, and SHALL return an element of integer or double data-type, respectively. However, the  
3741                   “add” functions MAY take more than two arguments. Each function evaluation SHALL proceed as  
3742                   specified by their logical counterparts in IEEE 754 [IEEE 754]. In an expression that contains any  
3743                   of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3744                   "Indeterminate". In the case of the divide functions, if the divisor is zero, then the function SHALL  
3745                   evaluate to “Indeterminate”.

- 3746   • integer-add
- 3747                   This function MAY have two or more arguments.
- 3748   • double-add
- 3749                   This function MAY have two or more arguments.
- 3750   • integer-subtract
- 3751   • double-subtract
- 3752   • integer-multiply
- 3753   • double-multiply
- 3754   • integer-divide
- 3755   • double-divide
- 3756   • integer-mod

3757                   The following functions SHALL take a single argument of the specified *data-type*. The round and  
3758                   floor functions SHALL take a single argument of data-type  
3759                   “http://www.w3.org/2001/XMLSchema#double” and return data-type

3760 "http://www.w3.org/2001/XMLSchema#double". In an expression that contains any of these  
3761 functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3762 "Indeterminate".

3763 • integer-abs

3764 • double-abs

3765 • round

3766 • floor

### 3767 **A14.3 String conversion functions**

3768 The following functions convert between values of the XACML  
3769 "http://www.w3.org/2001/XMLSchema#string" primitive types. In an expression that contains any of  
3770 these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3771 "Indeterminate".

3772 • string-normalize-space

3773 This function SHALL take one argument of data-type  
3774 "http://www.w3.org/2001/XMLSchema#string" and SHALL normalize the value by stripping  
3775 off all leading and trailing whitespace characters.

3776 • string-normalize-to-lower-case

3777 This function SHALL take one argument of "http://www.w3.org/2001/XMLSchema#string"  
3778 and SHALL normalize the value by converting each upper case character to its lower case  
3779 equivalent.

### 3780 **A14.4 Numeric data-type conversion functions**

3781 The following functions convert between the XACML  
3782 "http://www.w3.org/2001/XMLSchema#integer" and "http://www.w3.org/2001/XMLSchema#double"  
3783 primitive types. In any expression in which the functions defined below are applied, if any argument  
3784 while being evaluated results in "Indeterminate", the expression SHALL return "Indeterminate".

3785 • double-to-integer

3786 This function SHALL take one argument of data-type  
3787 "http://www.w3.org/2001/XMLSchema#double" and SHALL truncate its numeric value to a  
3788 whole number and return an element of data-type  
3789 "http://www.w3.org/2001/XMLSchema#integer".

3790 • integer-to-double

3791 This function SHALL take one argument of data-type  
3792 "http://www.w3.org/2001/XMLSchema#integer" and SHALL promote its value to an element  
3793 of data-type "http://www.w3.org/2001/XMLSchema#double" of the same numeric value.

### 3794 **A14.5 Logical functions**

3795 This section contains the specification for logical functions that operate on arguments of the  
3796 "http://www.w3.org/2001/XMLSchema#boolean" data-type.

3797

3798 • or  
3799 This function SHALL return "False" if it has no arguments and SHALL return "True" if one of  
3800 its arguments evaluates to "True". The order of evaluation SHALL be from first argument to  
3801 last. The evaluation SHALL stop with a result of "True" if any argument evaluates to "True",  
3802 leaving the rest of the arguments unevaluated. In an expression that contains any of these  
3803 functions, if ANY argument to this function evaluates to "Indeterminate", then the  
3804 expression SHALL evaluate to "Indeterminate".

3805 • and  
3806 This function SHALL return "True" if it has no arguments and SHALL return "False" if one of  
3807 its arguments evaluates to "False". The order of evaluation SHALL be from first argument  
3808 to last. The evaluation SHALL stop with a result of "False" if any argument evaluates to  
3809 "False", leaving the rest of the arguments unevaluated. In an expression that contains any  
3810 of these functions, if ANY argument to this function evaluates to "Indeterminate", then the  
3811 expression SHALL evaluate to "Indeterminate".

3812 • n-of  
3813 The first argument to this function SHALL be of data-type  
3814 "http://www.w3.org/2001/XMLSchema#integer", specifying the number of the remaining  
3815 arguments that MUST evaluate to "True" for the expression to be considered "True". If the  
3816 first argument is 0, the result SHALL be "True". If the number of arguments after the first  
3817 one is less than the value of the first argument, then the expression SHALL result in  
3818 "Indeterminate". The order of evaluation SHALL be: first evaluate the integer value, then  
3819 evaluate each subsequent argument. The evaluation SHALL stop and return "True" if the  
3820 specified number of arguments evaluate to "True". The evaluation of arguments SHALL  
3821 stop if it is determined that evaluating the remaining arguments will not satisfy the  
3822 requirement. In an expression that contains any of these functions, if ANY argument to this  
3823 function evaluates to "Indeterminate", then the expression SHALL evaluate to  
3824 "Indeterminate".

3825 • not  
3826 This function SHALL take one logical argument. If the argument evaluates to "True", then  
3827 the result of the expression SHALL be "False". If the argument evaluates to "False", then  
3828 the result of the expression SHALL be "True". In an expression that contains any of these  
3829 functions, if ANY argument to this function evaluates to "Indeterminate", then the  
3830 expression SHALL evaluate to "Indeterminate".

3831 Note: For an expression that is an application of AND, OR, or N-OF, it MAY NOT be necessary to  
3832 attempt a full evaluation of each boolean argument to a truth value in order to determine whether  
3833 the evaluation of the argument would result in "Indeterminate". Analysis of the argument regarding  
3834 its necessary attributes, or other analysis regarding errors, such as "divide-by-zero", may render the  
3835 argument error free. Such arguments occurring in the expression in a position after the evaluation is  
3836 stated to stop need not be processed.

## 3837 **A14.6 Arithmetic comparison functions**

3838 These functions form a minimal set for comparing two numbers, yielding a boolean result. They  
3839 SHALL comply with the rules governed by IEEE 754 [IEEE 754]. In an expression that contains  
3840 any of these functions, if any argument is "Indeterminate", then the expression SHALL evaluate to  
3841 "Indeterminate".

3842 • integer-greater-than

3843 • integer-greater-than-or-equal

- 3844 • integer-less-than
- 3845 • integer-less-than-or-equal
- 3846 • double-greater-than
- 3847 • double-greater-than-or-equal
- 3848 • double-less-than
- 3849 • double-less-than-or-equal

## 3850 **A14.7 Date and time arithmetic functions**

3851 These functions perform arithmetic operations with the date and time. In an expression that  
 3852 contains any of these functions, if any argument is "Indeterminate", then the expression SHALL  
 3853 evaluate to "Indeterminate".

- 3854 • dateTime-add-dayTimeDuration

3855 This function SHALL take two arguments, the first is of data-type  
 3856 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is of data-type  
 3857 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL  
 3858 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function SHALL  
 3859 return the value by adding the second argument to the first argument according to the  
 3860 specification of adding durations to date and time [XS Appendix E].

- 3861 • dateTime-add-yearMonthDuration

3862 This function SHALL take two arguments, the first is a  
 3863 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a  
 3864 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It  
 3865 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". This function  
 3866 SHALL return the value by adding the second argument to the first argument according to  
 3867 the specification of adding durations to date and time [XS Appendix E].

- 3868 • dateTime-subtract-dayTimeDuration

3869 This function SHALL take two arguments, the first is a  
 3870 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a  
 3871 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration". It SHALL  
 3872 return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second argument  
 3873 is a positive duration, then this function SHALL return the value by adding the  
 3874 corresponding negative duration, as per the specification [XS Appendix E]. If the second  
 3875 argument is a negative duration, then the result SHALL be as if the function  
 3876 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-dayTimeDuration" had been applied  
 3877 to the corresponding positive duration.

- 3878 • dateTime-subtract-yearMonthDuration

3879 This function SHALL take two arguments, the first is a  
 3880 "http://www.w3.org/2001/XMLSchema#dateTime" and the second is a  
 3881 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It  
 3882 SHALL return a result of "http://www.w3.org/2001/XMLSchema#dateTime". If the second  
 3883 argument is a positive duration, then this function SHALL return the value by adding the  
 3884 corresponding negative duration, as per the specification [XS Appendix E]. If the second  
 3885 argument is a negative duration, then the result SHALL be as if the function  
 3886 "urn:oasis:names:tc:xacml:1.0:function:dateTime-add-yearMonthDuration" had been  
 3887 applied to the corresponding positive duration.

3888 • date-add-yearMonthDuration  
3889 This function SHALL take two arguments, the first is a  
3890 "http://www.w3.org/2001/XMLSchema#date" and the second is a  
3891 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It  
3892 return a result of "http://www.w3.org/2001/XMLSchema#date". This function SHALL return  
3893 the value by adding the second argument to the first argument according to the  
3894 specification of adding durations to date [XS Appendix E].

3895 • date-subtract-yearMonthDuration  
3896 This function SHALL take two arguments, the first is a  
3897 "http://www.w3.org/2001/XMLSchema#date" and the second is a  
3898 "http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration". It  
3899 SHALL return a result of "http://www.w3.org/2001/XMLSchema#date". If the second  
3900 argument is a positive duration, then this function SHALL return the value by adding the  
3901 corresponding negative duration, as per the specification [XS Appendix E]. If the second  
3902 argument is a negative duration, then the result SHALL be as if the function  
3903 "urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration" had been applied to  
3904 the corresponding positive duration.

## 3905 **A14.8 Non-numeric comparison functions**

3906 These functions perform comparison operations on two arguments of non-numerical types. In an  
3907 expression that contains any of these functions, if any argument is "Indeterminate", then the  
3908 expression SHALL evaluate to "Indeterminate".

3909 • string-greater-than  
3910 This function SHALL take two arguments of data-type  
3911 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
3912 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
3913 arguments are compared byte by byte and, after an initial prefix of corresponding bytes  
3914 from both arguments that are considered equal by  
3915 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is  
3916 such that the byte from the first argument is greater than the byte from the second  
3917 argument by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-equal".

3918 • string-greater-than-or-equal  
3919 This function SHALL take two arguments of data-type  
3920 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
3921 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return a result as if evaluated  
3922 with the logical function "urn:oasis:names:tc:xacml:1.0:function:or" with two arguments  
3923 containing the functions "urn:oasis:names:tc:xacml:1.0:function:string-greater-than" and  
3924 "urn:oasis:names:tc:xacml:1.0:function:string-equal" containing the original arguments

3925 • string-less-than  
3926 This function SHALL take two arguments of data-type  
3927 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
3928 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if and only if the  
3929 arguments are compared byte by byte and, after an initial prefix of corresponding bytes  
3930 from both arguments are considered equal by  
3931 "urn:oasis:names:tc:xacml:1.0:function:integer-equal", the next byte by byte comparison is  
3932 such that the byte from the first argument is less than the byte from the second argument  
3933 by the use of the function "urn:oasis:names:tc:xacml:1.0:function:integer-less-than".

- 3934 • string-less-than-or-equal
  - 3935 This function SHALL take two arguments of data-type
  - 3936 “http://www.w3.org/2001/XMLSchema#string” and SHALL return an
  - 3937 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return a result as if evaluated
  - 3938 with the function “urn:oasis:names:tc:xacml:1.0:function:or” with two arguments containing
  - 3939 the functions “urn:oasis:names:tc:xacml:1.0:function:string-less-than” and
  - 3940 “urn:oasis:names:tc:xacml:1.0:function:string-equal” containing the original arguments.
  
- 3941 • time-greater-than
  - 3942 This function SHALL take two arguments of data-type
  - 3943 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
  - 3944 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
  - 3945 argument is greater than the second argument according to the order relation specified for
  - 3946 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
  
- 3947 • time-greater-than-or-equal
  - 3948 This function SHALL take two arguments of data-type
  - 3949 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
  - 3950 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
  - 3951 argument is greater than or equal to the second argument according to the order relation
  - 3952 specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
  
- 3953 • time-less-than
  - 3954 This function SHALL take two arguments of data-type
  - 3955 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
  - 3956 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
  - 3957 argument is less than the second argument according to the order relation specified for
  - 3958 “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
  
- 3959 • time-less-than-or-equal
  - 3960 This function SHALL take two arguments of data-type
  - 3961 “http://www.w3.org/2001/XMLSchema#time” and SHALL return an
  - 3962 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
  - 3963 argument is less than or equal to the second argument according to the order relation
  - 3964 specified for “http://www.w3.org/2001/XMLSchema#time” [XS Section 3.2.8].
  
- 3965 • dateTime-greater-than
  - 3966 This function SHALL take two arguments of data-type
  - 3967 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
  - 3968 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
  - 3969 argument is greater than the second argument according to the order relation specified for
  - 3970 “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].
  
- 3971 • dateTime-greater-than-or-equal
  - 3972 This function SHALL take two arguments of data-type
  - 3973 “http://www.w3.org/2001/XMLSchema#dateTime” and SHALL return an
  - 3974 “http://www.w3.org/2001/XMLSchema#boolean”. It SHALL return "True" if the first
  - 3975 argument is greater than or equal to the second argument according to the order relation
  - 3976 specified for “http://www.w3.org/2001/XMLSchema#dateTime” [XS Section 3.2.7].
  
- 3977 • dateTime-less-than

3978 This function SHALL take two arguments of data-type  
3979 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
3980 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3981 argument is less than the second argument according to the order relation specified for  
3982 "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

3983

- 3984 • dateTime-less-than-or-equal

3985 This function SHALL take two arguments of data-type  
3986 "http://www.w3.org/2001/XMLSchema#dateTime" and SHALL return an  
3987 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3988 argument is less than or equal to the second argument according to the order relation  
3989 specified for "http://www.w3.org/2001/XMLSchema#dateTime" [XS Section 3.2.7].

- 3990 • date-greater-than

3991 This function SHALL take two arguments of data-type  
3992 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
3993 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
3994 argument is greater than the second argument according to the order relation specified for  
3995 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- 3996 • date-greater-than-or-equal

3997 This function SHALL take two arguments of data-type  
3998 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
3999 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
4000 argument is greater than or equal to the second argument according to the order relation  
4001 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- 4002 • date-less-than

4003 This function SHALL take two arguments of data-type  
4004 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4005 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
4006 argument is less than the second argument according to the order relation specified for  
4007 "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

- 4008 • date-less-than-or-equal

4009 This function SHALL take two arguments of data-type  
4010 "http://www.w3.org/2001/XMLSchema#date" and SHALL return an  
4011 "http://www.w3.org/2001/XMLSchema#boolean". It SHALL return "True" if the first  
4012 argument is less than or equal to the second argument according to the order relation  
4013 specified for "http://www.w3.org/2001/XMLSchema#date" [XS Section 3.2.9].

## 4014 A14.9 Bag functions

4015 These functions operate on a *bag* of *type* values, where *data-type* is one of the primitive types. In  
4016 an expression that contains any of these functions, if any argument is "Indeterminate", then the  
4017 expression SHALL evaluate to "Indeterminate". Some additional conditions defined for each  
4018 function below SHALL cause the expression to evaluate to "Indeterminate".

- 4019 • type-one-and-only

4020 This function SHALL take an argument of a **bag** of *type* values and SHALL return a value  
4021 of *data-type*. It SHALL return the only value in the **bag**. If the **bag** does not have one and  
4022 only one value, then the expression SHALL evaluate to "Indeterminate".

4023 • *type-bag-size*

4024 This function SHALL take a **bag** of *type* values as an argument and SHALL return an  
4025 "http://www.w3.org/2001/XMLSchema#integer" indicating the number of values in the **bag**.

4026

4027

4028 • *type-is-in*

4029 This function SHALL take an argument of data-type *type* as the first argument and a **bag** of  
4030 *type* values as the second argument. The expression SHALL evaluate to "True" if the first  
4031 argument matches by the "urn:oasis:names:tc:xacml:1.0:function:type-equal" to any value  
4032 in the **bag**.

4033 • *type-bag*

4034 This function SHALL take any number of arguments of a single data-type and return a **bag**  
4035 of *type* values containing the values of the arguments. An application of this function to  
4036 zero arguments SHALL produce an empty **bag** of the specified data-type.

## 4037 A14.10 Set functions

4038 These functions operate on **bags** mimicking sets by eliminating duplicate elements from a **bag**. In  
4039 an expression that contains any of these functions, if any argument is "Indeterminate", then the  
4040 expression SHALL evaluate to "Indeterminate".

4041 • *type-intersection*

4042 This function SHALL take two arguments that are both a **bag** of *type* values. The  
4043 expression SHALL return a **bag** of *type* values such that it contains only elements that are  
4044 common between the two **bags**, which is determined by  
4045 "urn:oasis:names:tc:xacml:1.0:function:type-equal". No duplicates as determined by  
4046 "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL exist in the result.

4047 • *type-at-least-one-member-of*

4048 This function SHALL take two arguments that are both a **bag** of *type* values. The  
4049 expression SHALL evaluate to "True" if at least one element of the first argument is  
4050 contained in the second argument as determined by  
4051 "urn:oasis:names:tc:xacml:1.0:function:type-is-in".

4052 • *type-union*

4053 This function SHALL take two arguments that are both a **bag** of *type* values. The  
4054 expression SHALL return a **bag** of *type* such that it contains all elements of both **bags**. No  
4055 duplicates as determined by "urn:oasis:names:tc:xacml:1.0:function:type-equal" SHALL  
4056 exist in the result.

4057 • *type-subset*

4058 This function SHALL take two arguments that are both a **bag** of *type* values. It SHALL  
4059 return "True" if the first argument is a subset of the second argument. Each argument is  
4060 considered to have its duplicates removed as determined by  
4061 "urn:oasis:names:tc:xacml:1.0:function:type-equal" before subset calculation.

- 4062 • *type-set-equals*
- 4063 This function SHALL take two arguments that are both a *bag* of *type* values and SHALL
- 4064 return the result of applying "urn:oasis:names:tc:xacml:1.0:function:and" to the application
- 4065 of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the first and second arguments
- 4066 and the application of "urn:oasis:names:tc:xacml:1.0:function:type-subset" to the second
- 4067 and first arguments.

## 4068 **A14.11 Higher-order bag functions**

4069 This section describes functions in XACML that perform operations on *bags* such that functions  
4070 may be applied to the *bags* in general.

4071 In this section, a general-purpose functional language called Haskell [**Haskell**] is used to formally  
4072 specify the semantics of these functions. Although the English description is adequate, a formal  
4073 specification of the semantics is helpful.

4074 For a quick summary, in the following Haskell notation, a function definition takes the form of  
4075 clauses that are applied to patterns of structures, namely lists. The symbol "[]" denotes the empty  
4076 list, whereas the expression "(x:xs)" matches against an argument of a non-empty list of which "x"  
4077 represents the first element of the list, and "xs" is the rest of the list, which may be an empty list. We  
4078 use the Haskell notion of a list, which is an ordered collection of elements, to model the XACML  
4079 *bags* of values.

4080 A simple Haskell definition of a familiar function "urn:oasis:names:tc:xacml:1.0:function:and" that  
4081 takes a list of booleans is defined as follows:

4082         and:: [Bool] -> Bool

4083         and []         = "True"

4084         and (x:xs) = x && (and xs)

4085 The first definition line denoted by a "::" formally describes the data-type of the function, which takes  
4086 a list of booleans, denoted by "[Bool]", and returns a boolean, denoted by "Bool". The second  
4087 definition line is a clause that states that the function "and" applied to the empty list is "True". The  
4088 second definition line is a clause that states that for a non-empty list, such that the first element is  
4089 "x", which is a value of data-type Bool, the function "and" applied to x SHALL be combined with,  
4090 using the logical conjunction function, which is denoted by the infix symbol "&&", the result of  
4091 recursively applying the function "and" to the rest of the list. Of course, an application of the "and"  
4092 function is "True" if and only if the list to which it is applied is empty or every element of the list is  
4093 "True". For example, the evaluation of the following Haskell expressions,

4094         (and []), (and ["True"]), (and ["True","True"]), (and ["True","True","False"])

4095 evaluate to "True", "True", "True", and "False", respectively.

4096 In an expression that contains any of these functions, if any argument is "Indeterminate", then the  
4097 expression SHALL evaluate to "Indeterminate".

- 4098 • *any-of*
- 4099 This function applies a boolean function between a specific primitive value and a *bag* of
- 4100 values, and SHALL return "True" if and only if the predicate is "True" for at least one
- 4101 element of the *bag*.
- 4102 This function SHALL take three arguments. The first argument SHALL be a <Function>
- 4103 element that names a boolean function that takes two arguments of primitive types. The
- 4104 second argument SHALL be a value of a primitive data-type. The third argument SHALL

4105 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function  
4106 named in the <Function> element is applied to the second argument and each element  
4107 of the third argument (the **bag**) and the results are combined with  
4108 "urn:oasis:names:tc:xacml:1.0:function:or".

4109 In Haskell, the semantics of this operation are as follows:

```
4110 any_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4111 any_of f a [] = "False"
4112 any_of f a (x:xs) = (f a x) || (any_of f a xs)
```

4113 In the above notation, "f" is the function name to be applied, "a" is the primitive value, and  
4114 "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

4115 For example, the following expression SHALL return "True":

```
4116 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
4117 <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal" />
4118 <AttributeValue
4119 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4120 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4121 <AttributeValue
4122 DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4123 <AttributeValue
4124 DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4125 <AttributeValue
4126 DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4127 <AttributeValue
4128 DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4129 </Apply>
4130 </Apply>
```

4131 This expression is "True" because the first argument is equal to at least one of the  
4132 elements of the **bag**.

4133 • all-of

4134 This function applies a boolean function between a specific primitive value and a **bag** of  
4135 values, and returns "True" if and only if the predicate is "True" for every element of the **bag**.

4136 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4137 element that names a boolean function that takes two arguments of primitive types. The  
4138 second argument SHALL be a value of a primitive data-type. The third argument SHALL  
4139 be a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function  
4140 named in the <Function> element were applied to the second argument and each  
4141 element of the third argument (the **bag**) and the results were combined using  
4142 "urn:oasis:names:tc:xacml:1.0:function:and".

4143 In Haskell, the semantics of this operation are as follows:

```
4144 all_of :: ( a -> b -> Bool ) -> a -> [b] -> Bool
4145 all_of f a [] = "False"
4146 all_of f a (x:xs) = (f a x) && (all_of f a xs)
```

4147 In the above notation, "f" is the function name to be applied, "a" is the primitive value, and  
4148 "(x:xs)" represents the first element of the list as "x" and the rest of the list as "xs".

4149 For example, the following expression SHALL evaluate to "True":

```

4150 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
4151   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4152   greater"/>
4153   <AttributeValue
4154   DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4155   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4156     <AttributeValue
4157     DataType="http://www.w3.org/2001/XMLSchema#integer">9</AttributeValue>
4158     <AttributeValue
4159     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4160     <AttributeValue
4161     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4162     <AttributeValue
4163     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4164   </Apply>
4165 </Apply>

```

4166 This expression is "True" because the first argument is greater than *all* of the elements of  
4167 the **bag**.

4168 • any-of-any

4169 This function applies a boolean function between each element of a **bag** of values and  
4170 each element of another **bag** of values, and returns "True" if and only if the predicate is  
4171 "True" for at least one comparison.

4172 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4173 element that names a boolean function that takes two arguments of primitive types. The  
4174 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be  
4175 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function  
4176 named in the <Function> element were applied between *every* element in the second  
4177 argument and *every* element of the third argument (the **bag**) and the results were  
4178 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the  
4179 result of the expression SHALL be "True" if and only if the applied predicate is "True" for  
4180 *any* comparison of elements from the two **bags**.

4181 In Haskell, taking advantage of the "any\_of" function defined above, the semantics of the  
4182 "any\_of\_any" function are as follows:

```

4183 any_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4184 any_of_any f [] ys = "False"
4185 any_of_any f (x:xs) ys = (any_of f x ys) || (any_of_any f xs ys)

```

4186 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4187 element of the list as "x" and the rest of the list as "xs".

4188 For example, the following expression SHALL evaluate to "True":

```

4189 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
4190   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
4191   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4192     <AttributeValue
4193       DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4194     <AttributeValue
4195       DataType="http://www.w3.org/2001/XMLSchema#string">Mary</AttributeValue>
4196     </Apply>
4197     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4198       <AttributeValue
4199         DataType="http://www.w3.org/2001/XMLSchema#string">John</AttributeValue>
4200       <AttributeValue
4201         DataType="http://www.w3.org/2001/XMLSchema#string">Paul</AttributeValue>
4202       <AttributeValue
4203         DataType="http://www.w3.org/2001/XMLSchema#string">George</AttributeValue>
4204       <AttributeValue
4205         DataType="http://www.w3.org/2001/XMLSchema#string">Ringo</AttributeValue>
4206     </Apply>
4207 </Apply>

```

4208       This expression is "True" because at least one of the elements of the first **bag**, namely  
4209       "Ringo", is equal to at least one of the string values of the second **bag**.

4210 • all-of-any

4211       This function applies a boolean function between the elements of two **bags**. The  
4212       expression is "True" if and only if the predicate is "True" between each and all of the  
4213       elements of the first **bag** collectively against at least one element of the second **bag**.

4214       This function SHALL take three arguments. The first argument SHALL be a <Function>  
4215       element that names a boolean function that takes two arguments of primitive types. The  
4216       second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be  
4217       a **bag** of a primitive data-type. The expression SHALL be evaluated as if function named in  
4218       the <Function> element were applied between every element in the second argument  
4219       and every element of the third argument (the **bag**) using  
4220       "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the  
4221       expression SHALL be "True" if and only if the applied predicate is "True" for each element  
4222       of the first **bag** and any element of the second **bag**.

4223       In Haskell, taking advantage of the "any\_of" function defined in Haskell above, the  
4224       semantics of the "all\_of\_any" function are as follows:

```

4225               all_of_any :: ( a -> b -> Bool ) -> [a ]-> [b ]-> Bool
4226               all_of_any f []     ys = "False"
4227               all_of_any f (x:xs) ys = (any_of f x ys) && (all_of_any f xs ys)

```

4228       In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4229       element of the list as "x" and the rest of the list as "xs".

4230       For example, the following expression SHALL evaluate to "True":

```

4231 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-any">
4232   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4233   greater"/>
4234   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4235     <AttributeValue
4236     DataType="http://www.w3.org/2001/XMLSchema#integer">10</AttributeValue>
4237     <AttributeValue
4238     DataType="http://www.w3.org/2001/XMLSchema#integer">20</AttributeValue>
4239   </Apply>
4240   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4241     <AttributeValue
4242     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4243     <AttributeValue
4244     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4245     <AttributeValue
4246     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4247     <AttributeValue
4248     DataType="http://www.w3.org/2001/XMLSchema#integer">21</AttributeValue>
4249   </Apply>
4250 </Apply>

```

4251 This expression is "True" because all of the elements of the first **bag**, each "10" and "20",  
4252 are greater than at least one of the integer values "1", "3", "5", "21" of the second **bag**.

4253 • any-of-all

4254 This function applies a boolean function between the elements of two **bags**. The  
4255 expression SHALL be "True" if and only if the predicate is "True" between at least one of  
4256 the elements of the first **bag** collectively against all the elements of the second **bag**.

4257 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4258 element that names a boolean function that takes two arguments of primitive types. The  
4259 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be  
4260 a **bag** of a primitive data-type. The expression SHALL be evaluated as if the function  
4261 named in the <Function> element were applied between *every* element in the second  
4262 argument and *every* element of the third argument (the **bag**) and the results were  
4263 combined using "urn:oasis:names:tc:xacml:1.0:function:or". The semantics are that the  
4264 result of the expression SHALL be "True" if and only if the applied predicate is "True" for  
4265 *any* element of the first **bag** compared to *all* the elements of the second **bag**.

4266 In Haskell, taking advantage of the "all\_of" function defined in Haskell above, the semantics  
4267 of the "any\_of\_all" function are as follows:

```

4268 any_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4269 any_of_all f [] ys = "False"
4270 any_of_all f (x:xs) ys = (all_of f x ys) || ( any_of_all f xs ys)

```

4271 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4272 element of the list as "x" and the rest of the list as "xs".

4273 For example, the following expression SHALL evaluate to "True":

```

4274 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-all">
4275   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4276   greater"/>
4277   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4278     <AttributeValue
4279     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4280     <AttributeValue
4281     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4282   </Apply>
4283   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4284     <AttributeValue
4285     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4286     <AttributeValue
4287     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4288     <AttributeValue
4289     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4290     <AttributeValue
4291     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4292   </Apply>
4293 </Apply>

```

4294 This expression is "True" because at least one element of the first **bag**, namely "5", is  
4295 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4296 • all-of-all

4297 This function applies a boolean function between the elements of two **bags**. The  
4298 expression SHALL be "True" if and only if the predicate is "True" between each and all of  
4299 the elements of the first **bag** collectively against all the elements of the second **bag**.

4300 This function SHALL take three arguments. The first argument SHALL be a <Function>  
4301 element that names a boolean function that takes two arguments of primitive types. The  
4302 second argument SHALL be a **bag** of a primitive data-type. The third argument SHALL be  
4303 a **bag** of a primitive data-type. The expression is evaluated as if the function named in the  
4304 <Function> element were applied between *every* element in the second argument and  
4305 *every* element of the third argument (the **bag**) and the results were combined using  
4306 "urn:oasis:names:tc:xacml:1.0:function:and". The semantics are that the result of the  
4307 expression is "True" if and only if the applied predicate is "True" for *all* elements of the first  
4308 **bag** compared to *all* the elements of the second **bag**.

4309 In Haskell, taking advantage of the "all\_of" function defined in Haskell above, the semantics  
4310 of the "all\_of\_all" function is as follows:

```

4311 all_of_all :: ( a -> b -> Bool ) -> [a ]-> [b ] -> Bool
4312 all_of_all f [] ys = "False"
4313 all_of_all f (x:xs) ys = (all_of f x ys) && (all_of_all f xs ys)

```

4314 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4315 element of the list as "x" and the rest of the list as "xs".

4316 For example, the following expression SHALL evaluate to "True":

```

4317 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of-all">
4318   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-
4319   greater"/>
4320   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4321     <AttributeValue
4322     DataType="http://www.w3.org/2001/XMLSchema#integer">6</AttributeValue>
4323     <AttributeValue
4324     DataType="http://www.w3.org/2001/XMLSchema#integer">5</AttributeValue>
4325   </Apply>
4326   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-bag">
4327     <AttributeValue
4328     DataType="http://www.w3.org/2001/XMLSchema#integer">1</AttributeValue>
4329     <AttributeValue
4330     DataType="http://www.w3.org/2001/XMLSchema#integer">2</AttributeValue>
4331     <AttributeValue
4332     DataType="http://www.w3.org/2001/XMLSchema#integer">3</AttributeValue>
4333     <AttributeValue
4334     DataType="http://www.w3.org/2001/XMLSchema#integer">4</AttributeValue>
4335   </Apply>
4336 </Apply>

```

4337 This expression is "True" because all elements of the first **bag**, "5" and "6", are each  
4338 greater than all of the integer values "1", "2", "3", "4" of the second **bag**.

4339 • map

4340 This function converts a **bag** of values to another **bag** of values.

4341 This function SHALL take two arguments. The first function SHALL be a <Function>  
4342 element naming a function that takes a single argument of a primitive data-type and returns  
4343 a value of a primitive data-type. The second argument SHALL be a **bag** of a primitive data-  
4344 type. The expression SHALL be evaluated as if the function named in the <Function>  
4345 element were applied to each element in the **bag** resulting in a **bag** of the converted value.  
4346 The result SHALL be a **bag** of the primitive data-type that is the same data-type that is  
4347 returned by the function named in the <Function> element.

4348 In Haskell, this function is defined as follows:

```

4349     map:: (a -> b) -> [a] -> [b]
4350     map f []     = []
4351     map f (x:xs) = (f x) : (map f xs)

```

4352 In the above notation, "f" is the function name to be applied and "(x:xs)" represents the first  
4353 element of the list as "x" and the rest of the list as "xs".

4354 For example, the following expression,

```

4355 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:map">
4356   <Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
4357   normalize-to-lower-case">
4358   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
4359     <AttributeValue
4360     DataType="http://www.w3.org/2001/XMLSchema#string">Hello</AttributeValue>
4361     <AttributeValue
4362     DataType="http://www.w3.org/2001/XMLSchema#string">World!</AttributeValue>
4363   </Apply>
4364 </Apply>

```

4365 evaluates to a **bag** containing "hello" and "world!".

4366

## A14.12 Special match functions

4367 These functions operate on various types and evaluate to  
4368 "http://www.w3.org/2001/XMLSchema#boolean" based on the specified standard matching  
4369 algorithm. In an expression that contains any of these functions, if any argument is "Indeterminate",  
4370 then the expression SHALL evaluate to "Indeterminate".

4371 • regex-string-match

4372 This function decides a regular expression match. It SHALL take two arguments of  
4373 "http://www.w3.org/2001/XMLSchema#string" and SHALL return an  
4374 "http://www.w3.org/2001/XMLSchema#boolean". The first argument SHALL be a regular  
4375 expression and the second argument SHALL be a general string. The function  
4376 specification SHALL be that of the "xf:matches" function with the arguments reversed [XF  
4377 Section 6.3.15].

4378 • x500Name-match

4379 This function shall take two arguments of "urn:oasis:names:tc:xacml:1.0:data-  
4380 type:x500Name" and shall return an "http://www.w3.org/2001/XMLSchema#boolean". It  
4381 shall return "True" if and only if the first argument matches some terminal sequence of  
4382 RDNs from the second argument when compared using x500Name-equal.

4383 • rfc822Name-match

4384 This function SHALL take two arguments, the first is of data-type  
4385 "http://www.w3.org/2001/XMLSchema#string" and the second is of data-type  
4386 "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name" and SHALL return an  
4387 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if  
4388 the first argument matches the second argument according to the following specification.

4389 An RFC822 name consists of a local-part followed by "@" followed by domain-part. The  
4390 local-part is case-sensitive, while the domain-part (which is usually a DNS name) is not  
4391 case-sensitive.<sup>4</sup>

4392 The second argument contains a complete rfc822Name. The first argument is a complete  
4393 or partial rfc822Name used to select appropriate values in the second argument as follows.

4394 In order to match a particular mailbox in the second argument, the first argument must  
4395 specify the complete mail address to be matched. For example, if the first argument is  
4396 "Anderson@sun.com", this matches a value in the second argument of  
4397 "Anderson@sun.com" and "Anderson@SUN.COM", but not "Anne.Anderson@sun.com",  
4398 "anderson@sun.com" or "Anderson@east.sun.com".

4399 In order to match any mail address at a particular domain in the second argument, the first  
4400 argument must specify only a domain name (usually a DNS name). For example, if the first  
4401 argument is "sun.com", this matches a value in the first argument of "Anderson@sun.com"  
4402 or "Baxter@SUN.COM", but not "Anderson@east.sun.com".

4403 In order to match any mail address in a particular domain in the second argument, the first  
4404 argument must specify the desired domain-part with a leading ".". For example, if the first  
4405 argument is ".east.sun.com", this matches a value in the second argument of

---

4 According to IETF RFC822 and its successor specifications [RFC2821], case is significant in the *local-part*. Many mail systems, as well as the IETF PKIX specification, treat the *local-part* as case-insensitive. This anomaly is considered an error by mail-system designers and is not encouraged. For this reason, rfc822Name-match treats *local-part* as case sensitive.

4406 "Anderson@east.sun.com" and "anne.anderson@ISRG.EAST.SUN.COM" but not  
4407 "Anderson@sun.com".

## 4408 **A14.13 XPath-based functions**

4409 This section specifies functions that take XPath expressions for arguments. An XPath expression  
4410 evaluates to a *node-set*, which is a set of XML nodes that match the expression. A node or node-  
4411 set is not in the formal data-type system of XACML. All comparison or other operations on node-  
4412 sets are performed in the isolation of the particular function specified. The XPath expressions in  
4413 these functions are restricted to the XACML request *context*. The `<xacml-context:Request>`  
4414 element is a context node for every XPath expression. The following functions are defined:

- 4415 • xpath-node-count

4416 This function SHALL take an "http://www.w3.org/2001/XMLSchema#string" as an  
4417 argument, which SHALL be interpreted as an XPath expression, and evaluates to an  
4418 "http://www.w3.org/2001/XMLSchema#integer". The value returned from the function  
4419 SHALL be the count of the nodes within the node-set that matches the given XPath  
4420 expression.

- 4421 • xpath-node-equal

4422 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments,  
4423 which SHALL be interpreted as XPath expressions, and SHALL return an  
4424 "http://www.w3.org/2001/XMLSchema#boolean". The function SHALL return "True" if any  
4425 XML node from the node-set matched by the first argument equals according to the  
4426 "op:node-equal" function [XF Section 13.1.6] any XML node from the node-set matched by  
4427 the second argument.

- 4428 • xpath-node-match

4429 This function SHALL take two "http://www.w3.org/2001/XMLSchema#string" arguments, which  
4430 SHALL be interpreted as XPath expressions and SHALL return an  
4431 "http://www.w3.org/2001/XMLSchema#boolean". This function SHALL evaluate to "True" if  
4432 either of the following two conditions is satisfied: (1) Any XML node from the node-set matched  
4433 by the first argument is equal according to "op:node-equal" [XF Section 13.1.6] to any XML node  
4434 from the node-set matched by the second argument. (2) Any attribute and element node below  
4435 any XML node from the node-set matched by the first argument is equal according to "op:node-  
4436 equal" [XF Section 13.1.6] to any XML node from the node-set matched by the second  
4437 argument.

4438 NOTE: The first condition is equivalent to "xpath-node-equal", and guarantees that "xpath-node-  
4439 equal" is a special case of "xpath-node-match".

## 4440 **A14.14 Extension functions and primitive types**

4441 Functions and primitive types are specified by string identifiers allowing for the introduction of  
4442 functions in addition to those specified by XACML. This approach allows one to extend the XACML  
4443 module with special functions and special primitive data-types.

4444 In order to preserve some integrity to the XACML evaluation strategy, the result of all function  
4445 applications SHALL depend only on the values of its arguments. Global and hidden parameters  
4446 SHALL NOT affect the evaluation of an expression. Functions SHALL NOT have side effects, as  
4447 evaluation order cannot be guaranteed in a standard way.

---

## 4448 Appendix B. XACML identifiers (normative)

4449 This section defines standard identifiers for commonly used entities. All XACML-defined identifiers  
4450 have the common base:

4451 `urn:oasis:names:tc:xacml:1.0`

### 4452 B.1. XACML namespaces

4453 There are currently two defined XACML namespaces.

4454 Policies are defined using this identifier.

4455 `urn:oasis:names:tc:xacml:1.0:policy`

4456 Request and response **contexts** are defined using this identifier.

4457 `urn:oasis:names:tc:xacml:1.0:context`

### 4458 B.2. Access subject categories

4459 This identifier indicates the system entity that initiated the **access** request. That is, the initial entity  
4460 in a request chain. If **subject** category is not specified, this is the default value.

4461 `urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`

4462 This identifier indicates the system entity that will receive the results of the request. Used when it is  
4463 distinct from the access-subject.

4464 `urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject`

4465 This identifier indicates a system entity through which the **access** request was passed. There may  
4466 be more than one. No means is provided to specify the order in which they passed the message.

4467 `urn:oasis:names:tc:xacml:1.0:subject-category:intermediary-subject`

4468 This identifier indicates a system entity associated with a local or remote codebase that generated  
4469 the request. Corresponding **subject attributes** might include the URL from which it was loaded  
4470 and/or the identity of the code-signer. There may be more than one. No means is provided to  
4471 specify the order they processed the request.

4472 `urn:oasis:names:tc:xacml:1.0:subject-category:codebase`

4473 This identifier indicates a system entity associated with the computer that initiated the **access**  
4474 request. An example would be an IPsec identity.

4475 `urn:oasis:names:tc:xacml:1.0:subject-category:requesting-machine`

### 4476 B.3. XACML functions

4477 This identifier is the base for all the identifiers in the table of functions. See Section A.1.

4478 `urn:oasis:names:tc:xacml:1.0:function`

### 4479 B.4. Data-types

4480 The following identifiers indicate useful data-types.

4481 X.500 distinguished name

4482 urn:oasis:names:tc:xacml:1.0:data-type:x500Name

4483 An x500Name contains an ITU-T Rec. X.520 Distinguished Name. The valid syntax for such a  
 4484 name is described in IETF RFC 2253 "Lightweight Directory Access Protocol (v3): UTF-8 String  
 4485 Representation of Distinguished Names".

4486 RFC822 Name

4487 urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name

4488 An rfc822Name contains an "e-mail name". The valid syntax for such a name is described in IETF  
 4489 RFC 2821, Section 4.1.2, Command Argument Syntax, under the term "Mailbox".

4490 The following data-type identifiers are defined by XML Schema.

4491 http://www.w3.org/2001/XMLSchema#string  
 4492 http://www.w3.org/2001/XMLSchema#boolean  
 4493 http://www.w3.org/2001/XMLSchema#integer  
 4494 http://www.w3.org/2001/XMLSchema#double  
 4495 http://www.w3.org/2001/XMLSchema#time  
 4496 http://www.w3.org/2001/XMLSchema#date  
 4497 http://www.w3.org/2001/XMLSchema#dateTime  
 4498 http://www.w3.org/2001/XMLSchema#anyURI  
 4499 http://www.w3.org/2001/XMLSchema#hexBinary  
 4500 http://www.w3.org/2001/XMLSchema#base64Binary

4501 The following data-type identifiers correspond to the dayTimeDuration and yearMonthDuration  
 4502 data-types defined in [XF Sections 8.2.2 and 8.2.1, respectively].

4503 http://www.w3.org/TR/2002/WD-xquery-operators-20020816#dayTimeDuration  
 4504 http://www.w3.org/TR/2002/WD-xquery-operators-20020816#yearMonthDuration

## 4505 B.5. Subject attributes

4506 These identifiers indicate **attributes** of a **subject**. When used, they SHALL appear within a  
 4507 <Subject> element of the request **context**. They SHALL be accessed via a  
 4508 <SubjectAttributeDesignator> or an <AttributeSelector> element pointing into a  
 4509 <Subject> element of the request **context**.

4510 At most one of each of these attributes is associated with each subject. Each attribute associated  
 4511 with authentication included within a single <Subject> element relates to the same authentication  
 4512 event.

4513 This identifier indicates the name of the **subject**. The default format is  
 4514 http://www.w3.org/2001/XMLSchema#string. To indicate other formats, use `DataType` attributes  
 4515 listed in B.4

4516 urn:oasis:names:tc:xacml:1.0:subject:subject-id

4517 This identifier indicates the **subject** category. "access-subject" is the default.

4518 urn:oasis:names:tc:xacml:1.0:subject-category

4519 This identifier indicates the security domain of the **subject**. It identifies the administrator and policy  
 4520 that manages the name-space in which the **subject** id is administered.

4521 urn:oasis:names:tc:xacml:1.0:subject:subject-id-qualifier

4522 This identifier indicates a public key used to confirm the **subject's** identity.

4523 urn:oasis:names:tc:xacml:1.0:subject:key-info

4524 This identifier indicates the time at which the **subject** was authenticated.

4525 urn:oasis:names:tc:xacml:1.0:subject:authentication-time

4526 This identifier indicates the method used to authenticate the **subject**.

4527 urn:oasis:names:tc:xacml:1.0:subject:authentication-method

4528 This identifier indicates the time at which the **subject** initiated the **access** request, according to the  
4529 **PEP**.

4530 urn:oasis:names:tc:xacml:1.0:subject:request-time

4531 This identifier indicates the time at which the **subject's** current session began, according to the  
4532 **PEP**.

4533 urn:oasis:names:tc:xacml:1.0:subject:session-start-time

4534 The following identifiers indicate the location where authentication credentials were activated. They  
4535 are intended to support the corresponding entities from the SAML authentication statement.

4536 This identifier indicates that the location is expressed as an IP address.

4537 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:ip-address

4538 This identifier indicates that the location is expressed as a DNS name.

4539 urn:oasis:names:tc:xacml:1.0:subject:authn-locality:dns-name

4540 Where a suitable attribute is already defined in LDAP [**LDAP-1**, **LDAP-2**], the XACML identifier  
4541 SHALL be formed by adding the **attribute** name to the URI of the LDAP specification. For  
4542 example, the **attribute** name for the userPassword defined in the rfc2256 SHALL be:

4543 http://www.ietf.org/rfc/rfc2256.txt#userPassword

## 4544 B.6. Resource attributes

4545 These identifiers indicate **attributes** of the **resource**. When used, they SHALL appear within the  
4546 <Resource> element of the request **context**. They SHALL be accessed via a  
4547 <ResourceAttributeDesignator> or an <AttributeSelector> element pointing into the  
4548 <Resource> element of the request **context**.

4549 This identifier indicates the entire URI of the **resource**.

4550 urn:oasis:names:tc:xacml:1.0:resource:resource-id

4551 A **resource attribute** used to indicate values extracted from the **resource**.

4552 urn:oasis:names:tc:xacml:1.0:resource:resource-content

4553 This identifier indicates the last (rightmost) component of the file name. For example, if the URI is:  
4554 "file://home/my/status#pointer", the simple-file-name is "status".

4555 urn:oasis:names:tc:xacml:1.0:resource:simple-file-name

4556 This identifier indicates that the **resource** is specified by an XPath expression.

4557 urn:oasis:names:tc:xacml:1.0:resource:xpath

4558 This identifier indicates a UNIX file-system path.

4559 urn:oasis:names:tc:xacml:1.0:resource:ufs-path

4560 This identifier indicates the scope of the **resource**, as described in Section 7.8.

4561 urn:oasis:names:tc:xacml:1.0:resource:scope

4562 The allowed value for this attribute is of data-type http://www.w3.org/2001/XMLSchema#string, and  
4563 is either "Immediate", "Children" or "Descendants".

## 4564 B.7. Action attributes

4565 These identifiers indicate **attributes** of the **action** being requested. When used, they SHALL  
4566 appear within the <Action> element of the request **context**. They SHALL be accessed via an  
4567 <ActionAttributeDesignator> or an <AttributeSelector> element pointing into the  
4568 <Action> element of the request **context**.

4569 urn:oasis:names:tc:xacml:1.0:action:action-id  
4570 Action namespace  
4571 urn:oasis:names:tc:xacml:1.0:action:action-namespace  
4572 Implied action. This is the value for action-id attribute when action is implied.  
4573 urn:oasis:names:tc:xacml:1.0:action:implied-action

## 4574 B.8. Environment attributes

4575 These identifiers indicate *attributes* of the *environment* within which the *decision request* is to be  
4576 evaluated. When used in the *decision request*, they SHALL appear in the <Environment>  
4577 element of the request *context*. They SHALL be accessed via an  
4578 <EnvironmentAttributeDesignator> or an <AttributeSelector> element pointing into  
4579 the <Environment> element of the request *context*.

4580 This identifier indicates the current time at the *PDP*. In practice it is the time at which the request  
4581 *context* was created.

4582 urn:oasis:names:tc:xacml:1.0:environment:current-time  
4583 urn:oasis:names:tc:xacml:1.0:environment:current-date  
4584 urn:oasis:names:tc:xacml:1.0:environment:current-dateTime

## 4585 B.9. Status codes

4586 The following status code identifiers are defined.

4587 This identifier indicates success.

4588 urn:oasis:names:tc:xacml:1.0:status:ok

4589 This identifier indicates that attributes necessary to make a policy decision were not available.

4590 urn:oasis:names:tc:xacml:1.0:status:missing-attribute

4591 This identifier indicates that some attribute value contained a syntax error, such as a letter in a  
4592 numeric field.

4593 urn:oasis:names:tc:xacml:1.0:status:syntax-error

4594 This identifier indicates that an error occurred during policy evaluation. An example would be  
4595 division by zero.

4596 urn:oasis:names:tc:xacml:1.0:status:processing-error

## 4597 B.10. Combining algorithms

4598 The deny-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4599 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides

4600 The deny-overrides policy-combining algorithm has the following value for  
4601 policyCombiningAlgId:

4602 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides

4603 The permit-overrides rule-combining algorithm has the following value for ruleCombiningAlgId:

4604 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides

4605 The permit-overrides policy-combining algorithm has the following value for  
4606 policyCombiningAlgId:

4607 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:permit-overrides

4608 The first-applicable rule-combining algorithm has the following value for ruleCombiningAlgId:  
4609 urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable  
4610 The first-applicable policy-combining algorithm has the following value for  
4611 policyCombiningAlgId:  
4612 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable  
4613 The only-one-applicable-policy policy-combining algorithm has the following value for  
4614 policyCombiningAlgId:  
4615 urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable  
4616 The ordered-deny-overrides rule-combining algorithm has the following value for  
4617 ruleCombiningAlgId:  
4618 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-deny-overrides  
4619  
4620 The ordered-deny-overrides policy-combining algorithm has the following value for  
4621 policyCombiningAlgId:  
4622 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-deny-overrides  
4623  
4624 The ordered-permit-overrides rule-combining algorithm has the following value for  
4625 ruleCombiningAlgId:  
4626 urn:oasis:names:tc:xacml:1.1:rule-combining-algorithm:ordered-permit-overrides  
4627  
4628 The ordered-permit-overrides policy-combining algorithm has the following value for  
4629 policyCombiningAlgId:  
4630 urn:oasis:names:tc:xacml:1.1:policy-combining-algorithm:ordered-permit-overrides

4631

## Appendix C. Combining algorithms (normative)

4632  
4633

This section contains a description of the rule-combining and policy-combining algorithms specified by XACML.

4634

### C.1. Deny-overrides.

4635

The following specification defines the “Deny-overrides” *rule-combining algorithm* of a *policy*.

4636  
4637  
4638  
4639  
4640  
4641

In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Deny", then the result of the *rule* combination SHALL be "Deny". If any *rule* evaluates to "Permit" and all other *rules* evaluate to "NotApplicable", then the result of the *rule* combination SHALL be "Permit". In other words, "Deny" takes precedence, regardless of the result of evaluating any of the other *rules* in the combination. If all *rules* are found to be "NotApplicable" to the *decision request*, then the *rule* combination SHALL evaluate to "NotApplicable".

4642  
4643  
4644  
4645

If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect* value of "Deny" then the evaluation SHALL continue to evaluate subsequent *rules*, looking for a result of "Deny". If no other *rule* evaluates to "Deny", then the combination SHALL evaluate to "Indeterminate", with the appropriate error status.

4646  
4647  
4648

If at least one *rule* evaluates to "Permit", all other *rules* that do not have evaluation errors evaluate to "Permit" or "NotApplicable" and all *rules* that do have evaluation errors contain *effects* of "Permit", then the result of the combination SHALL be "Permit".

4649

The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

4650  
4651  
4652  
4653  
4654  
4655  
4656  
4657  
4658  
4659  
4660  
4661  
4662  
4663  
4664  
4665  
4666  
4667  
4668  
4669  
4670  
4671  
4672  
4673  
4674  
4675  
4676  
4677  
4678  
4679

```
Decision denyOverridesRuleCombiningAlgorithm(Rule rule[])
{
    Boolean atLeastOneError = false;
    Boolean potentialDeny = false;
    Boolean atLeastOnePermit = false;
    for( i=0 ; i < lengthOf(rules) ; i++ )
    {
        Decision decision = evaluate(rule[i]);
        if (decision == Deny)
        {
            return Deny;
        }
        if (decision == Permit)
        {
            atLeastOnePermit = true;
            continue;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate)
        {
            atLeastOneError = true;

            if (effect(rule[i]) == Deny)
            {
                potentialDeny = true;
            }
            continue;
        }
    }
}
```

```

4680     }
4681     }
4682     if (potentialDeny)
4683     {
4684         return Indeterminate;
4685     }
4686     if (atLeastOnePermit)
4687     {
4688         return Permit;
4689     }
4690     if (atLeastOneError)
4691     {
4692         return Indeterminate;
4693     }
4694     return NotApplicable;
4695 }

```

4696 The following specification defines the “Deny-overrides” *policy-combining algorithm* of a *policy*  
4697 **set**.

4698 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Deny", then the  
4699 result of the *policy* combination SHALL be "Deny". In other words, "Deny" takes  
4700 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*  
4701 **set**. If all *policies* are found to be "NotApplicable" to the *decision request*, then the  
4702 *policy set* SHALL evaluate to "NotApplicable".

4703 If an error occurs while evaluating the *target* of a *policy*, or a reference to a *policy* is  
4704 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*  
4705 SHALL evaluate to "Deny".

4706 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4707 Decision denyOverridesPolicyCombiningAlgorithm(Policy policy[])
4708 {
4709     Boolean atLeastOnePermit = false;
4710     for( i=0 ; i < lengthOf(policy) ; i++ )
4711     {
4712         Decision decision = evaluate(policy[i]);
4713         if (decision == Deny)
4714         {
4715             return Deny;
4716         }
4717         if (decision == Permit)
4718         {
4719             atLeastOnePermit = true;
4720             continue;
4721         }
4722         if (decision == NotApplicable)
4723         {
4724             continue;
4725         }
4726         if (decision == Indeterminate)
4727         {
4728             return Deny;
4729         }
4730     }
4731     if (atLeastOnePermit)
4732     {
4733         return Permit;
4734     }
4735     return NotApplicable;
4736 }

```

4737 **Obligations** of the individual *policies* shall be combined as described in Section 7.11.

4738

## C.2. Ordered-deny-overrides (non-normative)

4739 The following specification defines the "Ordered-deny-overrides" *rule-combining algorithm* of a  
4740 *policy*.

4741 The behavior of this algorithm is identical to that of the Deny-overrides *rule-combining*  
4742 *algorithm* with one exception. The order in which the collection of *rules* is evaluated SHALL  
4743 match the order as listed in the *policy*.

4744 The following specification defines the "Ordered-deny-overrides" *policy-combining algorithm* of a  
4745 *policy set*.

4746 The behavior of this algorithm is identical to that of the Deny-overrides *policy-combining*  
4747 *algorithm* with one exception. The order in which the collection of *policies* is evaluated SHALL  
4748 match the order as listed in *the policy set*.

4749

## C.3. Permit-overrides

4750 The following specification defines the "Permit-overrides" *rule-combining algorithm* of a *policy*.

4751 In the entire set of *rules* in the *policy*, if any *rule* evaluates to "Permit", then the result of  
4752 the *rule* combination SHALL be "Permit". If any *rule* evaluates to "Deny" and all other  
4753 *rules* evaluate to "NotApplicable", then the *policy* SHALL evaluate to "Deny". In other  
4754 words, "Permit" takes precedence, regardless of the result of evaluating any of the other  
4755 *rules* in the *policy*. If all *rules* are found to be "NotApplicable" to the *decision request*,  
4756 then the *policy* SHALL evaluate to "NotApplicable".

4757 If an error occurs while evaluating the *target* or *condition* of a *rule* that contains an *effect*  
4758 of "Permit" then the evaluation SHALL continue looking for a result of "Permit". If no other  
4759 *rule* evaluates to "Permit", then the *policy* SHALL evaluate to "Indeterminate", with the  
4760 appropriate error status.

4761 If at least one *rule* evaluates to "Deny", all other *rules* that do not have evaluation errors  
4762 evaluate to "Deny" or "NotApplicable" and all *rules* that do have evaluation errors contain  
4763 an *effect* value of "Deny", then the *policy* SHALL evaluate to "Deny".

4764 The following pseudo-code represents the evaluation strategy of this *rule-combining algorithm*.

```
4765 Decision permitOverridesRuleCombiningAlgorithm(Rule rule[])
4766 {
4767     Boolean atLeastOneError = false;
4768     Boolean potentialPermit = false;
4769     Boolean atLeastOneDeny = false;
4770     for( i=0 ; i < lengthOf(rule) ; i++ )
4771     {
4772         Decision decision = evaluate(rule[i]);
4773         if (decision == Deny)
4774         {
4775             atLeastOneDeny = true;
4776             continue;
4777         }
4778         if (decision == Permit)
4779         {
4780             return Permit;
4781         }
4782         if (decision == NotApplicable)
4783         {
4784             continue;
```

```

4785     }
4786     if (decision == Indeterminate)
4787     {
4788         atLeastOneError = true;
4789
4790         if (effect(rule[i]) == Permit)
4791         {
4792             potentialPermit = true;
4793         }
4794         continue;
4795     }
4796 }
4797 if (potentialPermit)
4798 {
4799     return Indeterminate;
4800 }
4801 if (atLeastOneDeny)
4802 {
4803     return Deny;
4804 }
4805 if (atLeastOneError)
4806 {
4807     return Indeterminate;
4808 }
4809 return NotApplicable;
4810 }

```

4811 The following specification defines the "Permit-overrides" *policy-combining algorithm* of a *policy*  
4812 *set*.

4813 In the entire set of *policies* in the *policy set*, if any *policy* evaluates to "Permit", then the  
4814 result of the *policy* combination SHALL be "Permit". In other words, "Permit" takes  
4815 precedence, regardless of the result of evaluating any of the other *policies* in the *policy*  
4816 *set*. If all *policies* are found to be "NotApplicable" to the *decision request*, then the  
4817 *policy set* SHALL evaluate to "NotApplicable".

4818 If an error occurs while evaluating the *target* of a *policy*, a reference to a *policy* is  
4819 considered invalid or the *policy* evaluation results in "Indeterminate", then the *policy set*  
4820 SHALL evaluate to "Indeterminate", with the appropriate error status, provided no other  
4821 *policies* evaluate to "Permit" or "Deny".

4822 The following pseudo-code represents the evaluation strategy of this *policy-combining algorithm*.

```

4823 Decision permitOverridesPolicyCombiningAlgorithm(Policy policy[])
4824 {
4825     Boolean atLeastOneError = false;
4826     Boolean atLeastOneDeny = false;
4827     for( i=0 ; i < lengthOf(policy) ; i++ )
4828     {
4829         Decision decision = evaluate(policy[i]);
4830         if (decision == Deny)
4831         {
4832             atLeastOneDeny = true;
4833             continue;
4834         }
4835         if (decision == Permit)
4836         {
4837             return Permit;
4838         }
4839         if (decision == NotApplicable)
4840         {
4841             continue;
4842         }

```

```

4843     if (decision == Indeterminate)
4844     {
4845         atLeastOneError = true;
4846         continue;
4847     }
4848 }
4849 if (atLeastOneDeny)
4850 {
4851     return Deny;
4852 }
4853 if (atLeastOneError)
4854 {
4855     return Indeterminate;
4856 }
4857 return NotApplicable;
4858 }

```

4859 **Obligations** of the individual policies shall be combined as described in Section 7.11.

## 4860 C.4. Ordered-permit-overrides (non-normative)

4861 The following specification defines the "Ordered-permit-overrides" **rule-combining algorithm** of a  
4862 **policy**.

4863 The behavior of this algorithm is identical to that of the Permit-overrides **rule-combining**  
4864 **algorithm** with one exception. The order in which the collection of **rules** is evaluated SHALL  
4865 match the order as listed in the **policy**.

4866 The following specification defines the "Ordered-permit-overrides" **policy-combining algorithm** of  
4867 a **policy set**.

4868 The behavior of this algorithm is identical to that of the Permit-overrides **policy-combining**  
4869 **algorithm** with one exception. The order in which the collection of **policies** is evaluated  
4870 SHALL match the order as listed in the **policy set**.

## 4871 C.5. First-applicable

4872 The following specification defines the "First-Applicable " **rule-combining algorithm** of a **policy**.

4873 Each **rule** SHALL be evaluated in the order in which it is listed in the **policy**. For a  
4874 particular **rule**, if the **target** matches and the **condition** evaluates to "True", then the  
4875 evaluation of the **policy** SHALL halt and the corresponding **effect** of the **rule** SHALL be the  
4876 result of the evaluation of the **policy** (i.e. "Permit" or "Deny"). For a particular **rule** selected  
4877 in the evaluation, if the **target** evaluates to "False" or the **condition** evaluates to "False",  
4878 then the next **rule** in the order SHALL be evaluated. If no further **rule** in the order exists,  
4879 then the **policy** SHALL evaluate to "NotApplicable".

4880 If an error occurs while evaluating the **target** or **condition** of a **rule**, then the evaluation  
4881 SHALL halt, and the **policy** shall evaluate to "Indeterminate", with the appropriate error  
4882 status.

4883 The following pseudo-code represents the evaluation strategy of this **rule-combining algorithm**.

```

4884 Decision firstApplicableEffectRuleCombiningAlgorithm(Rule rule[])
4885 {
4886     for( i = 0 ; i < lengthOf(rule) ; i++ )
4887     {

```

```

4888     Decision decision = evaluate(rule[i]);
4889     if (decision == Deny)
4890     {
4891         return Deny;
4892     }
4893     if (decision == Permit)
4894     {
4895         return Permit;
4896     }
4897     if (decision == NotApplicable)
4898     {
4899         continue;
4900     }
4901     if (decision == Indeterminate)
4902     {
4903         return Indeterminate;
4904     }
4905 }
4906 return NotApplicable;
4907 }

```

4908 The following specification defines the “First-applicable” **policy-combining algorithm** of a **policy**  
4909 **set**.

4910 Each **policy** is evaluated in the order that it appears in the **policy set**. For a particular  
4911 **policy**, if the **target** evaluates to "True" and the **policy** evaluates to a determinate value of  
4912 "Permit" or "Deny", then the evaluation SHALL halt and the **policy set** SHALL evaluate to  
4913 the **effect** value of that **policy**. For a particular **policy**, if the **target** evaluate to "False", or  
4914 the **policy** evaluates to "NotApplicable", then the next **policy** in the order SHALL be  
4915 evaluated. If no further **policy** exists in the order, then the **policy set** SHALL evaluate to  
4916 "NotApplicable".

4917 If an error were to occur when evaluating the **target**, or when evaluating a specific **policy**,  
4918 the reference to the **policy** is considered invalid, or the **policy** itself evaluates to  
4919 "Indeterminate", then the evaluation of the **policy-combining algorithm** shall halt, and the  
4920 **policy set** shall evaluate to "Indeterminate" with an appropriate error status.

4921 The following pseudo-code represents the evaluation strategy of this **policy-combination**  
4922 **algorithm**.

```

4923 Decision firstApplicableEffectPolicyCombiningAlgorithm(Policy policy[])
4924 {
4925     for( i = 0 ; i < lengthOf(policy) ; i++ )
4926     {
4927         Decision decision = evaluate(policy[i]);
4928         if(decision == Deny)
4929         {
4930             return Deny;
4931         }
4932         if(decision == Permit)
4933         {
4934             return Permit;
4935         }
4936         if (decision == NotApplicable)
4937         {
4938             continue;
4939         }
4940         if (decision == Indeterminate)
4941         {
4942             return Indeterminate;
4943         }
4944     }
4945     return NotApplicable;

```

4946

```
}
```

4947

**Obligations** of the individual policies shall be combined as described in Section 7.11.

4948

## C.6. Only-one-applicable

4949

The following specification defines the "Only-one-applicable" **policy-combining algorithm** of a **policy set**.

4950

4951

In the entire set of policies in the **policy set**, if no **policy** is considered applicable by virtue of their **targets**, then the result of the policy combination algorithm SHALL be "NotApplicable". If more than one policy is considered applicable by virtue of their **targets**, then the result of the policy combination algorithm SHALL be "Indeterminate".

4952

4953

4954

4955

If only one **policy** is considered applicable by evaluation of the **policy targets**, then the result of the **policy-combining algorithm** SHALL be the result of evaluating the **policy**.

4956

4957

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.

4958

4959

4960

The following pseudo-code represents the evaluation strategy of this policy combining algorithm.

4961

```
Decision onlyOneApplicablePolicyPolicyCombiningAlgorithm(Policy policy[])
```

4962

```
{
```

4963

```
    Boolean          atLeastOne      = false;
```

4964

```
    Policy           selectedPolicy = null;
```

4965

```
    ApplicableResult appResult;
```

4966

```
    for ( i = 0; i < lengthOf(policy) ; i++ )
```

4967

```
    {
```

4968

```
        appResult = isApplicable(policy[i]);
```

4969

4970

```
        if ( appResult == Indeterminate )
```

4971

```
        {
```

4972

```
            return Indeterminate;
```

4973

```
        }
```

4974

```
        if( appResult == Applicable )
```

4975

```
        {
```

4976

```
            if ( atLeastOne )
```

4977

```
            {
```

4978

```
                return Indeterminate;
```

4979

```
            }
```

4980

```
            else
```

4981

```
            {
```

4982

```
                atLeastOne      = true;
```

4983

```
                selectedPolicy = policy[i];
```

4984

```
            }
```

4985

```
        }
```

4986

```
        if ( appResult == NotApplicable )
```

4987

```
        {
```

4988

```
            continue;
```

4989

```
        }
```

4990

```
    }
```

4991

```
    if ( atLeastOne )
```

4992

```
    {
```

4993

```
        return evaluate(selectedPolicy);
```

4994

```
    }
```

4995

```
    else
```

4996

```
    {
```

4997

```
        return NotApplicable;
```

4998

```
    }
```

4999  
5000  
5001



---

5002 **Appendix D. Acknowledgments**

5003 The following individuals contributed to the development of the specification:

5004 Anne Anderson  
5005 Bill Parducci  
5006 Carlisle Adams  
5007 Daniel Engovatov  
5008 Don Flinn  
5009 Ernesto Damiani  
5010 Gerald Brose  
5011 Hal Lockhart  
5012 James MacLean  
5013 John Merrells  
5014 Ken Yagen  
5015 Konstantin Beznosov  
5016 Michiharu Kudo  
5017 Pierangela Samarati  
5018 Pirasenna Velandai Thiyagarajan  
5019 Polar Humenn  
5020 Satoshi Hada  
5021 Sekhar Vajjhala  
5022 Seth Proctor  
5023 Simon Godik  
5024 Steve Anderson  
5025 Steve Crocker  
5026 Suresh Damodaran  
5027 Tim Moses  
5028

5029

---

## Appendix E. Revision history

Rev	Date	By whom	What
OS V1.0	18 Feb 2003	XACML Technical Committee	OASIS Standard

5030

---

5031

## Appendix F. Notices

5032 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
5033 that might be claimed to pertain to the implementation or use of the technology described in this  
5034 document or the extent to which any license under such rights might or might not be available;  
5035 neither does it represent that it has made any effort to identify any such rights. Information on  
5036 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
5037 website. Copies of claims of rights made available for publication and any assurances of licenses to  
5038 be made available, or the result of an attempt made to obtain a general license or permission for  
5039 the use of such proprietary rights by implementors or users of this specification, can be obtained  
5040 from the OASIS Executive Director.

5041 OASIS has been notified of intellectual property rights claimed in regard to some or all of the  
5042 contents of this specification. For more information consult the online list of claimed rights.

5043 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
5044 applications, or other proprietary rights which may cover technology that may be required to  
5045 implement this specification. Please address the information to the OASIS Executive Director.

5046 Copyright (C) OASIS Open 2003. All Rights Reserved.

5047 This document and translations of it may be copied and furnished to others, and derivative works  
5048 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
5049 published and distributed, in whole or in part, without restriction of any kind, provided that the above  
5050 copyright notice and this paragraph are included on all such copies and derivative works. However,  
5051 this document itself may not be modified in any way, such as by removing the copyright notice or  
5052 references to OASIS, except as needed for the purpose of developing OASIS specifications, in  
5053 which case the procedures for copyrights defined in the OASIS Intellectual Property Rights  
5054 document must be followed, or as required to translate it into languages other than English.

5055 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
5056 successors or assigns.

5057 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
5058 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
5059 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY  
5060 RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
5061 PARTICULAR PURPOSE.